CECS1020:

Final Report - Group Project

Group 6 - Titanic Survival Prediction

College of Engineering and Computer Science

Members: Nguyen Tiet Nguyen Khoi, Nguyen Duong Tung, Nguyen Hoang Trung Dung
Keywords: Titanic, Kaggle, Classification, Logistic Regression, Feature Engineering

1549 Words 18th June, 2021

Outline

1 Introduction

About the Titanic Challenge

Titanic survival prediction is a fundamental problems for ML newbies to get their hands dirty with one of the most infamous accident in the history - the sinking of the Titanic.

On April 15, 1912, during her maiden voyage, the widely considered "unsinkable" RMS Titanic sank after colliding with an iceberg. Unfortunately, there were not enough lifeboats for everyone onboard, resulting in the death of 1502 out of 2224 passengers and crew.

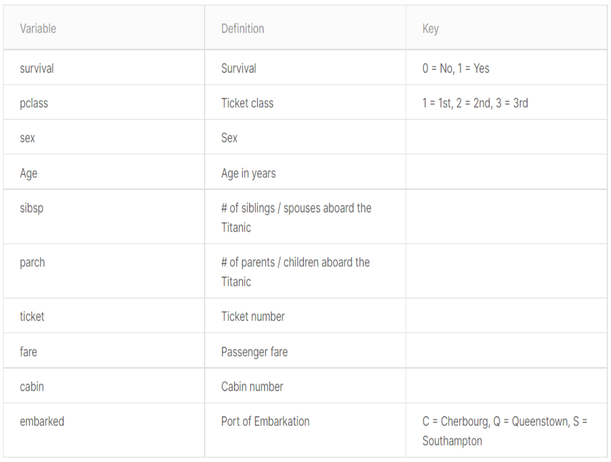
While there was some element of luck involved in surviving, it seems some groups of people were more likely to survive than others. This problem focus on building a classification model to selects important features that affected the survivability of the Titanic's passengers, and to predict weather a passenger with such features can be survived.

This problem has been published on Kaggle and attracts more than 45,000 teams to submit their solutions.

About the Dataset

The dataset used for problem solving in this report is given by Kaggle. The Kaggle's Titanic dataset is a sub-dataset which has been removed some features and reports that appears in the full report about this shipwreck.

The Kaggle's dataset is split into 2 sets, train and test sets. The train set is used for building models, and the test set is for evaluation purpose. The train set contains 891 datapoints and the test set contains 418 datapoints.



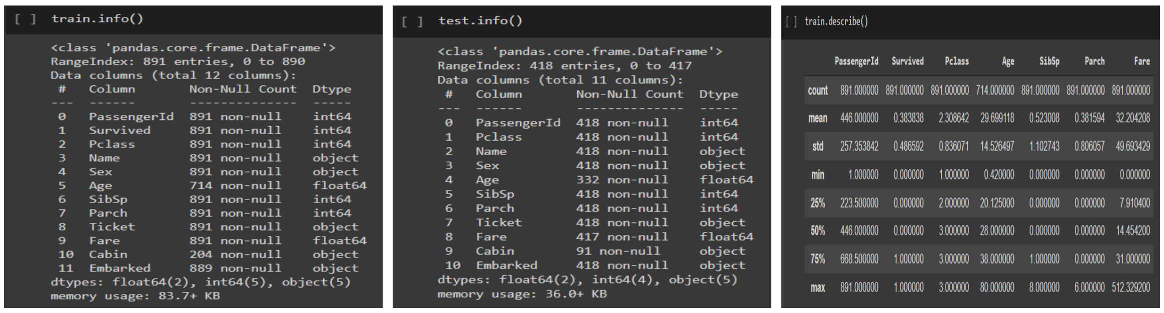
Variable	Definition	Key
survival	Survival	0 = No, 1 = Yes
pclass	Ticket class	1 = 1st, 2 = 2nd, 3 = 3rd
sex	Sex	
Age	Age in years	
sibsp	# of siblings / spouses aboard the Titanic	
parch	# of parents / children aboard the Titanic	
ticket	Ticket number	
fare	Passenger fare	
cabin	Cabin number	
embarked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

Fig. 1. Dataset description (Given by Kaggle).

2 Visualization & Pre-Processing

2.1 Looking at the data and visualizing simple graphs

After uploading the dataset onto our Google Colab, we took a general look at the data. We checked the datatype and the number of null using info() function, and went through some analysis for numerical features (mean, std, min-max values) using describe function().



```
[ ]  train.info()

     <class 'pandas.core.frame.DataFrame'>
     RangeIndex: 891 entries, 0 to 890
     Data columns (total 12 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
     dtypes: float64(2), int64(5), object(5)
     memory usage: 83.7+ KB
```

```
[ ]  test.info()

     <class 'pandas.core.frame.DataFrame'>
     RangeIndex: 418 entries, 0 to 417
     Data columns (total 11 columns):
      #   Column       Non-Null Count  Dtype
     ---  ------       --------------  -----
      0   PassengerId  418 non-null    int64
      1   Pclass       418 non-null    int64
      2   Name         418 non-null    object
      3   Sex          418 non-null    object
      4   Age          332 non-null    float64
      5   SibSp        418 non-null    int64
      6   Parch        418 non-null    int64
      7   Ticket       418 non-null    object
      8   Fare         417 non-null    float64
      9   Cabin        91 non-null     object
      10  Embarked     418 non-null    object
     dtypes: float64(2), int64(4), object(5)
     memory usage: 36.0+ KB
```

```
[ ]  train.describe()
```

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

Fig. 2. There were some null data that needed to be handled, especially the 'Cabin'.

We then plotted some graph, however, due to the report's length requirement, we do not show all of the graphs here. Below is a simple correlation graph to look for the correlation between some important (numerical) features and the survivability. We also changed Embarked and Sex from categorical into numerical features to check for the correlation by mapping 'S','Q','C' from the original into 0,1,2, and 'male' and 'female' into 0 and 1 (however, we still treated these features as categorical later). We witnessed that the correlations between Survival and Fare, Sex, and PClass, were strong.

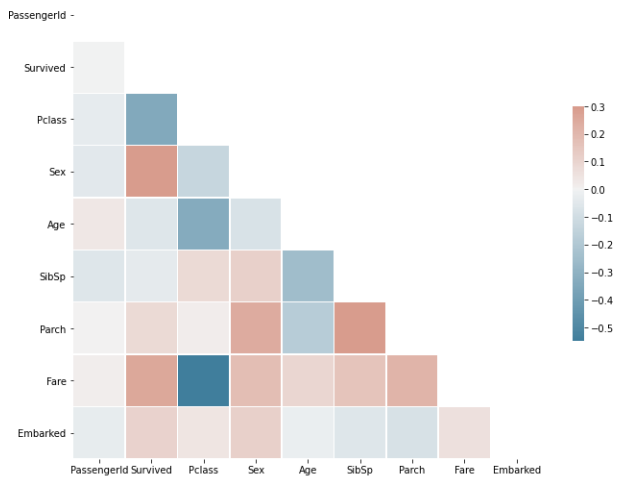


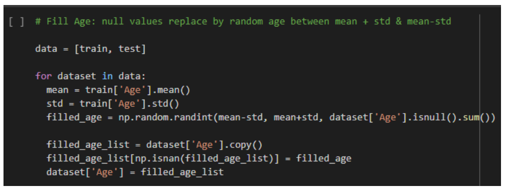
Fig. 3. Correlation plot using Seaborns library.

For convenience, we created a list 'dataset' which contained the train set and the test set. Therefore, we would pre-process both sets next by next by running a for-loop through the list.

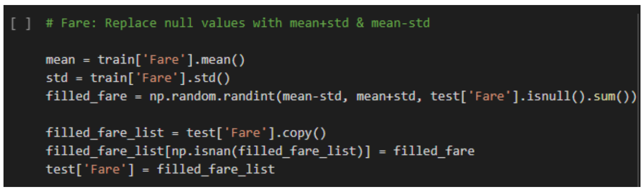
2.2 Filling null values & dropping unnecessary features

In order to preserve data for the training process, we avoided dropping row from the dataset and tried to fill null values by multiple ways. Also, at first we tried do drop some unrelated features such as Name, Ticket, and Passenger ID. However, the performance of the model decreased. After several updates, we decided not to drop any feature.

For Age and Fare numerical features, we generated random numbers in range $mean - std$ to $mean + std$ and replaced the null values with new generated data.



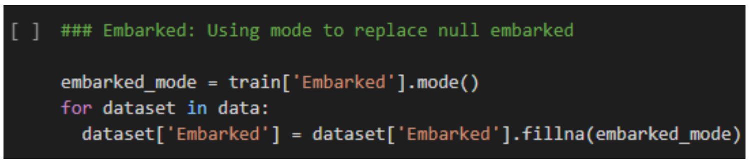
```
[ ]  # Fill Age: null values replace by random age between mean + std & mean-std

     data = [train, test]

     for dataset in data:
       mean = train['Age'].mean()
       std = train['Age'].std()
       filled_age = np.random.randint(mean-std, mean+std, dataset['Age'].isnull().sum())

       filled_age_list = dataset['Age'].copy()
       filled_age_list[np.isnan(filled_age_list)] = filled_age
       dataset['Age'] = filled_age_list
```



```
[ ]  # Fare: Replace null values with mean+std & mean-std

     mean = train['Fare'].mean()
     std = train['Fare'].std()
     filled_fare = np.random.randint(mean-std, mean+std, test['Fare'].isnull().sum())

     filled_fare_list = test['Fare'].copy()
     filled_fare_list[np.isnan(filled_fare_list)] = filled_fare
     test['Fare'] = filled_fare_list
```

Fig. 4. Filled Age & Fare null values with mean & std.

We used mode for filling null values in Embarked feature. Mode is the best measure of central tendency for nominal data.



```
[ ]  ### Embarked: Using mode to replace null embarked

     embarked_mode = train['Embarked'].mode()
     for dataset in data:
       dataset['Embarked'] = dataset['Embarked'].fillna(embarked_mode)
```

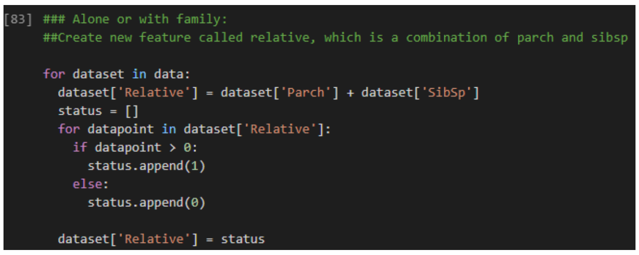
Fig. 5. Filled Embarked null value using mode.

Because of the serious lack of data in the Cabin feature, dealing with it using mean, mode, or median would not be an appropriate choice. Therefore, we handled it using one-hot encoding. This would be mentioned below. We have also re-plotted the correlation graph to see if it had any change, however, it remained the same.

2.3 Creating new features

After handling null values and looking at the graphs, we had some hypothesis and found that some original features could be processed to generate new features that could more strongly correlated to the survivability. We have created 3 new features, which are Relative, Age Range, and Name Title (we also tried to create some others, but it did not work really well).

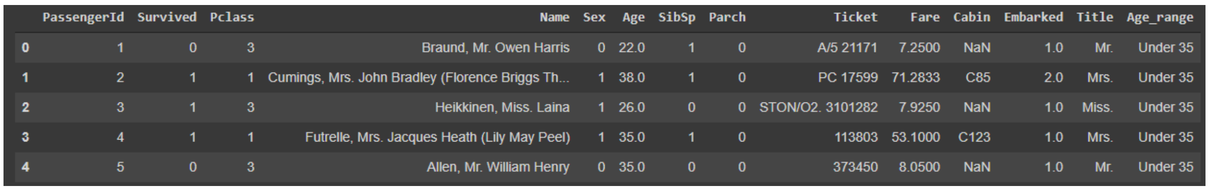
The Relative feature is a combination of Sibsp (siblings) and Parch (parents) features. It reflects whether a passenger went alone or with their relatives (parents or siblings). Another feature that we created was the Age Range, which divided the passenger into several group of ages. However, after several updates, we decided to drop the Age Range since we tried to treat Age and Fare as categorical features, which has boosted our model's performance (mentioned later).



```
[83] ### Alone or with family:
     ##Create new feature called relative, which is a combination of parch and sibsp

     for dataset in data:
       dataset['Relative'] = dataset['Parch'] + dataset['SibSp']
       status = []
       for datapoint in dataset['Relative']:
         if datapoint > 0:
           status.append(1)
         else:
           status.append(0)

       dataset['Relative'] = status
```

Fig. 6. Relative - went alone or with family

Lately, after several updates and trials, we realized that the name titles is very important, since they reflect the ages, social positions, and genders of passengers. Therefore, we created new features Name Title, which derived from the title in the Name feature. This new feature increased the model performance as well. There were about 11 name titles that appeared occasionally, and less-appeared titles were grouped as Others.



	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	Title	Age_range
0	1	0	3	Braund, Mr. Owen Harris	0	22.0	1	0	A/5 21171	7.2500	NaN	1.0	Mr.	Under 35
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	1	38.0	1	0	PC 17599	71.2833	C85	2.0	Mrs.	Under 35
2	3	1	3	Heikkinen, Miss. Laina	1	26.0	0	0	STON/O2. 3101282	7.9250	NaN	1.0	Miss.	Under 35
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	1	35.0	1	0	113803	53.1000	C123	1.0	Mrs.	Under 35
4	5	0	3	Allen, Mr. William Henry	0	35.0	0	0	373450	8.0500	NaN	1.0	Mr.	Under 35

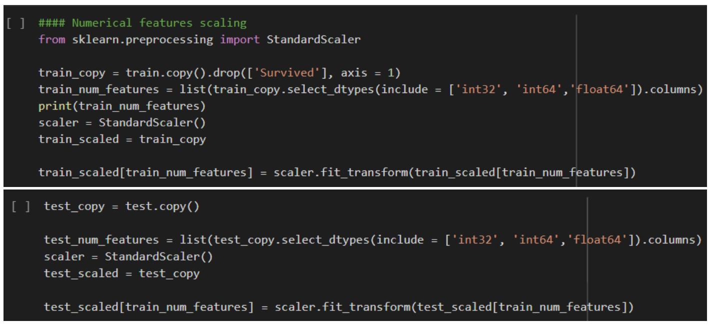
Fig. 7. Data after adding new features

2.4 Standard scaling for numerical features

The formula for standardization is defined by:

$$\boxed{X_{standardized} = \frac{X_i - X_{mean}}{Std}}$$

We implemented standardization using StandardScaler() in Sklearn library. We created a copy of our dataset for standardization. Then, we filled out numerical feature by select features with data type integer and float. In the train set, since we did not want to transform the Survival feature, which was our classes for prediction, we dropped the Survival feature out of the copy.



```
[ ]  #### Numerical features scaling
     from sklearn.preprocessing import StandardScaler

     train_copy = train.copy().drop(['Survived'], axis = 1)
     train_num_features = list(train_copy.select_dtypes(include = ['int32', 'int64','float64']).columns)
     print(train_num_features)
     scaler = StandardScaler()
     train_scaled = train_copy

     train_scaled[train_num_features] = scaler.fit_transform(train_scaled[train_num_features])
```

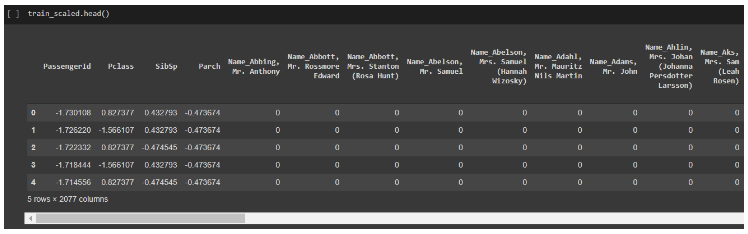
```
[ ]  test_copy = test.copy()

     test_num_features = list(test_copy.select_dtypes(include = ['int32', 'int64','float64']).columns)
     scaler = StandardScaler()
     test_scaled = test_copy

     test_scaled[train_num_features] = scaler.fit_transform(test_scaled[train_num_features])
```

Fig. 8. Imported the module and fit the dataset into StandardScaler().

2.5 One-hot encoding for categorical features

We encoded all categorical features using Pandas dummies. After encoding, we had more than 1700 features. There were some categories of features that just appeared in train set or test set only, for example, some Cabin categories appeared only in the test set and not the train set. Therefore, we also included these missing categories while encoding both the train and the test set. In general, the train set and the test set would have same dimension.

Age and Fare as categorical features: When we created new features, such as the Age Range (and also Fare Range), we tried to apply different divisions and intervals to look for the most suitable ones. However, we found that there were hardly any division that could strongly improve our model performance, and the accuracy of our model remained the same. So we discarded the Age Range and the Fare Range, and tried to treat Age and Fare as categorical features and applied one-hot encoding for these features. This resulted in more than 2000 features after encoding.



```
[ ] train_scaled.head()
```

	PassengerId	Pclass	SibSp	Parch	Name_Abbing, Mr. Anthony	Name_Abbott, Mr. Rossmore Edward	Name_Abbott, Mrs. Stanton (Rosa Hunt)	Name_Abelson, Mr. Samuel	Name_Abelson, Mrs. Samuel (Hannah Wizosky)	Name_Adahl, Mr. Mauritz Nils Martin	Name_Adams, Mr. John	Name_Ahlin, Mrs. Johan (Johanna Persdotter Larsson)	Name_Aks, Mrs. Sam (Leah Rosen)
0	-1.730108	0.827377	0.432793	-0.473674	0	0	0	0	0	0	0	0	0
1	-1.726220	-1.566107	0.432793	-0.473674	0	0	0	0	0	0	0	0	0
2	-1.722332	0.827377	-0.474545	-0.473674	0	0	0	0	0	0	0	0	0
3	-1.718444	-1.566107	0.432793	-0.473674	0	0	0	0	0	0	0	0	0
4	-1.714556	0.827377	-0.474545	-0.473674	0	0	0	0	0	0	0	0	0

5 rows × 2077 columns

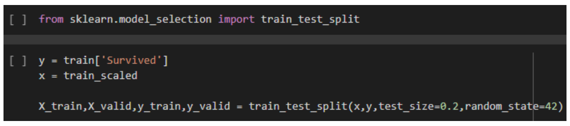
Fig. 9. Data after one-hot encoding

Accuracy improvement & Overfitting problem: Treating Age and Fare as categorical features was the most important progress during our project, since it increased the accuracy from 0.78947 to 0.80143, which is our final performance.

However, we do believe that the enormous number of features we created caused overfitting, since the model performed about 0.94 accuracy on the sub-train set, 0.8401 on the validation set (divided train set into sub-train used for training and validation sets, sub-train : validation = 8 : 2) but only 0.80143 on the test set.

3 Model selection and training

We did several research about suitable models for the Titanic problem and according to many Kagglers and papers, the Logistic Regression is still the best model for this challenge. Therefore, in the beginning, we focused on implementing Logistic Regression. We did try alternative solutions, such as Decision Tree and Random Forest, which would also be introduced below. First, we splitted the train set into sub-train and validation sets. The sub-train would be used for training.



```
[ ] from sklearn.model_selection import train_test_split

[ ] y = train['Survived']
    x = train_scaled

    X_train,X_valid,y_train,y_valid = train_test_split(x,y,test_size=0.2,random_state=42)
```

Fig. 10. Splitting the train set

3.1 Logistic Regression

About Logistic Regression: Introduction from machinelearningcoban.com [link].

We imported Sklearn Logistic Regression and fit our data into it. The trained model gave 0.94 accuracy on the sub-train set, 0.84 on the validation set, and 0.80143 on the test set.

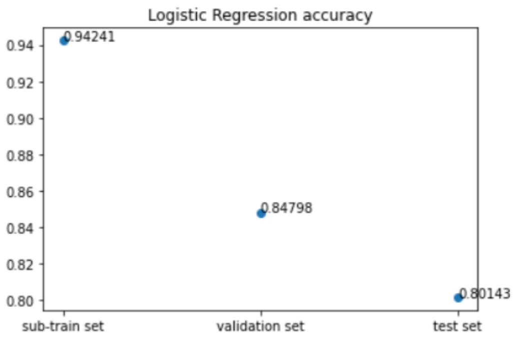


Fig. 11. Accuracy on 3 sets

In our previous progress, where we did not treat Age and Fare as categorical features, the trained model gave 0.87 on the sub-train set, 0.82 on the validation set, and 0.78947 on the test set, which we found had lower variance.

3.2 Alternative solutions: Decision Tree and Random Forest

About Decision Tree and Random Forest: Introduction [Decision Tree][Random Forest].

The accuracy of 2 models as given below. We witnessed that the accuracy of Decision Tree and Random Forest models that we trained for this problem were really similar to each other, which ended up at about 0.77 accuracy on test set. The Random Forest model performed a little bit better.

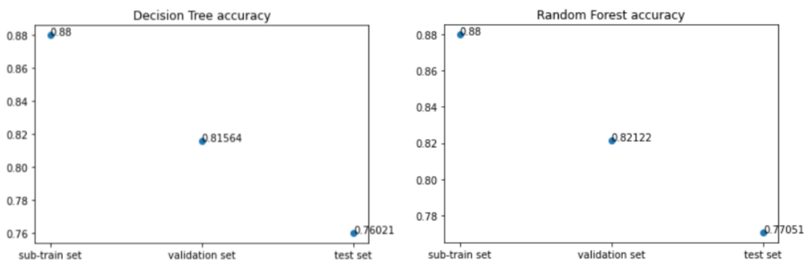


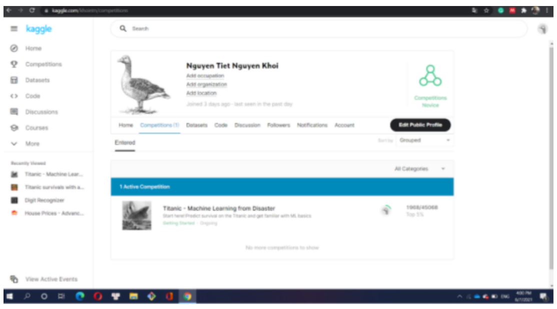
Fig. 12. Accuracy of Decision Tree (left) and Random Forest (right) models

4 Conclusion

4.1 Our team achievement

In general, we were happy to land in the top 5% on Kaggle with 0.80143 accuracy score, rank 1989 over 45.461 teams. Here is the link to our Kaggle profile and Google Colab.

- Kaggle profile
- Google Colab



Nguyen Tiet Nguyen Khoi
Add occupation
Add organization
Add location
Joined 3 days ago · last seen in the past day

1 Active Competition

Titanic - Machine Learning from Disaster
Start here! Predict survival on the Titanic and get familiar with ML basics
Getting Started · Ongoing
1968/45068
Top 5%

No more competitions to show

Fig. 13. Kaggle profile, screenshot on June 7th

4.2 Some important updates overtime

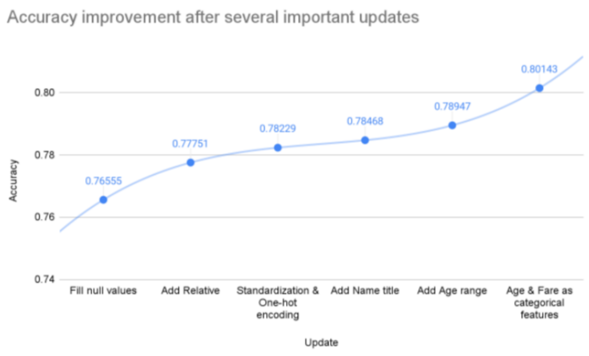


Fig. 14. Accuracy improvement for some critical progresses

4.3 Final conclusions

To sum up our report, we would like to give some final conclusions based on our experience solving this challenge.

- Logistic Regression is one of the most suitable model for this problems. However, some alternative methods may work and we should try as many models as we can to find the best one for ourselves.

- Visualization is very important to look into insights and set up our hypothesis for any given problems.
- Pre-processing is one of the most important part. If being handled correctly, the performance would be highly increased if we use standardization and feature engineering. Without using such techniques, we just ended up with only about 0.75 accuracy score.
- Create new features, in some specific cases, would be beneficial and make the trained model performs better. However, it would also cause counter-effects if we create many new unnecessary features.

Last words, we have learned so much throughout this project, and have got our hands dirty for the first time, as we were beginner to Machine Learning. There would be many things we should improve in the future. We would like to thank professor Minh Do, lecturers Hieu Pham and Hieu Hoang, and the teaching assistants for this course. Thank you.

5 References

[1] Chris Deotte, “Titanic using Name only [0.81818]. (English),” *Kaggle*, Oct. 2018.

[2] R. Ng, “Titanic Survival Data Exploration. (English),” *Ritchieng.Github.Io.*, May 2020.

[3] P. Schmidt, “Titanic Data Set – How I increased my score from 79% to 82%. (English),” *That Data Tho.*, Jan. 2021.

[4] Sedrak, “Titanic survivals with age prediction. (English),” *Kaggle*, 2018.

[5] J. Roberts, “Titanic Using Ticket Groupings. (English),” *Kaggle*, 2018.

[6] Ananyd36, “7 Feature Engineering Techniques in Machine Learning You Should Know. (English),” *Analytics Vidhya*, 2020.

[7] L. Fink, “Titanic Dive Through: Feature scaling and outliers. (English),” *Kaggle*, 2019.