Finish MySQL

Machine Learning with Python

The combination of excellent machine learning libraries and the data structures offered by pandas make data wrangling and modelling a joy in python. Though, I’m not a believer in the so-called competition between R and python (as a user of both languages I can tell you each has its advantages), I do however tend towards python because I have some software engineering experience under my belt. This post provides a broad overview of the different packages available for predictive modelling in python. I will discuss the output of the command line program through out, which explains each step taken. To get the code, you can download it here. Also, I recommend running this code at the command line as a python program, opposed to ipython in spyder.

Note: it is an unfortunate that the statistics and computer science disciplines use different terms to describe the variables handled. Note, that the independent variables are the same as features and the dependent variable is the same as the target.

To begin, you can download the program from here, then navigate to the folder containing titanicChallenge.py in your command line and enter: python titanicChallenge.py.

Preprocessing:

Hit enter to import the data. It should print out the imported data as a dataframe object. First, change we will make to the data set is to split it into the independent and dependent variables. Since we are trying to predict who will survive the titanic, the dependent variable y will store the column “Survived”, where ‘1’ stands for survived and ‘0’ did not survive. We will store all other variables in X. These are all potential independent variables

Next, lets look at the data we have. We can immediately see many potentially useful variables such as the passenger class, age, sex and family members for predicting who survives other such as passenger id or cabin are not so useful because they either clearly contain no relevant information about the passenger as is the case with passenger id or because they are simply missing way too much data such as is the case cabin. This also applies to ticket.

At first glance, this seems to apply as well to the names of the passengers but there is some hidden value there. Using a regular expression, we can extract the title of the person, Mr, Mrs, Sir etc. This may give some information about the social status of the passenger. So we group them as follows:

‘Female Common Title’ : ‘Miss’ , ‘Mlle’, ‘Mme’, ‘Mrs’, ‘Ms’

‘Male Common Title’: ‘Mr’, ‘Master’

‘Female Noble Title’: ‘Countess’, ‘Lady’

‘Male Noble Title’: ‘Don’, ‘Jonkheer’, ‘Sir’ \*Jonkheer is a dutch noble title and Don a noble title that belongs to languages that their roots in latin.

‘Military Title’: ‘Capt’, ‘Major’, ‘Col’

Imputation:

For this section, the first step is to check for missing values. The program returns a sum of all missing values by independent variable. Age is missing 177 values while embarked is missing 2. I take care of embarked first because it has such few missing values and it is a categorical variable, so we are limited in our techniques anyway. I use mode imputation, that is the most frequently appearing embarked location is used to fill in the missing values. In this case, the value will be ‘Southhampton.’

Next, for the age, there are two techniques I considered, first is using mean imputation and the second is regression imputation. I opt for regression imputation using linear regression with backward elimination because after testing both, the results I get with regression help to train a slightly better final predictive model. I broke down the linear regression into two steps, first, I get all possible polynomial features for the quantitative data to the second degree and use a greedy algorithm (RFECV – recursive feature elimination and cross validation selection ) that recommends the optimal features to use in the model based on evaluation of the f-statistic and p-values of the features generated. Similarly, in the second round, I generate polynomial features except this time I use the optimal features from the first run and include categorical variables and I only check for interaction effects. This two-stage process saves on computation and makes sense, since there is no use in squaring categorical variables. Ultimately, this produces a linear regression model with r-squared of approx. .30; a respectable number given the limited data and better performance than alternative imputation methods. As a final step, I ensure there are no more missing values.

Correlations:

Next, I produce a correlation matrix. There are instances where we have strong correlation relationships that don’t provide any insights into the data. For example, it is expected that whenever an observation is male that this would have a perfectly negative correlation with female. Another example can be seen with the embarkment location, only two locations such as: Cherbourg, Southampton need to be kept, because if it is not one of these two locations, it is implicitly the third location: Queenstown. The same logic can be applied with the engineered categorical variables of titles and passenger class. What we are doing is eliminating redundancies by targeting variables that have multicollinearity. The following categorical variables are dropped to prevent the dummy variable trap: “female”, “Queenstown”, “Pclass\_3” and “common\_title”.

Feature Scaling:

Next, a few more preprocessing steps. First, feature scaling is applied to: Age and Fare as a preprocessing step. This is necessary to ensure the range of all features are normalized so that each feature contributes approximately proportionately to the final distance. Another reason, it helps optimization algorithms to converge faster. Second, the data is split into training and test sets. Third, dimensionality reduction is optional, in the code principal component analysis is commented out, I decided to leave out for this exercise, since the number of features being used it not very large.

Machine Learning Algorithms:

Finally, we get to the machine learning algorithms. For this program, I present a menu at this point. There are 5 classification algorithms that can chosen from to train. In all cases, an exhaustive grid search is performed with cross validation exploring different hyperparameter settings to ensuring the best tuning. Each algorithm has its own set of hyperparameters for tuning. In this post, I won’t go through the results of all six algorithms, but just the best of the lot. I’ll breakdown the results and what they mean.

There are two trained algorithms whose metrics make them stand out above the rest: Logistic Regression and Decision Tree. The metrics I considered for selecting the best model are: Accuracy, Precision, Recall and F1-score. The most intuitive of is Accuracy and it is where I will begin.

First, lets take a look at the hypertuning results. For logistic regression,

Accuracy measures the ratio of correctly predicted observations to the total observations. This is great and simple metric to measure model performance. However, care must be taken to not solely rely on this metric, particularly if the data set is not symmetric. If the data set is symmetric, then the cost of a false positive and a false negative is the same, which is what the accuracy metric gives us. However, this is not the case, our data set is not symmetric: Survived = 342 (38%) and Did not Survive = 549(62%). Further, in the description of the data set from Kaggle, we are told that in the population 32% survived and 68% did not. So, while accuracy is important, F1 score is very important as well, to ensure that our model is making good predictions for both ‘Survived’ and ‘Did not Survive’.

Precision is the ratio of correctly predicted positive observations to the total predicted positive observations. In this context, how many passengers were correctly predicted to survive compared to how many were predicted to survive in total, that is both those who actually did survive and those that did not survive. Precision is an important metric for us, because we have fewer passengers who did survive in our data set and it would be easy to train a model that is bias toward predicting did not survive and push our accuracy score up.

Recall is defined as the number of true positives over the number of true positives plus the number of false negatives. In this context, how many passengers were correctly predicted to survive over those who were correctly predicted to survive plus those that were incorrectly predicted to not survive (so they actually survived).

F1 Score is the weighted average of Precision and Recall. Therefore, this score takes both false positives and false negatives into account. This is a particularly useful measure for asymmetric data sets and as a measurement of overall model performance since it measures both positive and negative predictions.