titanic_survival_exploration

May 10, 2018

1 Lab: Titanic Survival Exploration with Decision Trees

1.1 Getting Started

In the introductory project, you studied the Titanic survival data, and you were able to make predictions about passenger survival. In that project, you built a decision tree by hand, that at each stage, picked the features that were most correlated with survival. Lucky for us, this is exactly how decision trees work! In this lab, we'll do this much quicker by implementing a decision tree in sklearn.

We'll start by loading the dataset and displaying some of its rows.

```
In [1]: # Import libraries necessary for this project
        import numpy as np
        import pandas as pd
        from IPython.display import display # Allows the use of display() for DataFrames

        # Pretty display for notebooks
        %matplotlib inline

        # Set a random seed
        import random
        random.seed(42)

        # Load the dataset
        in_file = 'titanic_data.csv'
        full_data = pd.read_csv(in_file)

        # Print the first few entries of the RMS Titanic data
        display(full_data.head())
```

	PassengerId	Survived	Pclass	\
0	1	0	3	
1	2	1	1	
2	3	1	3	
3	4	1	1	
4	5	0	3	

```
                                                Name     Sex   Age  SibSp  \
0                            Braund, Mr. Owen Harris    male  22.0      1
1  Cumings, Mrs. John Bradley (Florence Briggs Th...  female  38.0      1
2                             Heikkinen, Miss. Laina  female  26.0      0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
4                           Allen, Mr. William Henry    male  35.0      0

   Parch            Ticket     Fare Cabin Embarked
0      0         A/5 21171   7.2500   NaN        S
1      0          PC 17599  71.2833   C85        C
2      0  STON/O2. 3101282   7.9250   NaN        S
3      0            113803  53.1000  C123        S
4      0            373450   8.0500   NaN        S
```

Recall that these are the various features present for each passenger on the ship: - **Survived**: Outcome of survival (0 = No; 1 = Yes) - **Pclass**: Socio-economic class (1 = Upper class; 2 = Middle class; 3 = Lower class) - **Name**: Name of passenger - **Sex**: Sex of the passenger - **Age**: Age of the passenger (Some entries contain `NaN`) - **SibSp**: Number of siblings and spouses of the passenger aboard - **Parch**: Number of parents and children of the passenger aboard - **Ticket**: Ticket number of the passenger - **Fare**: Fare paid by the passenger - **Cabin** Cabin number of the passenger (Some entries contain `NaN`) - **Embarked**: Port of embarkation of the passenger (C = Cherbourg; Q = Queenstown; S = Southampton)

Since we're interested in the outcome of survival for each passenger or crew member, we can remove the **Survived** feature from this dataset and store it as its own separate variable `outcomes`. We will use these outcomes as our prediction targets.
Run the code cell below to remove **Survived** as a feature of the dataset and store it in `outcomes`.

```
In [2]: # Store the 'Survived' feature in a new variable and remove it from the dataset
        outcomes = full_data['Survived']
        features_raw = full_data.drop('Survived', axis = 1)

        # Show the new dataset with 'Survived' removed
        display(features_raw.head())
```

```
   PassengerId  Pclass                                                Name  \
0            1       3                            Braund, Mr. Owen Harris
1            2       1  Cumings, Mrs. John Bradley (Florence Briggs Th...
2            3       3                             Heikkinen, Miss. Laina
3            4       1       Futrelle, Mrs. Jacques Heath (Lily May Peel)
4            5       3                           Allen, Mr. William Henry

      Sex   Age  SibSp  Parch            Ticket     Fare Cabin Embarked
0    male  22.0      1      0         A/5 21171   7.2500   NaN        S
1  female  38.0      1      0          PC 17599  71.2833   C85        C
2  female  26.0      0      0  STON/O2. 3101282   7.9250   NaN        S
3  female  35.0      1      0            113803  53.1000  C123        S
4    male  35.0      0      0            373450   8.0500   NaN        S
```

The very same sample of the RMS Titanic data now shows the **Survived** feature removed from the DataFrame. Note that `data` (the passenger data) and `outcomes` (the outcomes of survival) are now *paired*. That means for any passenger `data.loc[i]`, they have the survival outcome `outcomes[i]`.

1.2 Preprocessing the data

Now, let's do some data preprocessing. First, we'll one-hot encode the features.

```
In [3]: features = pd.get_dummies(features_raw)
```

And now we'll fill in any blanks with zeroes.

```
In [4]: features = features.fillna(0.0)
        display(features.head())
```

```
   PassengerId  Pclass   Age  SibSp  Parch     Fare  Name_Abbing, Mr. Anthony  \
0            1       3  22.0      1      0   7.2500                          0
1            2       1  38.0      1      0  71.2833                          0
2            3       3  26.0      0      0   7.9250                          0
3            4       1  35.0      1      0  53.1000                          0
4            5       3  35.0      0      0   8.0500                          0

   Name_Abbott, Mr. Rossmore Edward  Name_Abbott, Mrs. Stanton (Rosa Hunt)  \
0                                 0                                      0
1                                 0                                      0
2                                 0                                      0
3                                 0                                      0
4                                 0                                      0

   Name_Abelson, Mr. Samuel     ...      Cabin_F G73  Cabin_F2  Cabin_F33  \
0                         0     ...                0         0          0
1                         0     ...                0         0          0
2                         0     ...                0         0          0
3                         0     ...                0         0          0
4                         0     ...                0         0          0

   Cabin_F38  Cabin_F4  Cabin_G6  Cabin_T  Embarked_C  Embarked_Q  Embarked_S
0          0         0         0        0           0           0           1
1          0         0         0        0           1           0           0
2          0         0         0        0           0           0           1
3          0         0         0        0           0           0           1
4          0         0         0        0           0           0           1

[5 rows x 1730 columns]
```

1.3 (TODO) Training the model

Now we're ready to train a model in sklearn. First, let's split the data into training and testing sets. Then we'll train the model on the training set.

```
In [5]: from sklearn.model_selection import train_test_split
        X_train, X_test, y_train, y_test = train_test_split(features, outcomes, test_size=0.2, r
```

```
In [10]: # Import the classifier from sklearn
         from sklearn.tree import DecisionTreeClassifier

         # TODO: Define the classifier, and fit it to the data
         model = DecisionTreeClassifier()

         model.fit(X_train, y_train)
```

```
Out[10]: DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
                     max_features=None, max_leaf_nodes=None,
                     min_impurity_decrease=0.0, min_impurity_split=None,
                     min_samples_leaf=1, min_samples_split=2,
                     min_weight_fraction_leaf=0.0, presort=False, random_state=None,
                     splitter='best')
```

1.4 Testing the model

Now, let's see how our model does, let's calculate the accuracy over both the training and the testing set.

```
In [11]: # Making predictions
         y_train_pred = model.predict(X_train)
         y_test_pred = model.predict(X_test)

         # Calculate the accuracy
         from sklearn.metrics import accuracy_score
         train_accuracy = accuracy_score(y_train, y_train_pred)
         test_accuracy = accuracy_score(y_test, y_test_pred)
         print('The training accuracy is', train_accuracy)
         print('The test accuracy is', test_accuracy)
```

```
The training accuracy is 1.0
The test accuracy is 0.810055865922
```

2 Exercise: Improving the model

Ok, high training accuracy and a lower testing accuracy. We may be overfitting a bit.

So now it's your turn to shine! Train a new model, and try to specify some parameters in order to improve the testing accuracy, such as: - `max_depth` - `min_samples_leaf` - `min_samples_split`

You can use your intuition, trial and error, or even better, feel free to use Grid Search!

Challenge: Try to get to 85% accuracy on the testing set. If you'd like a hint, take a look at the solutions notebook next.

In [38]:
```
from sklearn.metrics import make_scorer
from sklearn.model_selection import GridSearchCV

parameters = {'max_depth':[2,4,6], 'min_samples_leaf':[2,4,6,8,10], 'min_samples_split'

scorer = make_scorer(accuracy_score)

model = DecisionTreeClassifier()

grid_obj = GridSearchCV(model, parameters, scoring =scorer)

grid_fit = grid_obj.fit(X_train, y_train)

best_fit = grid_fit.best_estimator_
# TODO: Train the model

best_fit.fit(X_train, y_train)

# TODO: Make predictions

best_y_test_pred = best_fit.predict(X_test)

# TODO: Calculate the accuracy

Score = accuracy_score(best_y_test_pred, y_test)

print(Score)

best_fit
```

```
0.854748603352
```

Out[38]:
```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=6,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=6, min_samples_split=5,
                       min_weight_fraction_leaf=0.0, presort=False, random_state=None,
                       splitter='best')
```