Assignment 2: Titanic: Machine Learning from Disaster

MSc Data Science & Business Analytics – ESSEC Business School | CentraleSupélec

Team Four Sigma

Rafaëlle Aygalenq B00724587 Sarah Lina Hammoutene B00712035 Dóra Linda Kocsis B00714326 Noémie Quéré B00719656

1 SECTION 1: FEATURE ENGINEERING

1.1 FEATURE CREATION

1.1.1 Creating the Feature: NameLength

By using the Name attribute, we created a new feature that contains the length of the name of each passenger. At that time, longer name referred to nobility titles. The longer the name was, the most it indicates that the person came from a long hierarchy of noblemen, and thus possibly will be more likely to be saved first.

1.1.2 Missing Value Analysis

The datasets at hand were initially split into two: a test and a training set with 10 and 11 variables respectively. During the data cleaning and feature engineering processes and for the sake of simplicity, we have decided to combine them into one. Missing values were imputed for the following variables:

- **Age:** Given, the variable has 263 missing values, taking the mean or median of the combined dataset, would skew the distribution of the variable. We decided to impute age by applying an `ExtraTreesRegressor`[1] on the variable. The tree allowed us to predict the missing values by using the following attributes: `Fare`, `Parch`, `Pclass`, `SibSp`, `TitleCat`, `CabinCat`, `Sex`, `EmbarkedCat`, `FamilySize`, `NameLength`. (In this case the datasets were considered separated as in the initial scenario to avoid data leakage from the test set.)
- **Fare:** the one missing value was replaced by taking the variable's mean across the combined dataset.
- **Cabin:** We decided to set NAs to 0 as unknown. Missing values might be attributed to the crew of Titanic or low-class passengers, both of whom might have had lower chances of survival, whence it can be relevant to record this information. The attribute was encoded as scalar in this case.
- **Embarked**: missing embarkment locations were imputed by the most common value

1.1.3 Creating the Feature: Social_titles

We observed in the Name column, that some kind of social title is recorded, such as Mr., Mrs., Countess or Captain. First, we split the Name column into 3 parts, which allowed us to examine the unique values of social titles. We decided to create 5 groups and encode the variable as a scalar: Officer (1), Royalty (2), Mrs. (3), Miss (4), Mr. (5), Master (6). We intended to better understand how social title affects the ability to survive. We intended to capture, whether an Officer is more to choose helping the other passengers, whence lowering his chances. Or would a Royalty be amongst the first to leave the sinking ship given their title?

1.1.4 Creating the Feature: Married

Again, by looking at the Name attribute, we could recover whether a passenger was married and traveled with his/her spouse. Here, we expected a higher survival rate for couples as they are more likely to stick together. From the Name column we retrieved the passenger's' family names and titles. The latter was then converted into Mr., Mrs. or Miss. Taking the female passengers with a Mrs. title and a matching family name with a male passenger of above the age of 18 indicates a marital status (1) in our binary feature. Initially, we considered the Ticket number and matching the equivalents, as we assumed spouses bought their tickets together and travel together. However, after further analysis, we found one extension from this hypothesis and did not follow through with the approach.

1.1.5 Creating the Feature: Family_Size

Given we have information whether a passenger traveled with their parent / children (`Parch`) or Siblings / Spouse (`SibSp`), it is imminent we can calculate the size of their families on the Titanic. By taking one plus the sum of the `Parch` and `SibSp` variables we obtained the variable. Given, our preliminary analysis of the dataset (see notebook provided), the biggest family was at the count of 11. Our intuition essentially suggests two scenarios:

- larger families were harder to rescue as family members wanted to stick together whence lowering their chances of survival,
- or on the contrary they were more likely to be placed faster in the lifeboats than single individuals.

[1]http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.ExtraTreesRegressor.html

1.1.6 Categorization according to Age and Sex

We also created the following features based on the sex and age attributes: `child`, `female_adult`, `male_adult`. This categorization helped us to create the features described in the next section. Furthermore, the extra features allow us to more easily predict survival as women and children were a priority at the time of boarding passengers to lifeboats.

1.1.7 Creating the Feature: Perishing Mother / Wife and Surviving Father / Husband

Based on our combined set, the `Name`, `Parch`, `SibSp` attributes, we could create new variables for the mothers and wives who passed away, and the fathers and husband who survived. The prior helps us in the prediction of children and husbands who did not survive, as we assumed families stayed together. The latter, gives an approximation to the married women and children who survived the disaster given the father or husband was observable in the training set.

1.1.8 Transformation of Cabin Numbers

Titanic's decks were numbered A to G from the top to its bottom. From research we know, that the lower decks were divided and separated with watertight, remotely controlled doors. The flooding of any more than 2 of these compartments would cause the ship to sink further. Passengers in the lower decks and engine rooms had to go through the longest way out with the smallest amount of time before the water spread or they got trapped between the doors. We expect, that passengers from the E, F, G decks have significantly lower chances to have survived the collision. Important to highlight, that we only had this information for 295 out of the 1309. We eventually do not expect this variable to significantly influence the chances of survival.

1.1.9 Transformation of Categorical Variables

We carried out two types of variable transformation. The `Sex` variable was converted into binary by assigning 0 to male and 1 to female passengers. Secondly, we performed codification on the following features: `Social titles`, `Cabin`, `Embarked`. `Ticket number` was also cleaned: prefixes were capitalized, extracted and encoded into dummy variables. This step is necessary as most of the machine learning algorithms we implemented and describe in Section 2 require numerical input and output variables.

1.2 FEATURE SELECTION

Our combined dataset consisted of 54 features after dropping redundant / converted features. To obtain the best final dataset, we tried to apply some feature selection techniques. Firstly, we looked at the features importance graph, presented in Figure 1. We can see that there are non-significant features, mostly indicators related to the Ticket variable.

We performed the following feature selection techniques:

- **Principal Components Analysis**, a dimensionality reduction method, after normalizing the features. Unfortunately, this technique did not give better results than using the whole dataset.
- **`SelectKBest` function** of scikit-learn, using {10,20,50,60} features. However, the resulting scores did not significantly improve.

Finally, we plotted Figure 1, which depicts feature importance for the Random Forest algorithm.

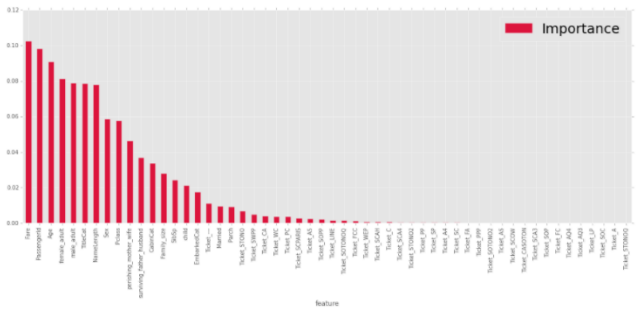

Figure 1. Feature Importance – Random Forest

We decided to work with the following subset of features: `Age`, `male adult`, `female adult`, `child`, `perishing mother wife`, `surviving father husband`, `Fare`, `Parch`, `Pclass`, `SibSp`, `TitleCat`, `CabinCat`, `Sex`, `EmbarkedCat`, `Family_size`, `NameLength`.

SECTION 2: MODEL TUNING & COMPARISON

Given that we face a binary classification problem where the goal is to predict which passengers survived the tragedy, we should explore the capabilities of classification algorithms. We chose to implement the following models: Random Forest, Support Vector Machines, Logistic Regression, K Nearest Neighbors, Neural Network, Gradient Boosting and XGBoost. We decided to run these models both with and without parameter tuning where it was possible.

2.1 MODEL PERFORMANCE

2.1.1 Random Forest (RF)

RF is an ensemble learning method, which constructs multiple decision trees and outputs the mode of the classes. RF is a robust approach to correct for the overfitting usually obtained via decision trees. As RF is implemented via several parameters, the training accuracy can depend on the tuning order. We decided to optimize the following parameters: `n_estimators`, `max_feature`, `min_samples_leaf`, `min_samples_split` and `max_depth`. To prevent overfitting, we applied cross-validation both during parameter optimization, and training score computation. Best results on both training set and Kaggle leaderboard were obtained with an optimized version of RF.

2.1.2 Support Vector Machines (SVM)

The SVM algorithm also includes the optimization of model complexity into its estimation. SVM is more efficient given a high dimensional problem. There are fewer parameters to be tuned compared to RF and these can be optimized at the same time, whence no strategy is needed. Like with the RF algorithm, we applied cross-validation for both parameter tuning and score computation. SVM model gives satisfying results with tuned parameters.

2.1.3 Gradient Boosting (GBM)

The gradient boosting algorithm combines weak “learners" such as decision trees into a single strong learner. The difference between this algorithm and the previous ones is that it is an iterative algorithm based on gradient descent. Parameter optimization in case of the GBM algorithm also needs a strategy just like with RF, as the tuning order matters and parallel optimization may explode execution time. This method gives good results on both train and test sets, but not better than RF.

2.1.4 Other models

In addition of the previous models, others have been implemented to see if they could give satisfying results too. Every model is different and more or less efficient on a given type of data.

Firstly, we implement a Logistic Regression. One of the first method used to predict a binary variable, logistic regression is a simple classification algorithm with only a few parameters to tune. It has given satisfactory results on Kaggle but not the best ones.

K Nearest Neighbors algorithm’s efficiency depends on the choice of the parameter K, that we tuned using cross-validation. The algorithm did not perform well on the training set and neither on the Kaggle.

We also tried to implement more flexible models such as Neural Network. For this method we have several parameters to tune such as `hidden_layer_sizes`, `activation`, `solver`, `learning_rate` and `alpha`. Given its computationally expensive nature, we adopted the same strategy as for RF or GBM for parameter tuning.

Finally, as GBM gave encouraging results, we decided to use the XGBoost library. XGBoost is a recent boosting algorithm which is an optimized and distributed alternative to Gradient Boosting. XGBoost’s advantage is parallel computation, so execution time should be smaller. However, it has a lot of parameters to tune, whence it requires a tuning strategy as well and a very powerful computer so we were not able to run that part but submission results with default parameters were similar to Gradient Boosting.

2.2 MODEL COMPARISON

To compare all the models presented before, we implemented a Monte Carlo cross-validation. 50 different train and test samples had been randomly sampled using the train.csv file. The accuracy scores for each method and each test sample were computed and plotted in the boxplot below (Figure 2). The figure indicates that the best performing and most stable models are: Random Forest, Boosting and SVM.

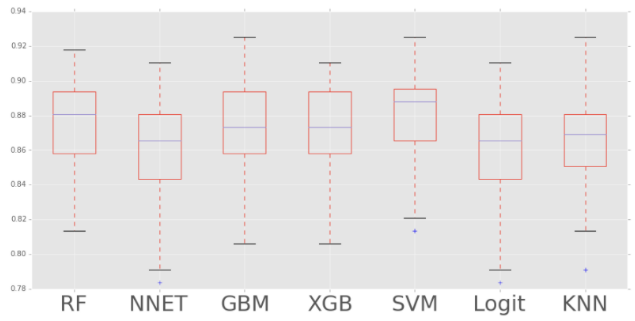

Figure 2. Model Comparison – Accuracy Scores

To improve our score, we decided to use the Voting Classifier that combines by pairs of two the three best models into a new one. This model results in a more powerful version with higher performance. Finally, we decided to keep the combination of Random Forest and SVM with which we obtained our best score on Kaggle that is 0.823 which puts us in the 255th place (TOP 3%). Our Kaggle submission scores are presented in Figure 3 where we can see, that the highest score is obtained by using the Voting method described above.

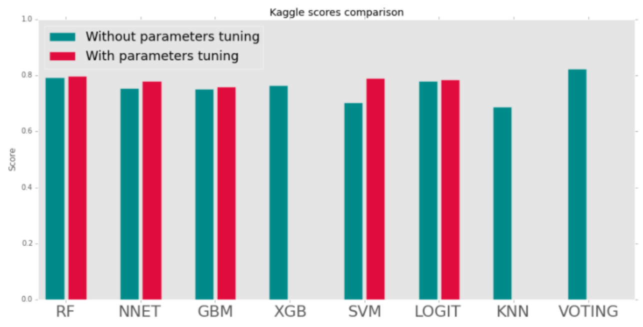

Figure 3. Model Comparison - Kaggle Submisson Results