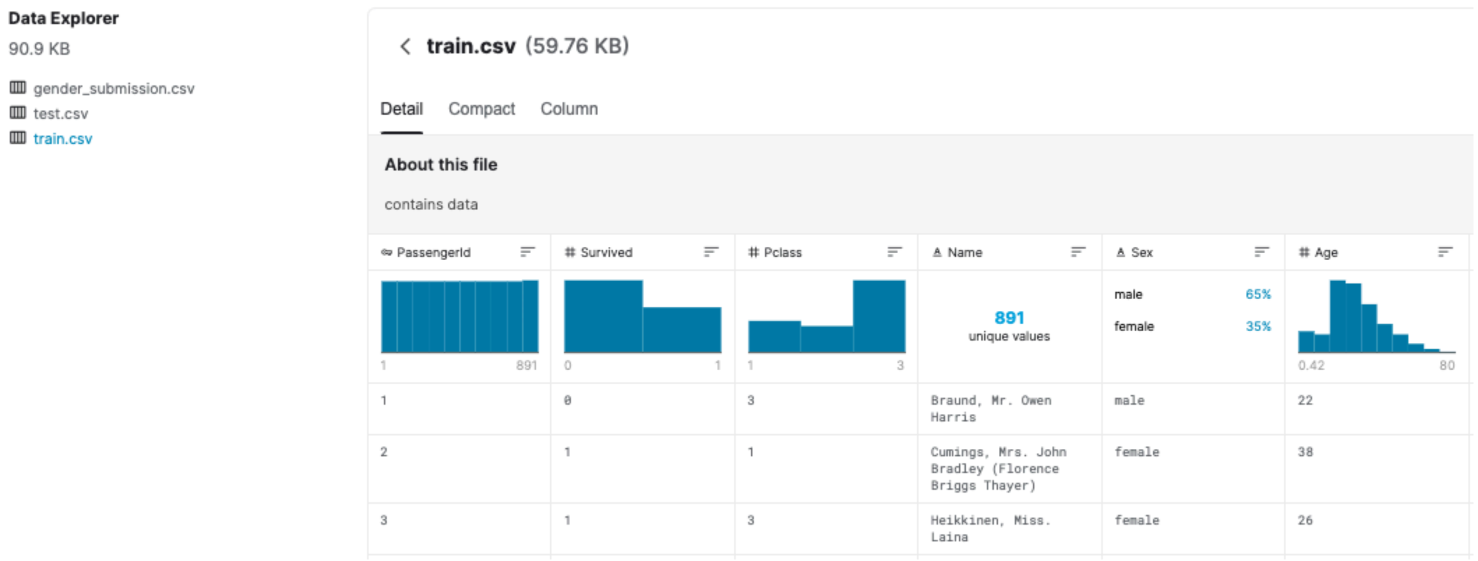
Term Project Tutorial
Titanic - Machine Learning from Disaster

The challenge here is we want you to use the Titanic passenger data (name, age, price of the ticket, etc) to try to predict who will survive and who will die.

There are three files in the data: (1) **train.csv**, (2) **test.csv**, and (3) **gender_submission.csv**.

(1) train.csv

train.csv contains the details of a subset of the passengers on board (891 passengers, to be exact -- where each passenger gets a different row in the table).



Data Explorer
90.9 KB

- gender_submission.csv
- test.csv
- train.csv

train.csv (59.76 KB)

Detail Compact Column

About this file

contains data

PassengerId	Survived	Pclass	Name	Sex	Age
1 – 891	0 – 1	1 – 3	891 unique values	male 65%, female 35%	0.42 – 80
1	0	3	Braund, Mr. Owen Harris	male	22
2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Thayer)	female	38
3	1	3	Heikkinen, Miss. Laina	female	26

The values in the second column (**"Survived"**) can be used to determine whether each passenger survived or not:

- if it's a "1", the passenger survived.
- if it's a "0", the passenger died.

For instance, the first passenger listed in **train.csv** is Mr. Owen Harris Braund. He was 22 years old when he died on the Titanic.

(2) test.csv

Using the patterns you find in **train.csv**, you have to predict whether the other 418 passengers on board (in **test.csv**) survived.

(3) gender_submission.csv

The **gender_submission.csv** file is provided as an example that shows how you should structure your predictions. It predicts that all female passengers survived, and all male passengers died. Your hypotheses regarding survival will probably be different, leading to a different submission file. But, just like this file, your submission should have

- a **"PassengerId"** column containing the IDs of each passenger from **test.csv**.
- a **"Survived"** column (that you will create!) with a "1" for the rows where you think the passenger survived, and a "0" where you predict that the passenger died.

Importing Data Frames

```
import numpy as np
import pandas as pd
```

NumPy is a library for the Python programming language, that adds support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.
Pandas is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series.

```
train_data = pd.read_csv("train.csv")
```

```
train_data.head()
```

Storing the train data into the 'train_data' variable and displaying the first 5 rows of the data.

```
test_data = pd.read_csv("test.csv")
test_data.head()
```

Storing the test data into the 'test_data' variable and displaying the first 5 rows of the data.

```
women = train_data.loc[train_data.Sex == 'female']["Survived"]
rate_women = sum(women)/len(women)

print("% of women who survived:", rate_women)
```

Checking the percentage of women who survived in the training dataset is calculated in the above snippet.

```
men = train_data.loc[train_data.Sex == 'male']["Survived"]
rate_men = sum(men)/len(men)

print("% of men who survived:", rate_men)
```

Checking the percentage of men who survived in the training dataset is calculated in the above snippet.

```
from sklearn.ensemble import RandomForestClassifier

y = train_data["Survived"]

features = ["Pclass", "Sex", "SibSp", "Parch"]
X = pd.get_dummies(train_data[features])
X_test = pd.get_dummies(test_data[features])

model = RandomForestClassifier(n_estimators=100, max_depth=5, random_state=1)
model.fit(X, y)
predictions = model.predict(X_test)

output = pd.DataFrame({'PassengerId': test_data.PassengerId, 'Survived': predictions})
output.to_csv('submission.csv', index=False)
```

```
print("Your submission was successfully saved!")
```

I built what's known as a **random forest model**. This model is constructed of several "trees" (there are three trees in the picture below, but we'll construct 100!) that will individually consider each passenger's data and vote on whether the individual survived. Then, the random forest model makes a democratic decision: the outcome with the most votes wins!

The code cell below looks for patterns in four different columns (**"Pclass"**, **"Sex"**, **"SibSp"**, and **"Parch"**) of the data. It constructs the trees in the random forest model based on patterns in the **train.csv** file, before generating predictions for the passengers in **test.csv**. The code also saves these new predictions in a CSV file **submission.csv**.

Random Forest grows multiple decision trees which are merged together for a more accurate prediction.

The logic behind the Random Forest model is that multiple uncorrelated models (the individual decision trees) perform much better as a group than they do alone. When using Random Forest for classification, each tree gives a classification or a “vote.” The forest chooses the classification with the majority of the “votes.” When using Random Forest for regression, the forest picks the average of the outputs of all trees.

The key here lies in the fact that there is a low (or no) correlation between the individual models—that is, between the decision trees that make up the larger Random Forest model. While individual decision trees may produce errors, the majority of the group will be correct, thus moving the overall outcome in the right direction.

By following the tutorial in kaggle I was able to get a prediction score of **0.77511**.

submission.csv
Complete · 8d ago
0.77511

Contribution

In the tutorial given in Kaggle I used "Pclass", "Sex", "SibSp", and "Parch" features for training the model and prediction. By analyzing the data set I figured out that the “Age” column can also be a factor in an individual survival rate.

But there are too many values missing in the “Age” column may be due to human error. This may lead to misprediction, so to avoid this I followed an approach of replacing all the missing values with NaN and then using the median of the whole data and replacing that with all the NaN values in our data. This will remove all the missing values and give us a better parameter to use in the prediction.

I tried using a few more different parameters but was not successful with them, my score even went to **0.77272.**

```
train_data.isnull().sum()
```

Finally, After adding the "Age" as one of the features and training the data I was able to increase my prediction score from **0.77511** to **0.7799**.

```
train_data = train_data.replace(" ",np.NaN)
```

```
train_data = train_data.fillna(train_data['Age'].median())
```

```
test_data = test_data.replace(" ",np.NaN)
```

```
test_data = test_data.fillna(train_data['Age'].median())
```

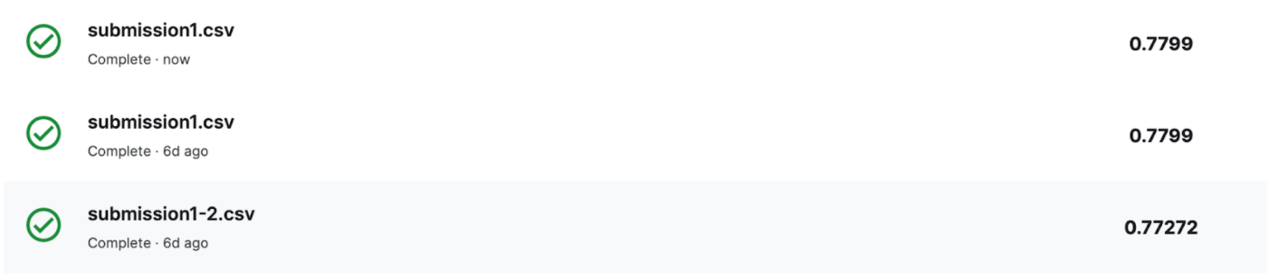
```
from sklearn.ensemble import RandomForestClassifier

y = train_data["Survived"]

features = ["Pclass", "Sex", "SibSp", "Parch","Age"]
X = pd.get_dummies(train_data[features])
X_test = pd.get_dummies(test_data[features])

model = RandomForestClassifier(n_estimators=600, max_depth=5, random_state=1)
model.fit(X, y)
predictions = model.predict(X_test)

output = pd.DataFrame({'PassengerId': test_data.PassengerId, 'Survived': predictions})
output.to_csv('submission1.csv', index=False)
print("Your submission was successfully saved!")
```


submission1.csv
Complete · now
0.7799

submission1.csv
Complete · 6d ago
0.7799

submission1-2.csv
Complete · 6d ago
0.77272

References

1. Kaggle – Got the source code and the tutorial description from here.

Source Code