Advanced Machine Learning
(2019 MOD006566 TRI1 F01CAM)

Titanic Survival Prediction

Research Report

SID: 0925739

Contents

Background

The “Titanic Survival Prediction” challenge is a Kaggle Machine learning competition to create a prediction model that will correctly predict the survival of a certain passenger given a set of features for that passenger. Samuel (1959) describes Machine Learning as a field of study that gives computers the ability to learn without being explicitly programmed. Additionally, Mitchell (1998) argues that a computer program is said to learn from experience E with respect to some task T and performance measure P, if its performance on task T, as measured by P, improves with experience E. This report will document the processes behind analysing the dataset of Titanic passengers and creating a learning model using the analysed dataset to make predictions on unseen data.

Data Analysis and Data Processing

The data analysis section of this report explains the measures taken to clean the dataset, engineer new features and pre-process the dataset for use in learning algorithm. The dataset was downloaded from Kaggle and it contained three files: ‘train.csv’, ‘test.csv’ and ‘gender_submission.csv’. Jupyter notebook was used to write the code blocks required for this task. The dataset files were loaded into Pandas dataframes by calling the ‘pandas.read_csv’ command. Pandas is a Python library that provides easy-to-use data structures and data analysis tools. (PyData, 2019) Examining the first five lines of the dataset reveals us the contents of the dataset. Below is the list of columns available on the 3 dataframes that were loaded.

Train.csv

```
['PassengerId',
 'Survived',
 'Pclass',
 'Name',
 'Sex',
 'Age',
 'SibSp',
 'Parch',
 'Ticket',
 'Fare',
 'Cabin',
 'Embarked']
```

Test.csv

```
['PassengerId',
```

```
 'Pclass',
 'Name',
 'Sex',
 'Age',
 'SibSp',
 'Parch',
 'Ticket',
 'Fare',
 'Cabin',
 'Embarked']
```

Gender_submission.csv

```
['PassengerId', 'Survived']
```

The 'gender_submission.csv' is a file that is given as a format to make prediction submission for the competition. This file is irrelevant to our data analysis task and will only be used for creating the final predictions file. The train and test dataset were slightly different in terms of their shape.

```
Shape of train.csv      :  (891, 12)
Shape of test.csv       :  (418, 11)
```

It was clear from the information that 'test.csv' lacks the column 'Survived' which is present in 'train.csv'. Furthermore, 'train.csv' has 891 data points whereas test.csv has 418. Hence, the dataset to use to train the learning model is 'train.csv' (training data) and it needs to be analysed before feeding it into Machine Learning algorithms. The very first step was to examine the data types present in the training set.

```
Training set:
object 5
int64 5
float64 2
dtype: int64
```

As seen from the result obtained, the training data consists of a mix of integer, float and object type data. The integer and float are numerical data types and are suitable to use in learning models, however, the object type data are non-numerical and need to be engineered into numerical data. It was also necessary to examine the amount of missing values in the whole dataset. Pandas' DataFrame.info () command allowed viewing the total data points present in each column.

```
Data columns (total 12 columns):
PassengerId    891 non-null int64
Survived       891 non-null int64
```

```
Pclass         891 non-null int64
Name           891 non-null object
Sex            891 non-null object
Age            714 non-null float64
SibSp          891 non-null int64
Parch          891 non-null int64
Ticket         891 non-null object
Fare           891 non-null float64
Cabin          204 non-null object
Embarked       889 non-null object
```

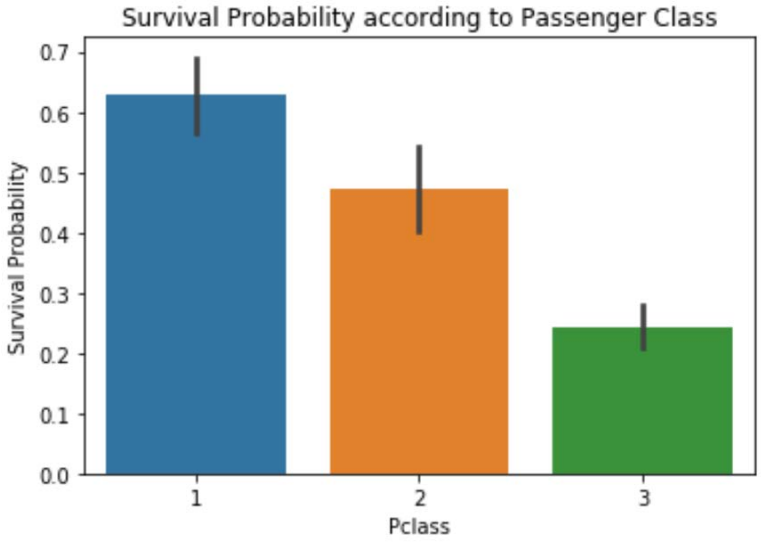
As not all columns had the same number of data points, it was necessary to specifically check the total sum of missing values for each column which was achieved by using DataFrame.isnull().sum() command.

```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
```

The training dataset suffered by missing values in ‘Age’, ‘Cabin’ and ‘Embarked’ columns. A good approach to analysing the data is by looking at the unique values of each column. A small function was created to print basic info of a column and first 50 unique values. This helped understand what each column meant for the purposes of our problem task at hand. Some online research helped find meanings of ambiguous columns like SibSp and Parch.

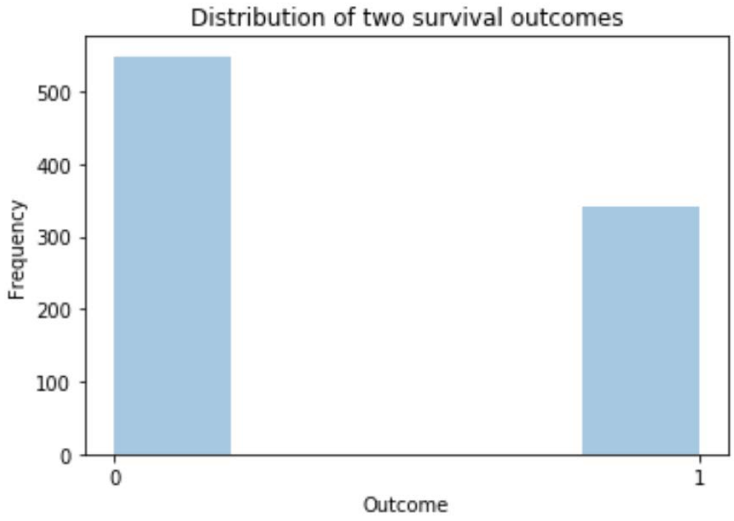
```
PassengerId      : Passenger ID
Survived         : Survival Prediction (1 for survived and 0 otherwise)
Pclass           : Passenger Class
Name             : Name of Passenger
Sex              : Sex of Passenger (M or F)
Age              : Age of Passenger
SibSp            : Siblings/Spouse on Board
Parch            : Parents/Children on Board
Ticket           : Ticket number of the Passenger
Fare             : Ticket fare paid by the Passenger
Cabin            : Cabin number allocated to the Passenger
Embarked         : Post of Embarkation of the Passenger
                   (S = Southampton, C = Cherbourg Q = Queenstown)
```

The 'PassengerID' column has consecutive numbers assigned to each passenger in the dataset which is unlikely to have any effect on survival prediction. The 'PClass' column has 3 unique values assigned to passengers belonging to first, second and third class respectively. Python's seaborn library which is built upon the Matplotlib library was used to create a bar chart showing the survival probability based on the passenger distribution.

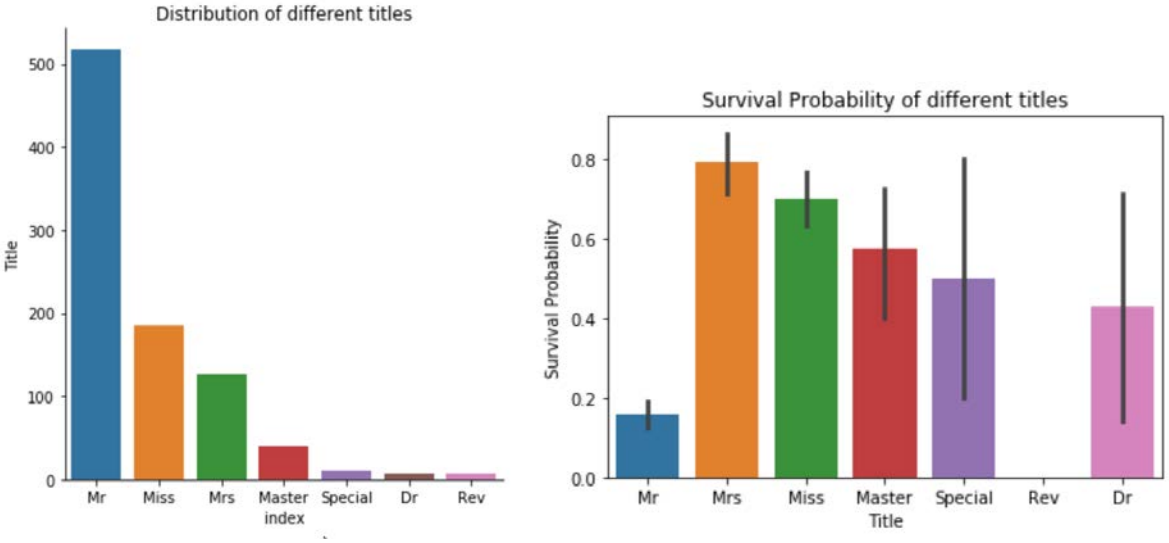


Seaborn aggregates the chart before prediction and since the survival values have only 0 and 1, the survival probability shown above is based on distribution. Passengers travelling in first class have the highest chance of survival while passengers travelling in third class have the lowest chance.

After inspecting the 'Survived' column it is also noticed that more people died than survived in the Titanic.



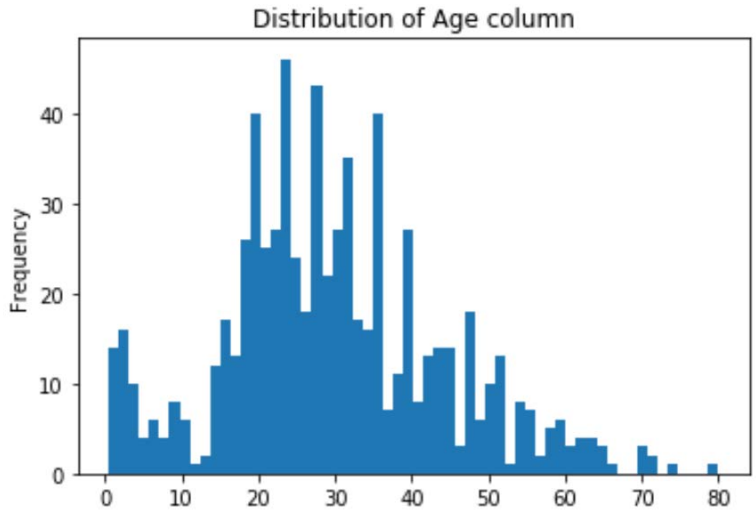
These initial insights are useful to understand how the test predictions should be generally like. Moving on, the 'Name' column had the names of the passengers in Titanic which is not really a factor that would affect survival as it is an object type data. However, every name entry has its title mentioned within their name. So, extracting the titles from the names seemed to be a good idea to group the passengers. This process is known as feature engineering as we are constructing a new column feature for every data point.



The plot above shows how passengers of each title are distributed and what their survival probability is based on the distribution.

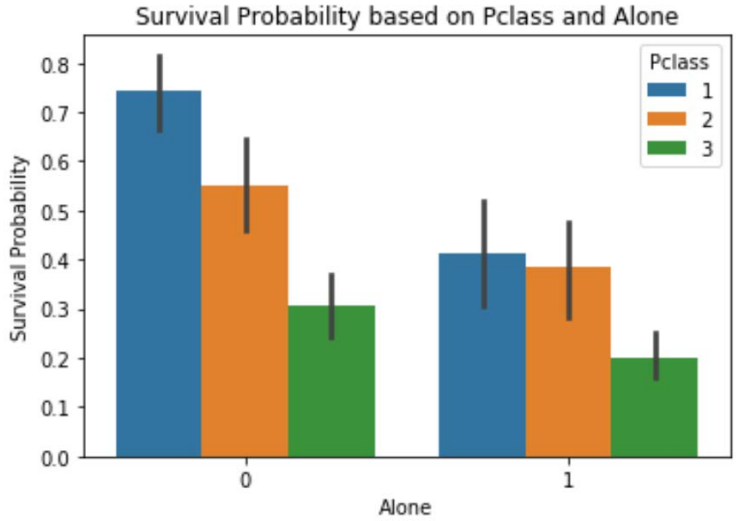
The next column analysed was 'Sex' which represents the gender of a person and has only two unique values, namely 'male' and 'female'. As it is currently an object type column, the best way to convert this to numeric is by changing its values to boolean where 1 represents 'male' and '0' represents 'female'.

Moving on, 'Age' column seemed to have a lot of missing values. The non-null values had an almost normal distribution as shown below.



The 'SibSp' and 'Parch' columns has numerical values that represents the number of Sibling/Spouse on board and Parent/Child on board. It made sense to create a new column 'Family on board' by adding the two columns. Adding meaningful features as such adds further dimension to the data. This could result in the learning algorithm finding new patterns which would not have been realised otherwise.

The next column is the 'Ticket' column that consists of the ticket numbers. The ticket numbers were in varied formats. Some of them had strings and some were totally numerical. As 76% of the values were unique, it did not make sense to group the data based on tickets. However, since some of the ticket numbers were shared by passengers travelling together, it made sense to create a new column that had Boolean values for whether they were travelling alone. A new column 'Alone' was then created using the logic that 'all passengers that have a unique ticket number AND do not have any Family on board are travelling alone'.



The chart above was then created to see the survival probability of passengers in different classes based on whether they were alone. It is observed that for all passenger classes, one was more likely to survive if they were not alone.

The ‘Fare’ column was analysed next which contains the fare paid by passengers. It was interesting to see how the fares compared with Age. And the boxplot shows in great detail the distribution and also the outliers can be seen where two passengers paid more than 500 for their fares.

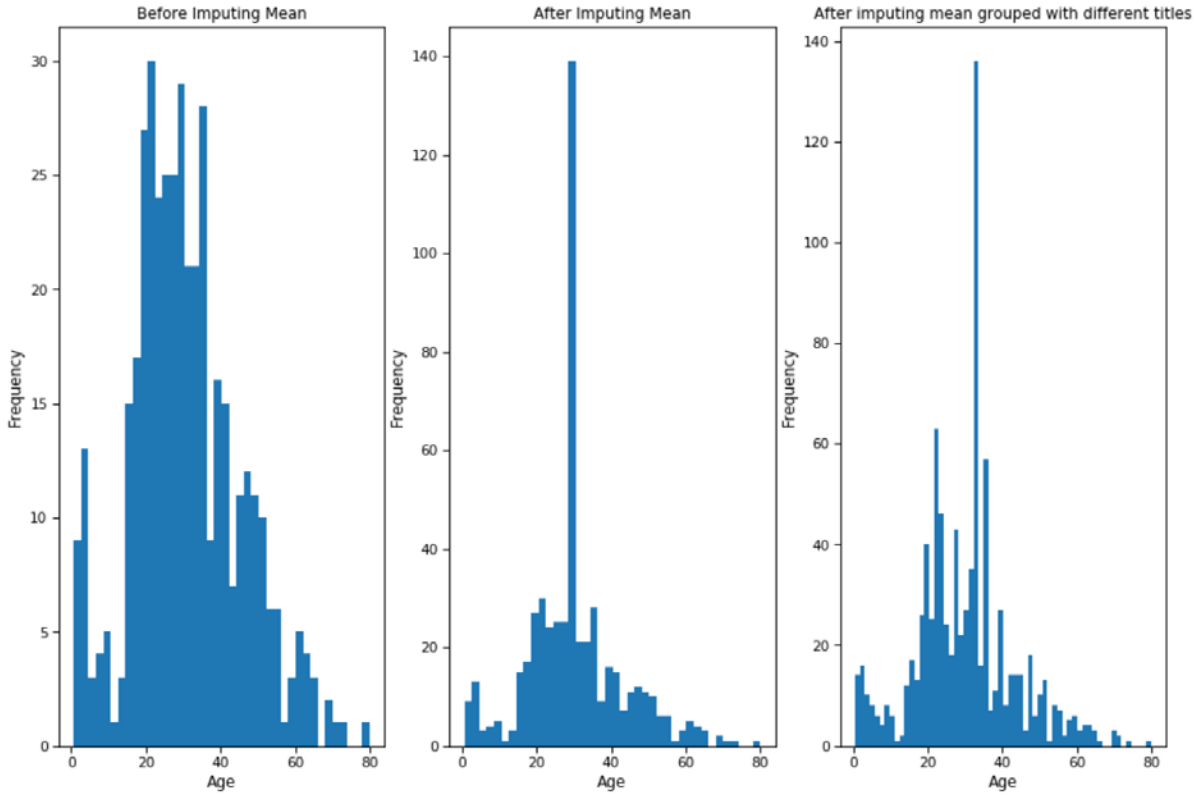
The ‘Cabin’ column has about 77% of missing values and it is an object type data. Feature engineering was considered but since most of the existing values are unique, it was considered best to drop this column entirely. Trying to over-predict the information when most of the data are missing can result into a biased data which will affect the learning model’s prediction capability.

The final column to explore was the ‘Embarked’ column which contained only 3 unique object type values representing the Port of Embarkation. It had 2 values missing which were imputed by the most occurring value among the 3 unique values.

After cleaning the dataset, it was time to get to imputing the values in ‘Age’ column. The impute method required the usage of aggregating the values from all the rows. This is a critical time where a data analyst working in the field of Machine Learning should decide to split the dataset to training and validation sets. This split could be done after the imputation but doing so will cause data leakage from the validation set into the training set. This will increase the chance of the learning model not generalising well on unseen data. Hence a split was made at this point to make test set and validation set using the following command.

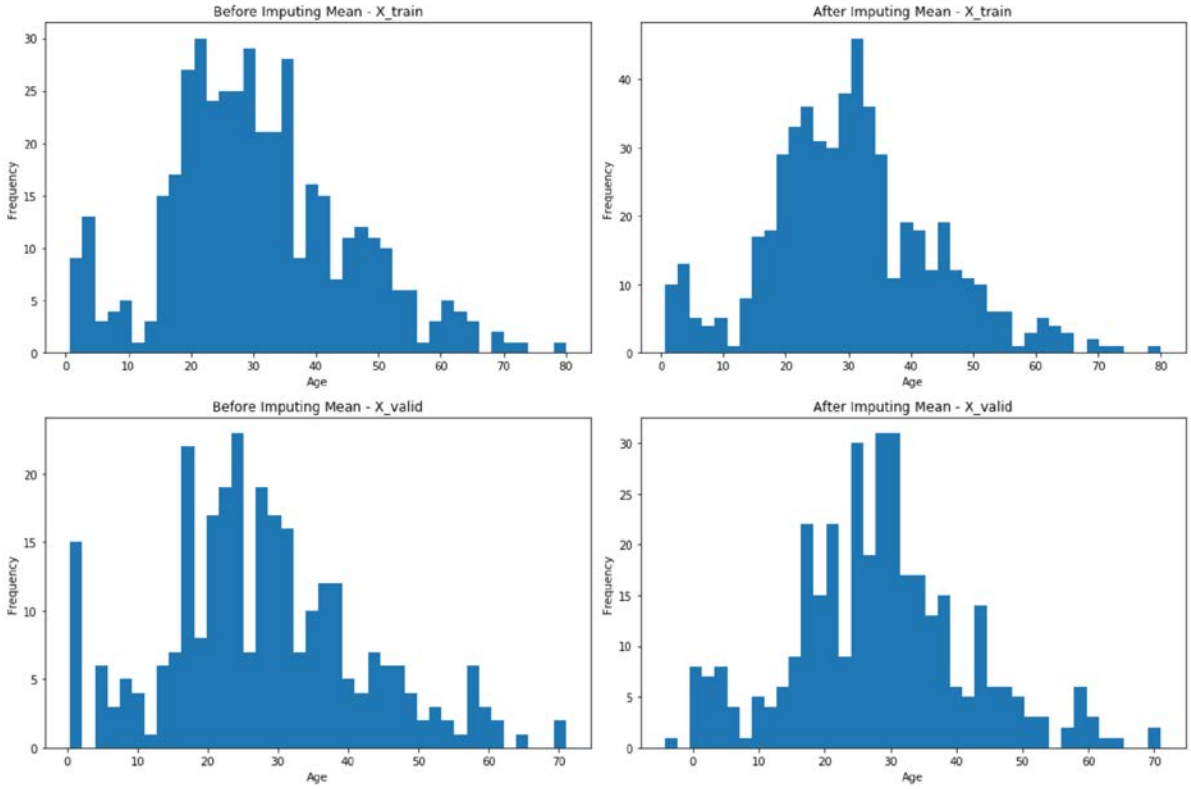
X_train, X_valid, y_train, y_valid = train_test_split(X, y, test_size = 0.4, random_state=1)

After the split, the X_train dataset that will be used for training our model was ready for imputation of missing ‘Age’ values. A widely used method is the mean imputation.



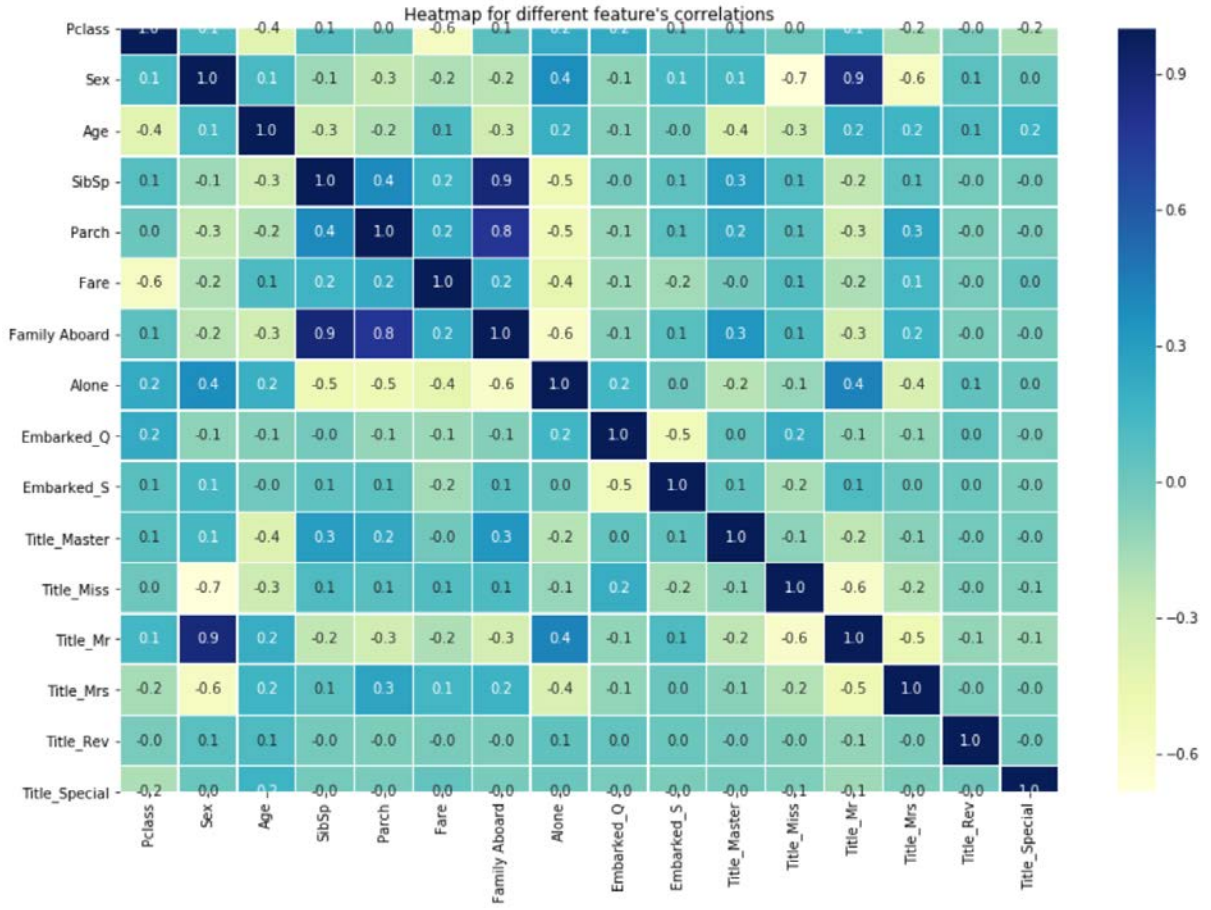
Two different ways of imputing the mean were tried. The first method was to impute all missing values with overall mean of the 'Age' values and the second method was to impute the mean grouped by the passenger's title. Both methods resulted in significant change in the distribution and hence mean imputation was discarded for this problem.

Next method was to impute using some learning algorithm. This meant looking at all other feature values of all data points to predict the missing age values. ScikitLearn's IterativeImputer was good at this problem because it allowed for Multivariate Feature Imputation. The resulting change of distribution for both X_train and X_valid were plotted and the change in distribution was negligible as shown below.



So, the IterativeImputer was kept as the preferred method for missing value imputation.

After the impute, the data was finally ready with all numerical values and a correlation matrix was created to see the relationship between all columns. The correlation matrix was then plotted into a heatmap using Seaborn for better visualisation of the relationships.



Several directly and inversely proportional relationships can be seen above. The heatmap by Seaborn returns a coloured chart with numbers representing the intensity of the relationship. Values range from -1 to 1. A high positive value means that the relationship is directly proportional, and a high negative value means that the relationship is inversely proportional. Even though some relationship values are small, they can project meanings in higher dimensions. An example is the ‘Title_Mr’ column that has a +0.4 value on relationship with the column ‘Alone’. As discussed earlier, ‘Alone’ is a column that is engineered from other columns. And the heatmap shows that a directly proportional relationship exists between being alone and having a title ‘Mr’. This would not have been otherwise seen if not for the feature engineering methods applied.

This concluded the exploratory data analysis aspect of the Titanic Survival Prediction task. At this point, the training and the validation datasets were saved into new csv files. It is important to save copies of data so that we can retrieve the clean version multiple times while training our models. The test dataset was cleaned with the same

methods as applied on the training set and saved as a new file ready for learning algorithms to make predictions on.

Legal, ethical and privacy concerns

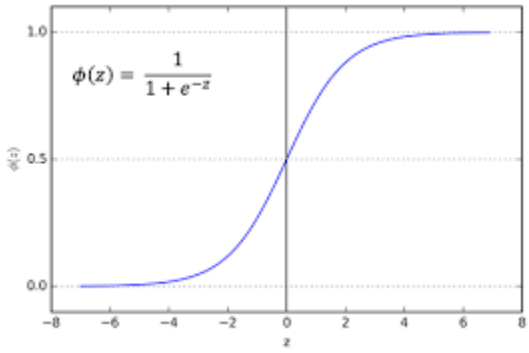
As this data originates from Europe and is old, there were not many data related privacy issues when the Titanic sank, and hence this data was compiled. Today, this dataset serves as a Machine Learning task. However, since the data originates from Europe, if such datasets were to be created, the new datasets would be bound by the General Data Protection regulation (GDPR). GDPR protects individuals from their personal data being used without their consent. The 'Name' column of the Titanic dataset contains the most sensitive information. It contains information using which individuals can be directly identifiable. GDPR recommends anonymising the data before using it so that people are not targeted as a result of the findings of the data analysis. (Gruschka et al., 2018) Any such data collected for a similar task would have to adhere to GDPR regulations.

Classification

For the Component 2 of this task, data was first loaded into dataframes as they were named before saving and appropriately formatted. The first learning algorithm employed was the K-Nearest Neighbour (KNN) algorithm. This is sometimes also referred to as 'the lazy learning algorithm' as it needs to analyse the whole dataset every time it must make a prediction. It is a non-parametric type of learning algorithm because it cannot be characterised by a fixed set of parameters. Unlike parametric models like logistic regression or perceptron, KNN does **not** estimate parameters from the training set to learn a function that can classify new data points. It rather memorizes the dataset and classifies new data points based on how close they are to other classes (Raschka and Mirjalili, 2017). The KNN algorithm showed an accuracy of 80% on the training set and 64.4% on the validation set.

The next learning algorithm to try would be the logistic regression model which makes predictions by learning parameters for the different features to make predictions. Hence this is a parametric model. It maps the sum of the dot product between the features' matrix for each sample and a feature coefficients vector into the sigmoid

function returning values between 0 and 1. (Raschka and Mirjalili, 2017) The figure below shows the function's output plotted.



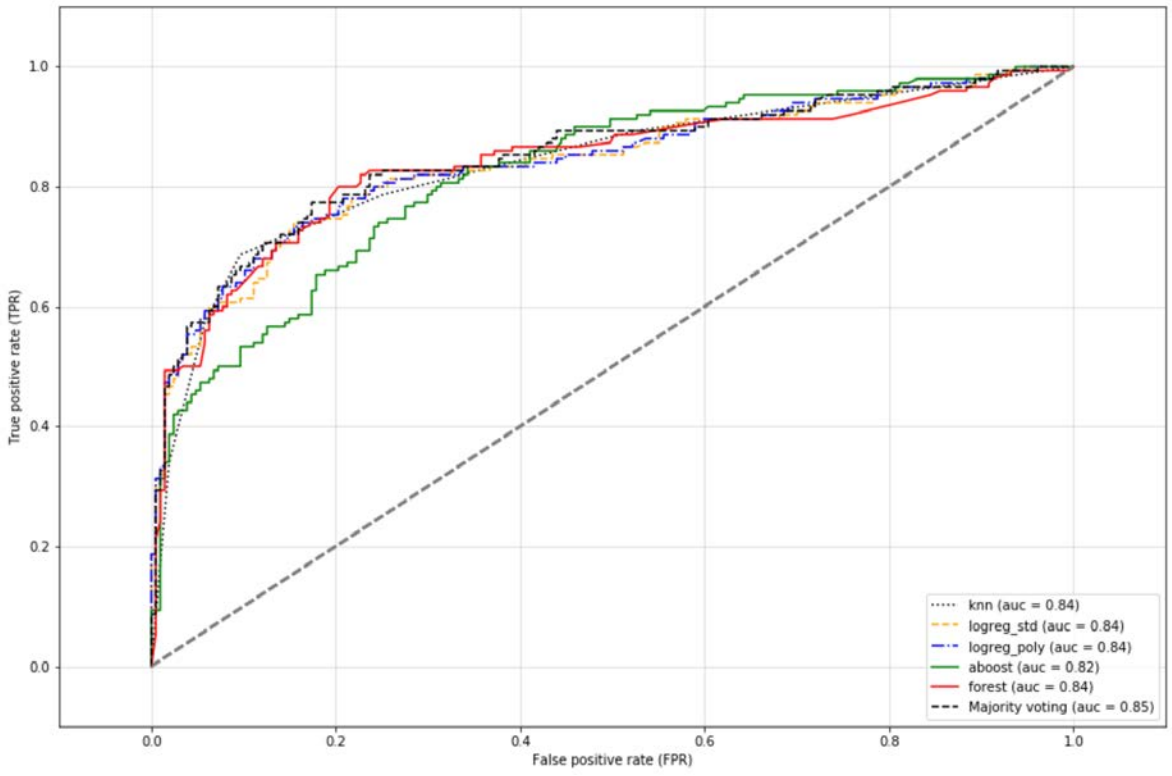
The logistic regression showed the accuracy score of the model's prediction as 84% training accuracy and 79.5% validation accuracy which is better than the KNN algorithm. A further two logistic regression models were tried on the dataset using data scaling using the StandardScaler and by generating polynomial features on the dataset. This increased the training accuracy slightly, but the validation accuracy either remained the same or was compromised slightly.

After logistic regression, another non-parametric model called Support Vector Machine (SVM) was tried on the dataset. SVM separates the data by creating a decision boundary street between classes. Kernel methods allow SVMs to predict datasets projected into higher dimensions. Both 'linear' and 'rbf' kernels were tried on the dataset. (Geron, 2017) Linear kernels work on linearly separable data whereas RBF (Radial Basis Function) kernels are good at non-linearly separable datasets. (Raschka and Mirjalili, 2017) Although, the training accuracy increased to 90% using 'rbf' kernel SVM, the validation accuracy stayed around 76% which is lower compared to the logistic regression model.

Moreover, a decision tree classifier was also applied on the data and its best parameters were learned using GridSearchCV class from scikitLearn. Then a random forest classifier was created using the best parameters learned. Decision tress divide the dataset into categories based on features to create a subdivided tree. New predictions are based on which tree nodes the data features fall towards. Random forest creates multiple trees with different tree creation methods and output a majority voting between the decision trees. The random forest classifier did quite well on the both training and validation sets and produced accuracy of 85.7% and 79.5%

respectively. An AdaBoost classifier was also implemented which pushed the training accuracy to 99.25% however, the test accuracy went down to 74%. This is an example of overfitting where training data performs better than validation data.

Finally, an ensemble of classifiers was created that included KNN, Logistic Regression with standardized data, Logistic Regression with polynomial data, Random Forest classifier and Adaboost. The models that performed better were provided with higher weights than models that performed less. The parameters for models were optimised using GridSearchCV over the majority vote ensemble classifier class called VotingClassifier. Then the final voting classifier with the best parameters was created. Finally, the ROC AUC Accuracy Score was calculated for all classifiers. It was a good sign that almost all classifiers generalised to ~85%. The best performing models were logistic regression models and the majority vote classifier.



As seen in the ROC AUC curve above, the Majority voting classifier worked best among all classifiers in the validation set with 85% ROC AUC score.

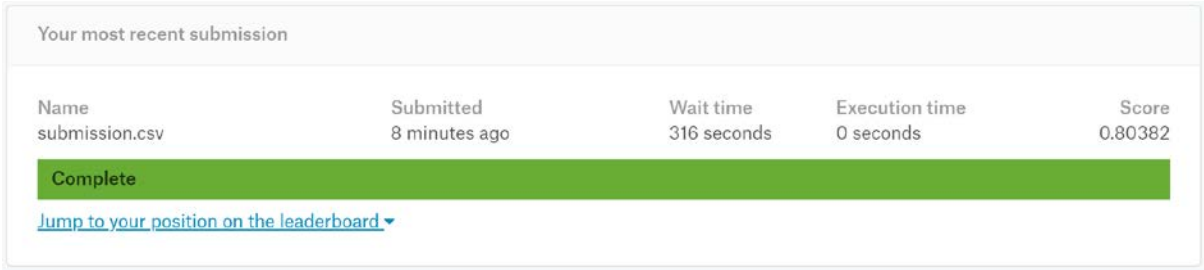
Kaggle Submission and Leader-board

The majority voting classifier was then used to make predictions on the cleaned test dataset. The predictions were then submitted to Kaggle and the accuracy achieved

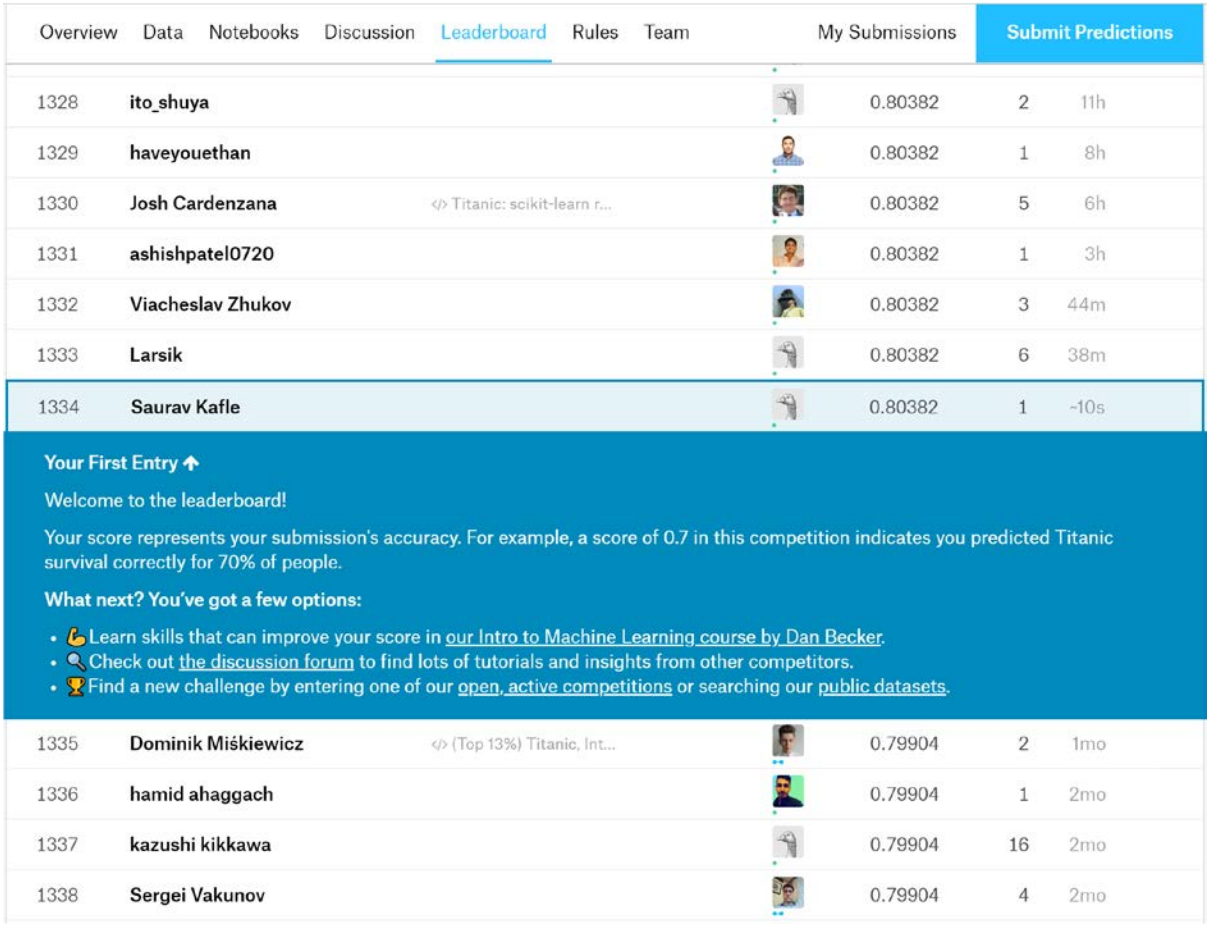
was 80%. A few variations of the classifiers were used to make few additional submissions, but the initial submission came out to be the best estimator. Further improvements on the analysis of this dataset include collecting more data, going into further depth to create engineered features and trying out a greater number of ensemble classifiers.

The submission proof and Leader-board snapshot are as follows:

Submission Proof



Leader-board Snapshot



References

Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, 3(3), 210-229.

Mitchell, T.M., 1997. *Machine Learning*, London: McGraw Hill

Raschka, S. and Mirjalili, V., 2017. *Python Machine Learning*, Birmingham: Packt Publishing

Geron, A., 2017. *Hands-on Machine Learning with Scikit-Learn & TensorFlow*, Sebastopol: O'Reilly Media

Gruschka, N., Mavroeidis, V., Vishi, K. and Jensen, M., 2018. *Privacy Issues and Data Protection in Big Data: A Case Study Analysis under GDPR*, [PDF]. Available at: < https://arxiv.org/pdf/1811.08531.pdf> [Accessed on: 11 December 2019]

PyData, 2019. Pandas: *Python Data Analysis Library*. [online] Available at: <https://pandas.pydata.org/> [Accessed on 12/12/2019]