Kickstarter Status Classifier

Gabriel Cayón

Github Link

Abstract

Documentation for the Decision Trees hand in for Tec de Monterrey Campus Querétaro Sistemas Inteligentes Ago-Dic 2019 as a Kickstarter status classifier. The code cleans the dataset (and splits it into test and training sets), generates or validates a decision tree with the set parameters, logs the accuracy, prints a pdf with the resulting tree and saves the model in a pickle file. A Kickstarter dataset found in Kaggle is used with the intention of being able to predict the success of a project with a reasonable accuracy. Results show that using gini criterion, a *max_depth* of 10 and basically the preset settings gave the best results for prediction (with 66% accuracy).

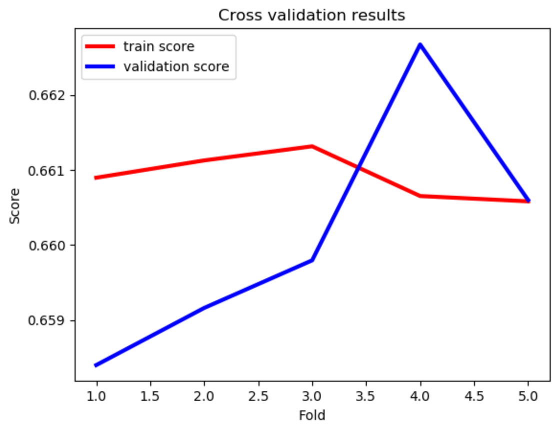

Fig. 1: Final model validation results

Introduction

A decision tree is a 'support tool' for visualizing decisions and their possible consequences by presenting them in a tree-like graph model (branches stem from 'parent' nodes up until a common root). Thanks to the evolution and convergence of statistical analysis and computational sciences we can model a decision tree in order to find patterns and predict instances in an applicable way. In data science and machine learning, decision trees are a supervised non-parametric learning method used for classification and regression. The branch nodes represent a conditional 'true' or 'false' selection of a certain dataset parameter which then creates a corresponding branch to the next conditional until the last node is found (called 'leaf node') that can finally categorize a sample.

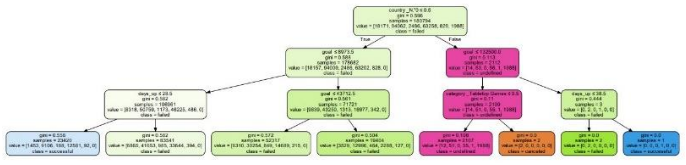

Fig. 2: Tree with max_depth 3

Crowdfunding is the practice of funding a project by raising 'small' amounts of money from a large number of people. Kickstarter is an online crowdfunding platform where projects get funded by the community in exchange of rewards.

After finding the database, I thought to myself that it would be really interesting to see if I could find a way to predict whether a project reaches a milestone or not by tweaking the initial parameters.

Development

Dataset

I found a rather big dataset of the platform when browsing Kaggle. The dataset has more than 300,000 samples, which could potentially give us a rather accurate model. On a first look, we see the website ID, name of the project, category, the date it was launched and the deadline, how much money was pledged, the state of the project, the backers and the country. The feature I'm most interested on is the **State**, so we will have to set this as our output *'Y'*. The rest of our columns, *'X'*, will be used to fit the model. The project's state can fall in one of these 6 categories: ['canceled', 'failed', 'live', 'successful', 'suspended', 'undefined']. I noticed after running many models however that the categories canceled, failed and suspended result in very noisy and inaccurate classifications. This is because these projects represent exceptions on the platform. It might be possible to find patterns but more data is definitely needed and perhaps a more suitable model.

Similar but more complex exercises found on the web seem to add more attributes to the data, ranging from user interaction (shares and comments by the backers) to how many Facebook friends the project creator has. This did seem to up the accuracy with which their models could predict results. Additional data that I propose as helpful could include historical funding data, the buzz the project has on social media, a linguistic analysis of the working title and perhaps an analysis of the 'pitch elements' (such as the impact of the promotional video and how the description of the project is worded). For the purpose of this delivery I'll try to get a model as good as possible with the data provided by this dataset.

Cleaning up the Data

The dates found in the dataset have some weird formatting, I used the already built Excel tools to subtract the initial date from the deadline, getting as a result a 'days_up' column.

I used the pandas library to get experience managing dataframes in the Python environment (instead of manually editing the Excel file). This let me change the code parameters and tune the data to experiment with different models and datasets quickly. Many cells were missing and the dataset had some unusable, erroneous data (ej. numbers in the category and main_category. columns, which are only expected to have strings). The ***split_csv()*** and ***prepare_dataset()*** functions I defined clean up the data by removing empty cells or invalid terms as well as dropping columns as needed.

Since we are going to validate our model after training it (more on this later), it's best if we shuffle all of our rows (so we can make sure our database isn't biased) and split our entire dataset in two parts, allowing us to use 90% of the data for a training set and 10% for a test set. If there is already a test and training set then these will not be generated again.

In order to translate categorical into numerical data I used One Hot Encoding with the function 'get_dummies()' in order to have the model working properly. In the case of the output, I tried both using strings for categories as well as an equivalent number to see if it made any difference (found none).

Model Fitting

After fixing the data, I used the *DecisionTreeClassifier*() class from the Scikit Learn module to set up a variable *clf* that will hold our model of the system. The parameters that we can change include:

- Criterion: choose the impurity factor to build the model with, ‘gini’ or ‘entropy’.
- Max depth: How many levels of depth our tree will have.
- Min. samples split: Minimum samples needed to perform a split.
- Min. samples leaf: Minimum samples needed to create a leaf node.
- Some more advanced stuff I’m not very familiar with yet (Random state, maximum leaf nodes, minimum impurity decrease, minimum impurity split, class weight and presort).

At a first instance, I included the pledged and the backers attributes, which correspond to final conditions. This does not make sense as we want to predict with only initial values. I did some testing and realized that even if I only dropped the status column I could classify with up to 99% accuracy the projects (useful if somehow the AI dropped its folders and needed to quickly categorize the data).

The next step is to use the ***fit*()** method to train the model with the input X and Y. I realized that I would be running multiple tests in order to find the most suitable hyperparameters for my trees, so I had to find a way to save and load my models for future use. I used the pickle library for this and with just a couple lines of code I got the saving and loading working. I also implemented a logger so I could have the historical data of the models run.

Conclusion

Results

In order to validate the model, I set up a validate function that runs for n number of ‘folds’, where it splits the dataset in two (a test and train set) and runs score tests. This will show us when our model score is way above than it can actually predict new values. After running it plots the resulting graph.

It seems to be that my optimal max_depth is 10, as going under drops the accuracy and anything above that overfits the model (which means, the model becomes too good at predicting the training dataset but misses new 'inputs').

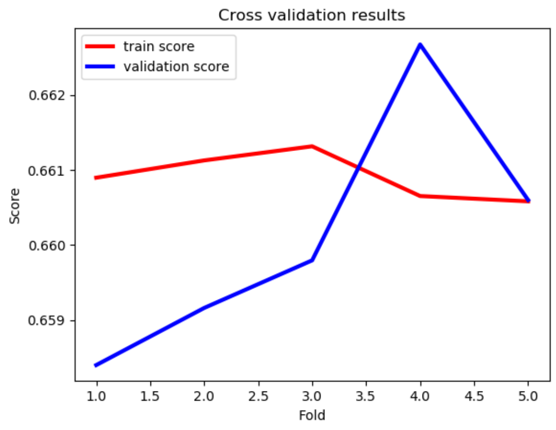

Fig. 1: Final model validation results

The most important values in order (that is, they have a higher gini value which means the feature moves biggest amount of samples with a single check) to determine the final status of a Kickstarter campaign are as follows:

- Goal
- Country: New Zealand
- Days up
- Category: Tabletop Games
- Main Category: Photography
- Currency: Australian Dollars
- Category: Hip-Hop
- Category: Short Video
- Main Category: Crafts
- Main Category: Design

I noticed that tweaking the criterion, minimun samples per split and minimun samples per split did not help my model (the accuracy only went down). Perhaps in the future when I'm more experienced with these kind of tools I'll be able to find better parameters without dropping accuracy or overfitting the model.

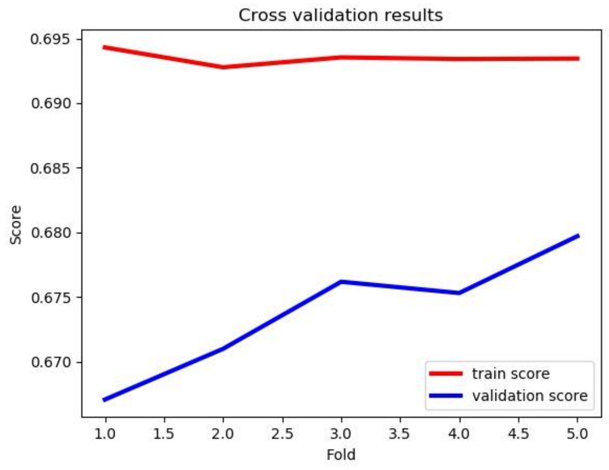

Fig. 3: Overfit model with max_depth of 15

Future Work

The next step would be to use another model and compare it to this one, maybe a random forest classifier can achieve a higher accuracy. I could also get more information from the projects and find features that give more interesting insights into the data.

Sources

Kemical. (2017). *Kickstarter Projects*. Obtenido de Kaggle: https://www.kaggle.com/kemical/kickstarter-projects

Scikit Learn. (2019). *sklearn.tree.DecisionTreeClassifier.* Obtenido de Scikit Learn: https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html