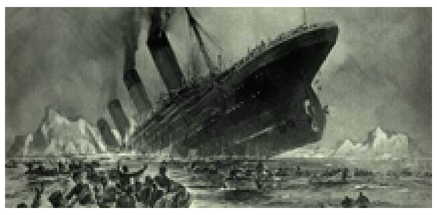


Knowledge • 5,771 teams

Titanic: Machine Learning from Disaster

Fri 28 Sep 2012 Sat 31 Dec 2016 (42 days to go)

Dashboard
Home
Data
Make a submission
Information
Description
Evaluation
Rules
Prizes
Frequently Asked Questio...
Getting Started With Excel
Getting Started With Pyth...
Getting Started With Pyth...
Getting Started With Rand...
New: Getting Started with R
Submission Instructions
Forum
Kernels
New Script
New Notebook
Leaderboard
Visualization

Getting Started With Random Forests

Getting Started *with Random Forests:* Titanic Competition

The last step in this series of tutorials will be a quick guide to multivariate models that truly learn from your data. As far as competing and doing well in Kaggle competitions, this is the most important page, however there is much more to learn beyond this. We hope this gives you a taste of how simple it can be to apply a sophisticated algorithm.

Quick Recap:

The previous tutorials allowed us to open the data in both Excel and in the scripting language Python. We created a simple model in both and wrote a csv to use as our submission. We are going to build on these skills to create what is known as a random forest™ (or, an ensemble of decision trees).

The beauty of Python

This is where the effort you put into getting Python running pays off! As was mentioned in the previous tutorial, Python has handy, built-in packages to help you manipulate arrays, compute complex math functions, read in complex file formats, and in this case, create random forests. You will need to install the scikit-learn package, which has the handy RandomForestClassifier class. (If you originally installed the Anaconda distribution of python, then scikit-learn is already installed too.) For more on this package, visit the scikit random forest page.

What is a Random Forest?

As with all the important questions in life, this is best deferred to the Wikipedia page. A random forest is an ensemble of decision trees which will output a prediction value, in this case survival. Each decision tree is constructed by using a random subset of the training data. After you have trained your forest, you can then pass each test row through it, in order to output a prediction. Simple! Well not quite! This particular python function requires floats for the input variables, so all strings need to be converted, and any missing data needs to be filled.

How do I clean and fill?

In the previous pandas tutorial you gained the skills to clean and fill in data using the pandas package. Now you can put that into practice.

Not all types of data can be converted into floats. For example, Names would be very difficult. In these cases let's decide to neglect these columns. Although they are strings, the categorical variables like male and female can be converted to 1 and 0, and the port of embarkment, which has three categories, can be converted to a 0, 1 or 2 (Cherbourg, Southamption and Queenstown). This may seem like a non-sensical way of classifying, since Queenstown is not twice the value of Southampton-- but random forests are somewhat robust when the number of different attributes are not too numerous.

Converting from categorical strings to floats is intuitive. However, filling in data can be more tricky. Some data cannot be trivially filled (such as Cabin) without complete teams

players

entries

knowledge of every cabin and ticket price for the entire ship. Nonetheless, Fare price can be estimated if you know the class, or the age of a passenger can be estimated using the median age of the people on board. Fortunately for us, the amount of missing data here is not too large, so the method for which you choose to fill the data shouldn't have too much of an effect on your predictive result.

My data is complete and floating nicely, I want to predict

Using the predictive capabilities of the scikit-learn package is very simple. In fact, it can be carried out in three simple steps: initializing the model, fitting it to the training data, and predicting new values.

Note that almost all of the model techniques in scikit-learn share a few common named functions, once they are initialized. You can always find out more about them in the documentation for each model. These are

some-model-name.**fit( )**

some-model-name.**predict( )**

some-model-name.**score( )**

At this point, it is assumed that you have read in the training data into an array train_data, where the first column [0] is the Survived column, that the Name, Cabin, and Ticket columns have been removed, and also that the Gender and Embarked columns have been converted to numbers. (If you just completed the previous pandas tutorial this means you also need to drop the PassengerId column.)

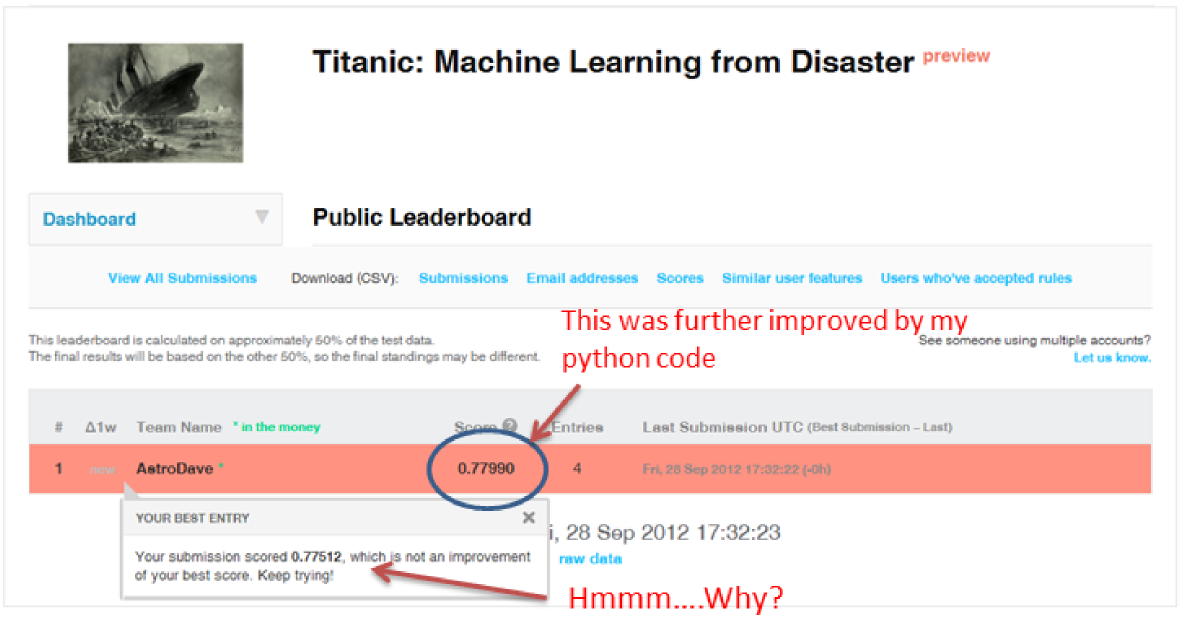
```
# Import the random forest package
from sklearn.ensemble import RandomForestClassifier

# Create the random forest object which will include all the parameters
# for the fit
forest = RandomForestClassifier(n_estimators = 100)

# Fit the training data to the Survived labels and create the decision trees
forest = forest.fit(train_data[0::,1::],train_data[0::,0])

# Take the same decision trees and run it on the test data
output = forest.predict(test_data)
```

There is much more to this script, which you can find on the Data page.

The output will be an array with a length equal to the number of passengers in the test set and a prediction of whether they survived. You can change the parameters as you see fit, as described on the RandomForestClassifier documentation page.

Word of Warning!!!

Looking at the leaderboard, I now see:



Titanic: Machine Learning from Disaster preview

Dashboard

Public Leaderboard

View All Submissions Download (CSV): Submissions Email addresses Scores Similar user features Users who've accepted rules

This leaderboard is calculated on approximately 50% of the test data.
The final results will be based on the other 50%, so the final standings may be different.

See someone using multiple accounts? Let us know.

This was further improved by my python code

#	Δ1w	Team Name * in the money	Score	Entries	Last Submission UTC (Best Submission – Last)
1	new	AstroDave *	0.77990	4	Fri, 28 Sep 2012 17:32:22 (-0h)

YOUR BEST ENTRY

Your submission scored 0.77512, which is not an improvement of your best score. Keep trying!

i, 28 Sep 2012 17:32:23

raw data

Hmmm....Why?

It appears that I have not improved my score! This seems strange, as your initial thoughts are "This is more complicated, therefore should be better!" This gives us three lessons to bear in mind:

1. A simple model is not always a bad model. Sometimes, concise, simple views of data reveal their true patterns and nature.
2. This is not the final score for my new submission! I have not done as well on the public leaderboard, but who knows what the private score may hold? I made my previous model on the assumptions of the training data: we still don't know how these will hold up in the private leaderboard.
3. Because the data set is very small, the differences in scores can be just one or two flips in decisions between survived or not survived. This means it will be very hard to determine the quality of the model from this data set. The aim of our Titanic Tutorial was to show you an easy way into more difficult problems, so don't be too disheartened if your super-complicated random forest doesn't beat the gender based model!

Conclusion

Now you have a budding set of skills to get you going in the larger Kaggle competitions. That, of course, is the primary prize of this Competition. There is still some improvement possible in your model, so please feel free to post comments on the forums, questions (and answers!). Chances are the experiences you have and questions you ask will be shared by others.

Know that a score of 0.79 - 0.81 is doing well on this challenge, and 0.81-0.82 is really going beyond the basic models! The dataset here is smaller than normal, so there is less signal to tap your models into. NOTE: You may see some people on the Leaderboard show accuracy of .90 up to even 100% -- but that's *probably not* from statistical modeling, just trying to look up the answers somewhere else on the Internet (which defeats the entire purpose.) So seriously, ignore those people! A discussion in the forum talks further about What accuracy should I be aiming for?

In terms of improving your model from here, you could consider any of these paths to try on your own:

- Revisit your assumptions about how you cleaned and filled the data.
- Be creative with additional feature engineering, so that your chosen model has more columns to train from.
- Use the sklearn documentation to experiment with different parameters for your random forest.
- Consider a different model approach. For example, a logistic regression model is often used to predict binary outcomes like 0/1.

Added note: If you would like to try out a different tool than Python for accomplishing the same kind of analysis, we now have links to outside tutorials for Getting Started with R.

We also have Further Reading with additional tutorials and suggestions.

Good luck with the competition!