














Overview	Data	Kernels	Discussion	Leaderboard	Rules	Team	My Submissions	Submit Predictions		
156	▼ 23	sword999						0.83253	3	16d
157	new	Some ML projects					0.83253	9	3h	
Your Best Entry ↑										
Your submission scored 0.83253, which is not an improvement of your best score. Keep trying!										
158	▼ 24	udaykrishna						0.82775	16	2mo
159	▼ 24	o_malandro						0.82775	6	2mo
160	▼ 24	Mehdi Saeedi						0.82775	21	2mo
161	▼ 24	kaixuan						0.82775	5	1mo
162	▼ 24	ShushuFang						0.82775	1	1mo
163	▼ 24	kezman9						0.82775	123	1mo
164	▼ 24	Wind_ZFL_Corps						0.82775	32	25d
165	▼ 24	ForrestLe						0.82775	12	22d
166	▼ 24	HanHsiangLiu						0.82775	29	20d
167	▼ 24	AmitSharma 3						0.82775	2	17d
168	▼ 24	Bayes_17214508						0.82775	10	16d

Titanic: Using Machine Learning to Predict the Survived of Disaster

Kaggle Team : Some ML projects
Team members :Tailai Zhang ;Xiaoyu Tian ;Dongxue shao ;Shenghua Du

1. Project Description

As we know that The Titanic sinking resulted in the loss of more than 1500 passengers because of the lacking of enough lifeboats. Although there was some element of luck involved in surviving the sinking, some groups of people were more likely to survive than others, such as women, children, and the upper-class.

Now we have a dataset including the passengers' information which can be used to establish and evaluate our model to predict whether a passenger can survive or not. Thus the problem can be expressed as a two-class classification problem. Our goal is to design a model that achieves high classification accuracy using machine learning methods.

2. Feature Engineering

1) Understanding the data structure

First let us have an overlook at the training data and the testing data.

Training Data (891 entries)	Testing Data (418 entries)
<pre><class 'pandas.core.frame.DataFrame'> RangeIndex: 891 entries, 0 to 890 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 dtypes: float64(2), int64(5), object(5) memory usage: 83.6+ KB</pre>	<pre><class 'pandas.core.frame.DataFrame'> RangeIndex: 418 entries, 0 to 417 Data columns (total 11 columns): PassengerId 418 non-null int64 Pclass 418 non-null int64 Name 418 non-null object Sex 418 non-null object Age 332 non-null float64 SibSp 418 non-null int64 Parch 418 non-null int64 Ticket 418 non-null object Fare 417 non-null float64 Cabin 91 non-null object Embarked 418 non-null object dtypes: float64(2), int64(4), object(5) memory usage: 27.8+ KB</pre>

For training data, we have 891 entries. 'Age', 'Cabin' and 'Embarked' contain missing values.

For testing data, we have 418 entries. 'Age', 'Fare' and 'Cabin' contain missing values. Among all the columns, 'Survived' is a categorical feature with '0' refers to 'Death' and '1' refers to 'Survival'. This is what we are going to predict with machine learning tools.

Age, Pclass, Sex, Embarked: As we can see, there are different kinds of features referring to passenger's information. 'Pclass' is ordinal features. 'Sex' and 'Embarked' are categorical features. 'Age', 'SibSp', 'Parch' and 'Fare' are numerical ones. Both alphanumeric data and missing values appear, so it seems that we have to do some extractions and transformations before using them.

Name: What we intend to achieve here is to get more variables out of passenger name. Because using the name in our model will be a lot like using a passenger's id - each passenger has an almost unique value. Instead, we want to look at passenger similarity (or difference) at a higher level. In this case, the honorary title helps group similar passengers, and this title is present in the name variable. We used regular expressions to extract substrings which end with a dot (e.g. Miss., Master., etc.) from the Name column and put this result into a new column named "Title".

PassengerID: The PassengerID is the serial number of the passenger. It didn't include information. So we will drop the column.

2) Modify the features

1. Fill in the missing values in 'Embarked', 'Cabin' and 'Fare' (ProcessCabin.py, fill_n_scale.py)

Embarked: we found that all the missing value is on the group of Pclass 1, and Pclass 1's sample are either S or C, and more than half are S. So we fill in the missing values as S.

Cabin: we set the missing value as 'U0', and separate all the values to 'Cabin Letter' and 'Cabin Number'. The reason why we modify this feature is that we think that the 'Cabin letter' may be important – it shows how far the cabin to the outside. And as for the 'cabin number', maybe there are some patterns of the people in the same room. For the Fare, we use the median of each Pclass to fill in the NAs.

2. Modify other features (processFare.py, ProcessName.py)

Ticket: we separate the number and letter.

Name: we find a lot of which have titles or second name (in the parentheses) and we think they might be important. So, we construct two new features, title and number of names, to the data set. And for the title, we generally separate all the titles into 4 class: 'Mr.' and 'Mrs.' for the male and female with strong social background; 'Master' and 'Miss' for the male and female without strong social background.

3. Fill in missing values in Age (data_processing.py)

For the missing values in Name, we use the RandomForestRegressor model to fit other features and to predict the missing terms. The cross-validation R^2 is about 40%, and we think for a regression problem, it is good enough to use – after all, it could be better than just use the mean (or median) value of each class.

4. Standard-scale and make interaction features (interaction.py)

Lastly, for each feature we use StandardScaler to preprocess the data. And for Age and Fare, we use qcut to bin the data to 4 groups (same number of samples per bin).

Next, we apply 4 operations, plus, minus, times and divide to each two of the features and make the total number of features to 357 (it is the 'dfdf.csv'). While the pre-processing will generate a huge number of high-correlated and redundant features, and we will drop them later. For example, later we create a new data set which drop features with more than 0.9 correlation (113 features). (remove_corr.py)

3. Model Tuning and Comparison

1) Modeling with learning algorithms (Model_tunning.ipynb)

We used three different datasets: datasets from our comprehensive feature engineering (dfdf.csv), dataset from dfdf.csv but dropped Age1 and Age columns, dataset from a simplifier feature engineering (see Jupyter notebook file for more detail).

We trained our datasets and used GridSearchCV package to tune the following five models: 1) Random Forest; 2) SVM; 3) KNN; 4) Decision Tree; 5) Logistic Regression

- a) Random Forest:
 - a. As after feature engineering, we decided to drop out all the columns that are highly correlated (above 0.9): we found out that the best score can be achieved with `n_estimator = 70`, `min_samples_split = 7`.
 - b. Then we tried to use `SelectKBest` method based on `f_classif` (F-test), and found out that the cross-validation score is slightly below the previous one.
 - c. Thirdly, we selected features based on `feature_importance` from `SelectFromModel` package. The result is slightly higher than the above two.
 - d. We tested our model using datasets (only drop `Age1` and `Age` columns) and simplified feature engineering. Both results are lower than the result from c)
 - e. Finally, we tried another feature engineering method (extract more features from `surname` and does not bin `age` column), and the result is surprisingly higher all the other methods. cross-validation score of 0.899 with params of `{'min_samples_split': 8, 'n_estimators': 2700}`
 - f. Best cross-validation score: 0.899 with method(e) and parameters of `{'min_samples_split': 7, 'n_estimators': 2700}`
- b) SVC
 - a) Same as above we tested our model on three different datasets.
 - b) Best cross-validation score: 0.823 with dataset from `dfdf.csv` (dropped `Age1` and `Age` columns) and parameters of `{'C': 1, 'gamma': 0.0001}`
- c) KNN
 - a) Same as above we tested our model on three different datasets.
 - b) Best cross-validation score: 0.804 with dataset from `dfdf.csv` (dropped `Age1` and `Age` columns) and parameters of `{'n_neighbors': 3}`
- d) Decision Tree: generally speaking decision tree method often underperformed Ensemble methods like Random Forests
 - a) Best cross-validation score: 0.798 with dataset from simplified feature engineering with default parameters.
- e) Logistic Regression
 - a) Best cross-validation score: 0.827 with dataset from `dfdf.csv` after “`SelectFromModel`” method and parameters of `{'C': 10, 'penalty': 'l2'}`

2) Weighted Vote and stacking

For the weighted voting, we choose KNN, Logistic Regression, SVC and Random Forest. We discard other methods because they are essentially similar to the existing methods and may bring bias to the voting: for instance, Decision Tree and Xgboost are using similar methods with random forest.

We use the Cross-validation score to set the weight. Because the score is close (around 0.84), using the absolute weight is meaningless. So we scale them manually as 2:1:1:4 according to the standard-scaled cross-validation score. Finally we got 0.832525.

About the stacking, we first try to make a new data set, features the results returned by different models and the samples are the train data and test data. And then we use new train set to fit models to predict the accurately `y_test`. However, it turned to something similar to the unweighted vote. I assume that the results from train set are generally overfitting, and predict even more poorly to the new data. The result is even worse than using single algorithm (0.73).