TITANIC: MACHINE LEARNING FROM DISASTER - ELEN4009 COURSE PROJECT - ARTIFICIAL INTELLIGENCE 2022

Robin Jonker (1827572)

School of Electrical & Information Engineering, University of the Witwatersrand, Private Bag 3, 2050, Johannesburg, South Africa

Abstract: The purpose of this report is to give a breakdown of the work that was done within creating a machine learning program that predicts the survivability of passengers on board the Titanic. As the team lead of the project I was heavily involved in every aspect of the project development, either directly or by giving insight and guidance to my fellow team members. The breakdown below will look at three main sections, namely Data Analysis, Model Building, and Model Tuning/Ensembling, along with an analysis of the results and future improvements. My fellow group members are Tristan de Groot (1355583), Van Niekerk Ferreira (1834038), Tristan Lilford (1843691), Tristan Basel (1862166), and Jesse Iyuke (1606965).

Key words: Data Analysis, Model Building, Model Tuning/Ensembling

1. INTRODUCTION

Machine Learning gives users the power to create models and receive predictions for outcomes that are unheard of before the emergence of Artificial Intelligence. A dataset of over 800 passengers is used to train different models, which are then used to create a prediction of the survivability of over 400 passengers with an unknown survival attribute at a high level of accuracy.

In order to train the model in a specific way, the dataset that will be used to train the model needs to be analysed in order to inspect problematic specifications and find correlations between specific attributes and the passengers' chance of survival. The training dataset can then be transformed in ways to make the models prediction accuracy higher. The model is then trained, at which point the accuracy of the different feature engineering sets can be accessed before the selection of the top models is done. These top models are further tuned to enhance their accuracy before applying possible ensembling techniques. The final prediction can then be made with that final model against the testing data which results in an accuracy measure of the model that has been built.

Analysis of the accuracy and possible shortcomings or future improvements are then studied. My specific contributions within the team will be the focus throughout each section, while referencing sections my fellow team members completed.

2. DATA ANALYSIS

There are many factors that can be studied when analysing the specific dataset. This section was done in collaboration with Tristan Lilford and Tristan de Groot. My specific focus point was the shared basic analysis of the dataset before focusing on the numerical categories of the data.

2.1 Basic Analysis

As seen in *Figure 1* there are multiple columns that contain null values. These are Age, Cabin and Embarked. These entries will need to be altered in some way as models cannot be trained with null values. The test data set that will be used in the final prediction also contains data points. As seen in *Figure 2* the test set also contains null values for columns Cabin, Age and Fare. Survived is the column that needs to be determined for the test data set.

```
training.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 13 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    int64
 2   Pclass       891 non-null    int64
 3   Name         891 non-null    object
 4   Sex          891 non-null    object
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64
 7   Parch        891 non-null    int64
 8   Ticket       891 non-null    object
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object
 11  Embarked     889 non-null    object
```

Figure 1: The initial training dataset's different attributes

```
test.isnull().sum().sort_values(ascending = False)
```

```
Survived       418
Cabin          327
Age             86
Fare             1
PassengerId      0
Pclass           0
Name             0
Sex              0
SibSp            0
Parch            0
Ticket           0
Embarked         0
```

Figure 2: The initial test dataset's different attributes

This information must be noted during the feature engineering and data preprocessing section these needs to be attended to.

2.2 Numerical Categories Analysis

There are 12 different columns within the training set that will be analysed. As mentioned above, my focus was on analysing the numerical columns. The following columns that were analysed are Age, SibSp, Parch, and Fare.

	Parch	Survived
3	3	0.600000
1	1	0.550847
2	2	0.500000
0	0	0.343658
5	5	0.200000
4	4	0.000000
6	6	0.000000

	SibSp	Survived
1	1	0.535885
2	2	0.464286
0	0	0.345395
3	3	0.250000
4	4	0.166667
5	5	0.000000
6	8	0.000000

Figure 3: The SibSp and Parch relation with attribute and survival.

Figure 3 shows the relation of having a sibling or spouse on board and having a parent or children on board will effect the survivability. Both columns have a similar outlook where having none in each attribute lowers the chance of survival, having 1 or 2 of each attribute increases the chance of survival, and having too large of a family connection resulting in the lowest chance of survival. This analysis is important as it shows how these columns can have feature engineering applied to them to retrieve the vital aspects of the information.

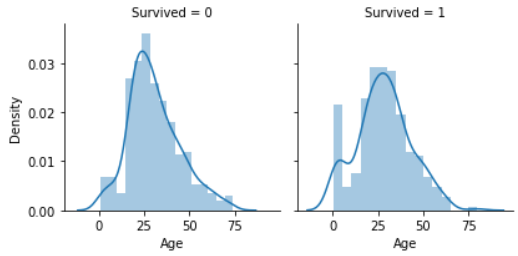

Figure 4: Age distribution in relation to survival

Figure 4 shows the relation of being of a certain age to the chance of survival. The notable differences between the 'did not survive' and the 'survived' distributions are that children were most likely to survive and young adults were least likely to survive.

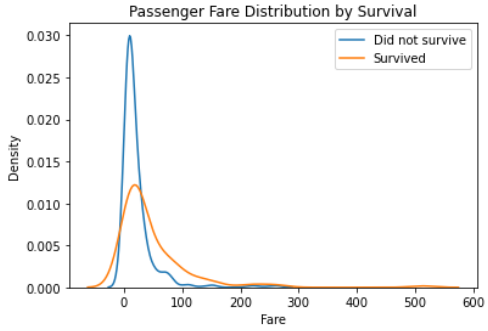

Figure 5: Fare distribution in relation to survival

Figure 5 shows the relation of how expensive the price of the Fare was to the chance of survival. The notable difference between the two graphs show that the density of passengers that survived paid a higher Fare amount compared to those that did not survive. This shows that the rich were prioritised and had a higher chance of survival.

```
df_num_1 = training[['Survived','Age','SibSp','Parch','Fare']]
print(df_num_1.corr())
sns.heatmap(df_num_1.corr())
```

```
            Survived       Age     SibSp     Parch      Fare
Survived    1.000000 -0.077221 -0.035322  0.081629  0.257307
Age        -0.077221  1.000000 -0.308247 -0.189119  0.096067
SibSp      -0.035322 -0.308247  1.000000  0.414838  0.159651
Parch       0.081629 -0.189119  0.414838  1.000000  0.216225
Fare        0.257307  0.096067  0.159651  0.216225  1.000000
```

```
<AxesSubplot:>
```

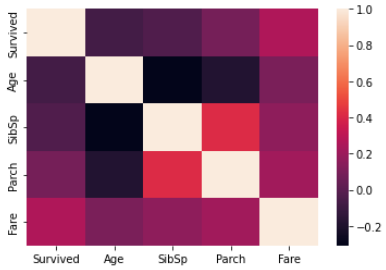

Figure 6: Correlation between the different numerical attributes and survival

Figure 6 shows a heatmap of the correlation between the different numerical attributes; Age, SibSp, Parch, and Fare, and Survived. This heatmap shows that SibSp and Parch has the strongest correlation between the attributes and that Fare has the biggest impact on survivability out of these attributes.

```
pd.pivot_table(training, index = 'Survived', values = ['Age','SibSp','Parch','Fare'])
```

Survived	Age	Fare	Parch	SibSp
0	30.626179	22.117887	0.329690	0.553734
1	28.343690	48.395408	0.464912	0.473684

Figure 7: The difference between survived and not survived for the numerical attributes

Figure 7 shows the respective mean values of the attributes of the passengers that survived versus those that did not survive for the numerical attributes. This reinforces the narrative that the rich passengers were prioritised and were most likely to survive.

The categorical columns were analysed by my fellow team members and this information was then used during the feature engineering and data preprocessing section.

3. MODEL BUILDING

3.1 Feature Engineering Data Preprocessing

The majority of this section was completed by my fellow team members, however, the initial setup and start of the section was my focus. The 2 entries of Embarked that had null values were replaced with the most common values of Embarked. Being less than 1% of the total entries, replacing it with the mode seemed the best fix and was not considered a major feature.

The next step was converting the categorical values of Sex and Embarked from strings to numerical values that is needed to train the model. Sex was converted from male and female to 0 and 1 while Embarked was converted from S, C and Q to 0, 1, and 2 to represent the different categories. The result of this is shown in *Figure 8*.

```
training['Sex'].value_counts(dropna = False)
```

```
0    577
1    314
Name: Sex, dtype: int64
```

```
training['Embarked'].value_counts(dropna = False)
```

```
0    646
1    168
2     77
Name: Embarked, dtype: int64
```

Figure 8: Value counts of the contents of the Sex and Embarked columns

My next focus was creating 2 different training sets with basic feature engineering applied to them which will both be run to train the models differently. These were not meant to be accurate models but rather as a starting point before more advanced feature engineering is applied to our training sets. The first set was simply all the relevant columns of the training set but with the columns dropped that contain any NaN values. 'PassengerId', 'Name', and 'Ticket' was dropped due to being largely irrelevant while after dropping NaN's Age and Cabin were dropped. *Figure 8* shows the first training set. Secondly, I created another training set that was similar to the first set except for all outliers within the data set was dropped before applying the same dropping of NaN columns.

	SibSp	Parch	Fare	Survived	Pclass	Sex	Embarked
0	1	0	7.2500	0	3	0	0
1	1	0	71.2833	1	1	1	1
2	0	0	7.9250	1	3	1	0
3	1	0	53.1000	1	1	1	0
4	0	0	8.0500	0	3	0	0
...	...	...	...	...	...	...	...
886	0	0	13.0000	0	2	0	0
887	0	0	30.0000	1	1	1	0
888	1	2	23.4500	0	3	1	0
889	0	0	30.0000	1	1	0	1
890	0	0	7.7500	0	3	0	2

891 rows × 7 columns

Figure 9: Dropping NaN columns Feature Set

```
We will drop these 10 indices:  [27, 88, 159, 180, 201, 324, 341, 792, 846, 863]
Before: 891 rows
After: 881 rows
```

	SibSp	Parch	Fare	Survived	Pclass	Sex	Embarked
0	1	0	7.2500	0	3	0	0
1	1	0	71.2833	1	1	1	1
2	0	0	7.9250	1	3	1	0
3	1	0	53.1000	1	1	1	0
4	0	0	8.0500	0	3	0	0
...	...	...	...	...	...	...	...
876	0	0	13.0000	0	2	0	0
877	0	0	30.0000	1	1	1	0
878	1	2	23.4500	0	3	1	0
879	0	0	30.0000	1	1	0	1
880	0	0	7.7500	0	3	0	2

881 rows × 7 columns

Figure 10: Dropping Outliers of Basic Set and then dropping NaN values

Multiple features were then created based on the analysis that was done. For example a Size of Family and Alone column was created by Van Niekerk Ferreira which took the analysis of SibSp and Parch and combined them into two columns, one refereeing to if the passengers were Alone on board and another computing each passengers Size of Family. The resultant feature matches what the analysis said that passengers that travelled alone were less likely to survive. This is shown in *Figure 11*.

	Alone	Survived
0	0	0.505650
1	1	0.303538

Note: In the table above '1' refers to a passenger that travelled alone.

Figure 11: New Feature column Alone based on SibSp and Parch analysis

I provided insight into the remaining features however each feature was created and analysed by my fellow team members. The features regarding replacing the NaN values from the age column in different ways namely; mean, median and using age groups, is an important feature added. The analysis of these features matched my analysis of age within the Data Analysis section.

After all the respective features were created. I created a system where my team members can create subsets of the training set with a focus on the set that they believe will be the most accurate using the newly created features. These different dataframe sets were inserted into a list that will be used in the model training section below. *Figure 12* shows an example of creating the new training set and *Figure 13* shows how these are inserted into a list.

```
fe14 = training[['Survived','Pclass','Sex','Age_mean','Alone','Fare','Embarked','name_title']]
fe14_test = test[['Pclass','Sex','Age_mean','Alone','Fare','Embarked','name_title']]
fe14
```

Figure 12: Example of the new training set with respective new features

```
featureEngineering = [fe0, fe1, fe2, fe3, fe4, fe5, fe6, fe7, fe8, fe9, fe10, fe11, fe12, fe13, fe14, fe15]
testFeatureEngineering = [fe0_test, fe1_test, fe2_test, fe3_test, fe4_test, fe5_test, fe6_test, fe7_test, fe8_test, fe
```

Figure 13: Showcasing how the different feature sets were stored in a list

3.2 Model Training

Within Model Building, I created a method that iterates through all the different feature sets for each model. A total of 16 different feature training sets were created that will each train the models. The feature training set was divided into a train and validation set. I measured the accuracy with which the model predicts the correct output by comparing the predicted results from the validation set data with the actual values of the validation set results. This is done for each feature engineering set. This is shown in *Figure 14*. The accuracy is then found by getting the mean of the different accuracy measurement techniques. *Figure 15* shows an example of the output for the Logistic Regression model. This is then used to establish which feature engineering set results in the most accurate model.

```
for i in range(len(featureEngineering)):
    if type(featureEngineering[i]) != np.ndarray:
        feature = featureEngineering[i].copy(deep=True)
        X = feature.drop(columns= ['Survived'])
        y = feature['Survived']
    test_data_log = testFeatureEngineering[i].copy(deep=True)

    # validation
    X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=1)
    logreg = LogisticRegression()
    logreg.fit(X_train, y_train)
    Y_pred = logreg.predict(X_test)

    # accuracy
    print('Accuracy for fe' + str(i))
    a1 = metrics.accuracy_score(y_train, logreg.predict(X_train))
    print("Train set Accuracy: ", a1)
    a2 = metrics.accuracy_score(y_test, Y_pred)
    print("Test set Accuracy: ", a2)
    a3 = f1_score(y_test, Y_pred, average='weighted')
    print('F1 Score:', a3)
    a4 = cross_val_score(logreg,X_train,y_train,cv=5)
    a4 = a4.mean()
    print('Cross-val mean: ', a4)
    mean_acc = round(((a1 + a2 + a3 + a4)/4) * 100, 2)
    logreg_acc.append(mean_acc)
    print('logreg has a mean accuracy score of: ' + str(mean_acc) + '%\n')
    validationAccuracy.append(mean_acc)
```

Figure 14: Showcasing how the method iterates through the feature engineering sets, the validation set and its accuracy measurements.

```
Accuracy for fe0
Train set Accuracy:  0.8047752808988764
Test set Accuracy:  0.7821229050279329
F1 Score: 0.7786200811127533
Cross-val mean:  0.8006599034768047
logreg has a mean accuracy score of: 79.15%

Accuracy for fe1
Train set Accuracy:  0.8039772727272727
Test set Accuracy:  0.7796610169491526
F1 Score: 0.777139486676327
Cross-val mean:  0.7969098277608916
logreg has a mean accuracy score of: 78.94%

Accuracy for fe2
Train set Accuracy:  0.8089887640449438
Test set Accuracy:  0.7988826815642458
F1 Score: 0.7952644995294874
Cross-val mean:  0.8006500541711811
```

Figure 15: An example of the output of the method shown in *Figure 14*

At the end of each model, the training error graph is shown. In the end after all the models, a table was created showcasing the accuracies of each model for each feature engineering set that is retrieved from the outputs of each, i.e. *Figure 15*. These were done by Tristan Basel and Van Niekerk Ferreira. An example of the table is shown in *Figure 16*. Using inspection, from this table, our group decided that features 6, 12 and 14 are most likely to be the best. These are then used with the model tuning section.

	Model	Best_Score	Best_Score_Index	FE0_Score	FE1_Score	FE2_Score	FE3_Score	FE4_Score	FE5_Score	FE6_Score	FE7_Score	FE8_Score	FE9_
0	Random Forest	85.05	13	81.17	83.01	82.42	77.62	82.63	85.02	84.06	82.17	81.65	
1	Decision Tree	83.27	14	80.80	82.48	82.51	79.27	80.74	82.64	82.35	82.03	81.84	
2	Support Vector Machines	80.77	11	64.15	66.34	64.50	79.65	64.50	65.81	65.81	65.91	65.51	
3	Logistic Regression	80.27	6	79.15	78.94	80.09	79.12	78.93	79.93	80.27	79.85	80.10	
4	Linear SVC	80.00	11	75.40	78.60	75.30	79.13	78.27	77.24	72.25	77.15	77.77	
5	Naive Bayes	79.39	2	77.99	77.58	79.39	76.47	73.72	78.25	78.25	74.15	74.22	
6	KNN	79.35	11	77.19	77.53	77.50	78.07	75.47	71.71	72.82	76.98	77.83	
7	Stochastic Gradient Decent	79.30	11	73.81	68.31	45.87	76.28	73.82	60.01	71.14	68.37	76.73	
8	Perceptron	74.62	12	72.98	73.05	58.64	67.37	70.55	64.89	65.99	74.12	73.17	

Figure 16: Best scores for different features for each model

4. MODEL TUNING/ENSEMBLING

The feature set that is to be used throughout the model tuning section was done similarly to the model building section, shown in *Figure 17*.

```
sets = [fe6, fe12, fe14]
sets_test = [fe6_test, fe12_test, fe14_test]
```

Figure 17: Feature engineering sets to be used within model tuning

4.1 Cross Validation

The Cross Validation section was largely done by Jesse Iyuke. Cross validation was also used to measure each model for its accuracy which was shown in *Figure 14*. This section is done is order to find a more accurate result for the accuracy score received at each section. The mean and standard deviation of the cross validation is shown and it is found that Random Forest, Decision Tree and Logistic Regression were the most accurate models as shown in *Figure 19*. Random Forest is a classification method that works by constructing multiple Decision Trees when the model is trained. *Figure 18* shows an example of how the Random Forest algorithm works. Due to Random Forest being a combination of Decision Trees, and Random Forest performing better than Decision Tree, Decision Tree was replaced by a lesser accurate model, KNN, for testing purposes, to see if the accuracy improves drastically when it is tuned.

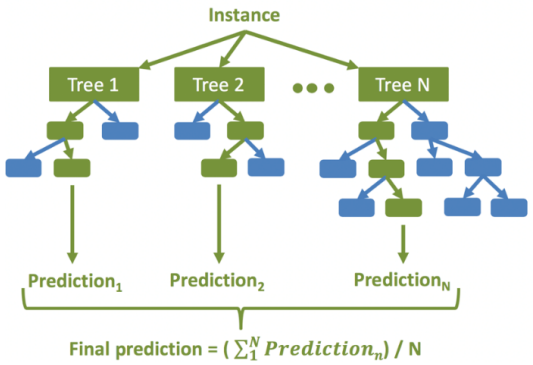

Figure 18: Diagram of how the Random Forest ML model works, retrieved from [5]

	Cross Validation Mean	Cross Validation Std	Algorithm
0	0.799201	0.047973	Random Forest
1	0.795743	0.039454	Decision Tree
2	0.777803	0.035278	Logistic Regression

Figure 19: Accuracy of models after cross validation

4.2 Model Tuning

The model tuning section consisted of a similar principle as that used within the model building section that I did except instead of using default parameters for each model, these parameters are found using Grid Search to find the best parameters for each model. This section, however, did not prove to be successful. *Figure 20* shows how the models accuracies changed after tuning for the 3 different feature engineering sets. This shows that none of the models benefits from the tuning and their accuracies decreased after tuning.

	Model	Score	Tuned Score for fe6	Tuned Score for fe12	Tuned Score for fe14
0	Random Forest	85.05	83.39	79.13	83.61
1	Logistic Regression	80.27	79.91	79.57	79.46
2	KNN	79.35	73.97	79.24	76.88

Figure 20: Accuracy of models after tuning for 3 sets

4.3 Ensembling Methods

This section was done by Tristan de Groot and Jesse Iyuke. Ensembling is the process where the combination of more than one model is used to predict a result. They used the Voting Classifier and ensembled the Random Forest and Logistic Regression models for feature set 12. This was found to be the most accurate predictor through inspection and trial and error. The bagging and boosting methods were also used however it was found the Voting Classifier was more accurate.

5. CONCLUSION

The final accuracy score that our group received was 0.775. There are multiple different approaches that can improve the score. The attempt of analysing 16 different feature engineering sets work great in concept, however the combination of the new features, the way those features were created and the selection of the training set attributes could have been done differently. A more concise approach with a stronger reference to the analysed data will have created features that improve the accuracy of the training model. The model building section was adequate however the fact that the tuned models performed worse than the default models is in theory not correct. The ensembling works well but is a section that relies on the models, and will have been better if the models individually performed at the correct standard. The selection of the best feature training set to use and selecting the best performing model is all correct, however, the actual feature set and the transformation of it from the full training set is likely where the problem lies. In conclusion, working as a team on this project has given us all a good foundation in Machine Learning and the different aspects of teamwork.

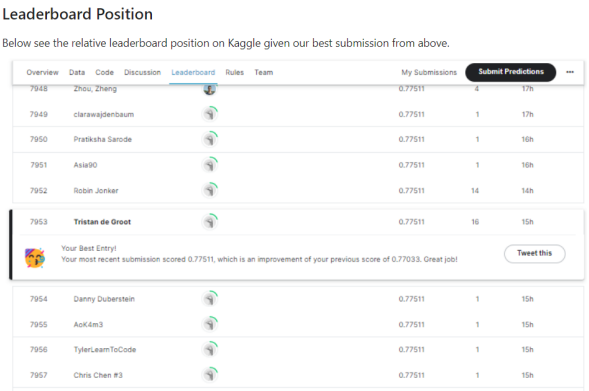

Figure 21: Our group position, as submitted by Tristan de Groot, on Kaggle

6. References

[1] Ken Jee, 2021, "Titanic Project Example." Available at https://www.kaggle.com/ kenjee/titanic-project-example

[2] Jason Chong, 2021, "Titanic: Machine Learning from Disaster." Available at https:// github.com/ chongjason914/ kaggle-titanic

[3] Nick Zhang, 2021, "Ensemble for Titanic." Available at https:// www.kaggle.com/code/ nickzylove/ ensemble-for-titanic/notebook.

[4] Minsuk-heo, 2017, "Titanic: Machine Learning from Disaster." Available at https:// github.com/ minsuk-heo /kaggle-titanic /blob/master/titanic-solution.ipynb.

[5] Tyler McCandless, 2019, "Figure 8. Diagram of the random forest machine learning method, which...," ResearchGate, Available at https://www.researchgate.net/ figure/Diagram-of-the- random-forest-machine-learning-method-which-is-an-ensemble- of-regression_fig7_ 333612054