Classification of Kickstarter's successful or failed projects based on data crawled by a scraper robot (Web Robots)

Gloria GONZÁLEZ CURTO
Promotion février 2020

April 30, 2020

Contents

1 Summary

The aim of this project is to classify and ultimately predict the success or failure of a kickstarter fund rising campaign based on data monthly crawled from the kickstarter.com web site by a scraper robot (http://webrobots.io).

2 Problem definition and background

Kickstarter is an American corporation with a public benefit status launched in 2009 in the United States. The company initiated its international expansion in 2012. The kickstarter platform is open to backers from anywhere in the world and to creators from many countries.

Kickstarter maintains a global crowdfunding platform focused on creativity. Project creators choose a deadline and a minimum funding goal. If the goal is not met by the deadline, no funds are collected and the project is considered failed. Project backers usually receive rewards in exchange for their pledges. Once a project collects enough pledges to bypass the goal before a deadline stablished by the creator, its state changes to successful. Only at that moment pledges are collected from backers. From that point onward, there is no guarantee for project delivery. Kickstarter advises backers to use their own judgment on supporting a project. They also warn project leaders that they could be liable for legal damages from backers for failure to deliver on promises.

On one hand, from a backer point of view, it would be interesting to have an analytic model to help with the decision of whether to pledge for a project or not. On the other hand, from the project creators, it would be interesting to have a guide to help them optimize their chances of success.

3 Data description and pre-processing

Raw data were collected from https://webrobots.io/kickstarter-datasets/ in CSV format. The downloaded data correspond to a crawl executed on February 13 2020. Raw data are composed by 57 CSV files with 3500 to 4000 rows each. See the subsection 3.2 to access the profile report of the raw data. The raw original variables are:

- **backers_count**: integer representing how many people supported the project (backers).
- **blurb**: textual description of the project (around 150-200 characters).
- **category**: JSON-encoded description on the kickstarter's fixed categories: id, name, slug, position, parent_id, parent_name, color, urls.
- **converted_pledged_amount**: pleged amount converted to USD.
- **country**: country initials (i.e: FR for France):
- **country_displayable_name**: country name.
- **created_at**: date on Unix time format.
- **creator**: JSON-encoded description on project's creator. id, name, is_registered, chosen_currency, is_superbacker, avatar, small, medium, url_web, url_api.

- **currency**: currency acronym.
- **currency_symbol**.
- **currency_trailing_code**: boolean variable.
- **current_currency**: USD.
- **deadline**: crowdfunding deadline in Unix time established by the project's creator.
- **disable_communication**: boolean variable.
- **friends**.
- **fx_rate**: conversion rate.
- **goal**: fund rising goal in creator's fixed currency.
- **id**: project identifier.
- **is_backing**.
- **is_starrable**.
- **is_starred**.
- **launched_at**: launch date on Unix time format.
- **location**: JSON-encoded description of the project's creator location: id, name, slug, short_name, displayable_name, localized_name, country, state, type, is_root, expanded_country, url_web, url_location, url_api_nearby_projects.
- **name**: project's name.
- **permissions**.
- **photo**: JSON-encoded description on project's photos: key, full, ed, med, little, small, thumb, 1024x576, 1536x864.
- **pledged**: pledged amount in original country currency.
- **profile**: JSON-encoded description on project's profile: id, project_id, state, state_changed_at, name, blurb, background_color, text_color, link_background_color, link_text_color, link_text, link_url, show_feature_image, background_image_opacity, should_show_feature_image_section, feature_image_attributes, image_urls.
- **slug**: small textual description with - as spacers.
- **source_url**: url pointing to project's category site on kickstarter.
- **spotlight**: boolean representing if the project was featured on kickstarter web site.
- **staff_pick**: boolean representing if the project was picked by kickstarter staff.
- **state**: project's state initialy composed by 5 categories (successful, failed, canceled, suspended and live.
- **state_changed_at**: date in Unix time format.
- **static_usd_rate**: static rate for currency conversion into USD.
- **urls**: JSON-encoded description containing urls: project, rewards.

- **usd_pledged**.
- **usd_type**: domestic.

3.1 Data pre-processing

The following steps were followed to obtain a clean dataset for model training or EDA from raw data.

- **Join CSV files**. Raw data contain ambigous JSON-encoded fields that induce errors while parsing. To avoid those errors, ambigous JSON-encoded containing rows were filtered out while joining data into an unique file , see `pre_join.py`.
- **Deserialize JSON-encoded columns**. Columns and rows with missing information were dropped ("friends", "is_backing", "is_starred", "permissions") from raw data before JSON deserialization. JSON-encoded columns (*'category', 'creator', 'location', 'photo', 'profile', 'urls'*) were isolated from the rest of raw data and deserialized using a custom made function (*deserialize_in_batch*) by means of the json_normalize method from the pandas package and loads from the json package. The implementation is available at `pre_decode_JSON.py`).
- **Selection and pre-processing of deserialized variables**. The implementation of the following steps can be accessed at `pre_deserialized_to_one_hot_encoding.py`.
  - Drop category_slug because it contains partial redundant information with category_name and category_parent_name.
  - Fill category_parent_name missing values with their corresponding category_name value.
  - Exploration of location related variables: location_localized_name, location_country, and location_expanded_country
    - location_localized_name contains no missing values and 12977 unique values. Such a number of levels for a categorical variable might not be informative. Drop location_localized_name.
    - location_country and location_expanded_country are redundant. Drop location_country because is less human readable.
  - Profile related variables. Codification of profile related variables into a binary code reflecting the involvement of the project creator in the generation of a profile.
    - Re-codification of categorical binary variables: profile_state: "inactive"=0, "active"=1 profile_show_feature_image: False=0, True=1 profile_should_show_feature_image_section: False=0, True=1
    - The rest of the profile related columns are going to be coded in a binary choice variable (missing value = 0= , variable contains a project creator provided value =1). The following columns were processed: 'profile_name', 'profile_blurb', 'profile_background_color', 'profile_text_color', 'profile_link_background_color', 'profile_link_text_color', 'profile_link_text', 'profile_link_url'.
  - One hot encoding of categorical variables while keeping the original columns for EDA. The variables 'category_name', 'category_parent_name', 'location_expanded_country' were codified using pd.get_dummies and the argument drop_first was set to True.

- **Change date format and compute date derived variables**. The implementation is available at `pre_date_format_and_derived_variables.py`.

  First, date and time coding variables ('created_at', 'deadline', 'launched_at', 'state_changed_at') in the raw data are in Unix time format and need to be transformed into datetime. The following variables where computed from the original ones: weekday columns for each date variable, month columns for each date variable, year columns for each date variable.

  'initial_found_rising_duration' was computed as the difference in days between 'deadline' and 'launched_at'.

  'found_rising_duration' was computed as the difference in days between 'state_changed_at' and 'launched_at'.

  'project_set_up_duration' was computed as the difference in days between 'launched_at' and 'created_at'.

  Date variables were prunned before model training to remove the 'state_changed_at' derived ones.

- **Currency derived variables pre-processing**.

  The implementation is available at `pre_currency_and_other_variables.py`.

  The following original variables were dropped due to redundancies or because they are not informative for the model: 'current_currency', 'currency_trailing_code', 'fx_rate', 'converted_pledged_amount', 'pledged', 'slug', 'usd_type'.

  The variable 'usd_goal' was computed by multiplication of the variables 'goal' 'and 'static_usd_rate' to able to compare the pledge goals for all the projects. 'goal' and 'static_usd_rate' were dropped after the generation of 'usd_goal'.

  The variable 'is_starrable' was also dropped, because of lack of relevance.

  The variables 'disable_communication', 'spotlight' and 'staff_pick' were recodified by replacing False=0 and True=1.

  The variable currency was dummified using get_dummies (the original varaible was kept for visualization purpose).

- **State pre-processing (target variable)**.

  The implementation is available at `pre_currency_and_other_variables.py`.

  State is the target variable for classification. The raw variable is a categorical variable with 5 levels('successful', 'failed', 'live', 'canceled', 'suspended') and no missing values.

  The variable 'state_group' was generated by grouping the canceled and suspended projects into 'failed'. It contains then 3 distinct levels ('successful', 'failed' and 'live').

  The variable 'state_code' was generated by dummification from 'state_grouped' (failed = 0, successful=1, live=2). There are 114940 successful projects, 84311 failed projects and 6651 live projects.

- **Join the JSON decoded variables to the rest of the data**.

  The implementation is available at `pre_join_and_drop_live_state.py`.

- **Drop rows corresponding to projects on a "live" state**.

  The implementation is available at `pre_join_and_drop_live_state.py`.

- **Evaluate and drop duplicate rows**.

  The implementation is available at `pre_duplicates.py`. 21392 duplicate rows were detected and removed before further processing using pandas duplicated and drop_duplicates functions.

- **Drop variables carrying low information**.

  Data dimensions at this processing stage was 177859 rows and 428 columns. In the following scripts I will evaluate the informative potential of different subsets of variables and drop the less informative ones in order to reduce the number of binary encoded columns and reduce the overfitting potential.

  – Generation of the variable 'profile'.

    The implementation is available at `pre_profile_pruning.py`.

    'profile' was generated by addition of the following columns: 'profile_ state', 'profile_ name', 'profile_ blurb', 'profile_ background_ color']+ df['profile_ text_ color', 'profile_ link_ background_ color', 'profile_ link_ text_ color', 'profile_ link_ text', 'profile_ link_ url', 'profile_ show_ feature_ image', 'profile_ should_ show_ feature_ image_ section'. It represents an score accounting for profile completeness. The variables were dropped after the score was computed.

    The number of columns was reduced to 418.

  – Category name pruning.

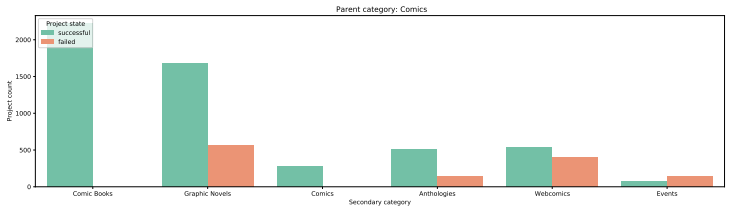
    The implementation is available at `pre_category_name_pruning.py`.

    'category_name_ori' encodes 159 subcategories of the 15 parent categories for kickstarter's projects. In order to evaluate the information content of each of these subcategory levels, plots and percentages of successful projects by secondary category were plotted to decide which one hot encoded columns under category_name to drop. Only categories with a percentage of successful projects of at least 55% (project success rate for the data is 53%) that were informative within their parent categories were kept.

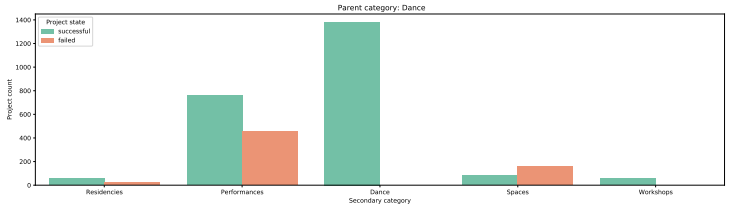
    After verification of plots and percentages, all comic, theater, dance secondary categories contain more successful tha failed projects, see Figure 1. As they are not more informative than their parent categories, those secondary categories were removed.

    The following subcategories were kept: 'Anthologies', 'Letterpress', "Children's Books", 'Art Books', 'Publishing', 'Literary Spaces', 'Fiction', 'Nonfiction', 'Playing Cards', 'Video Games', 'Tabletop Games', 'Games', 'Puzzles', 'Journalism', 'Public Art', 'Illustration', 'Painting', 'Art', 'Social Practice', 'Shorts', 'Narrative Film', 'Documentary', 'Webseries', 'Film & Video', 'Crafts', 'Photography', 'Product Design', 'Typography', 'Design', 'Classical Music', 'Country & Folk', 'Indie Rock', 'Pop', 'Music', 'Jazz', 'Comedy', 'Rock', 'Chiptune', 'Apparel', 'Accessories', 'Fashion', 'Gadgets', 'Hardware', 'Technology'.

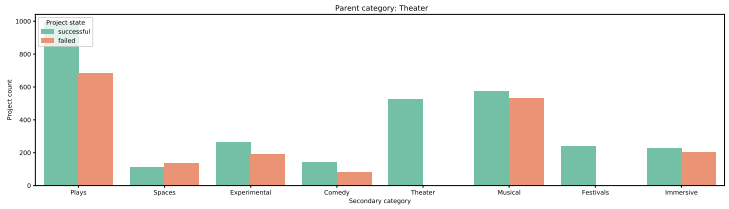
    The number of columns was reduced to 304.



(a) Count plot of successful and failed projects and subcategories of Comics principal category.



(b) Count plot of successful and failed projects and subcategories of Dance principal category.



(c) Count plot of successful and failed projects and subcategories of Dance principal category.

Figure (1) Count plots of successful and failed projects within categories.

– Country pruning. The implementation is available at `pre_country_pruning.py`.

  Projects in the dataset come from 198 countries. In order to reduce the number of variables, I performed count plots and percentages of successful projects by country. Only countries with more than 55% of successful projects and more than 50 projects were kept (United Kingdom, Hong Kong, Japan, Singapore, China, Poland, Israel, Taiwan, Czech Republic, Greece, Indonesia, Argentina, Kenya, Iceland, Ghana, Portugal, Slovenia, Finland).

  The number of columns was reduced to 125.

– Currency pruning. The implementation is available at `pre_currency_pruning.py`.

  Currency encodes 14 different levels. Count plots and percentages of succesfull projects were computed for tha variable 'currency_orig'. Currencies with more than 55% of successful projects were retained (GBP, HKD, SGD, JPY) and compared to the retained countries. All currency levels were dropped because the selected currencies correspond to countries that were already selected and the information was redundant.

  The number of columns was reduced to 112.

- **Drop rows with text in other languages than English**. The implementation is available at `pre_words.py`.

  The variables blurb and name contain brief descriptions of the project. In order to obtain valuable information for the model, these text variables are going to be processed to obtain keywords and compute a score due to their presence in the text. Projectś texts are writen in several languages. Several attempts to translate all text into English were performed using langdetect package. Unfortunately, the quantity of characters to translate was bigger than the established API limits. The estimation of the fees for the translation using the google translate API was 1200 €.

  English rows were detected and isolated using a custom made function. Briefly, for each row (project), blurb and name texts were tokenized and then compared to an extense list of 370101 English words (https://github.com/dwyl/english-words/blob/master/words_dictionary.json). An score was computed considering tokens with more than 3 characters and the lenght of the text. Rows with an score lower than 0.8 were considered not written in English and discarded (55269 rows).

  Data dimension after filtering was 122590 rows and 112 columns.

  To be able to extract category keywords associated to successful or failed projects, data were splitted in training and test sets at this point.

- **Split train and test**.

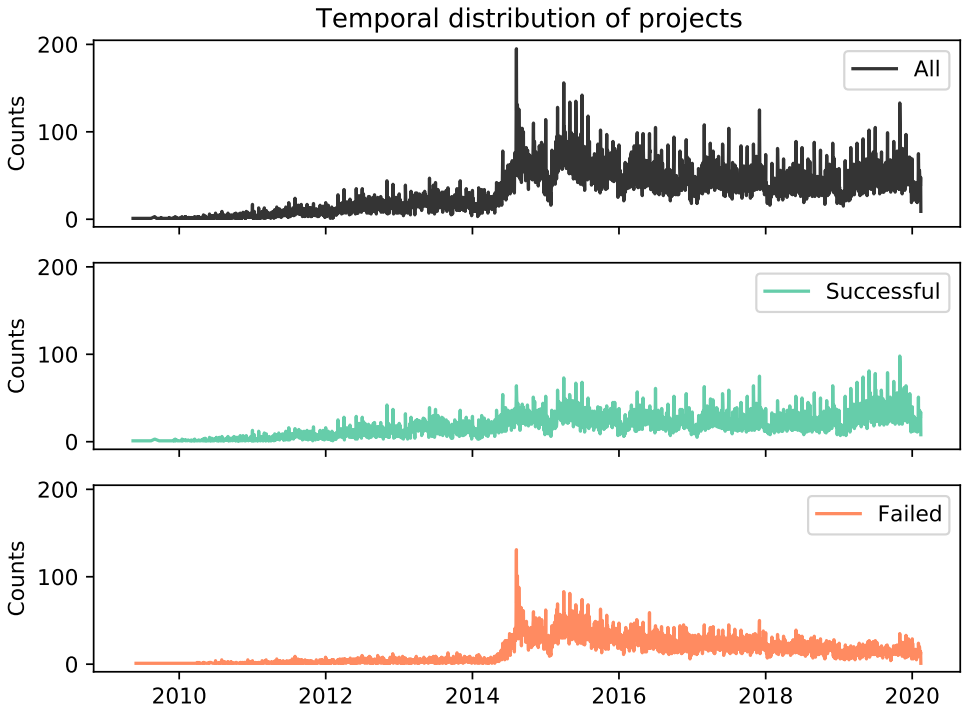
  The implementation is available at `pre_split_train_test.py`.

  Before splitting the data in train and test sets, I visualized the projects by year ("state_changed_at") and state (see Figure 2).

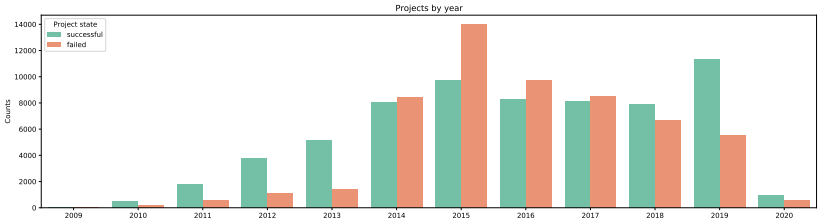
  The implementation is available at `pre_year_EDA.py` and

  `pre_data_prep_temporal_and_classification.py`.

  The target variable (state) shows no clear correlation with time, as we can observe in Figure 2. Moreover, our data are not evenly spaced in time. Therefore, after close exami-



(a) Total, successful and failed projects in time.



(b) Count plot of successful and failed projects by year.

Figure (2) Temporal dimension of the target variable.

nation of our data, I decided to implement a random split of train and test sets using the train_test_split functiikit learn model_selection package, a train size of 73 % a 37 as random seed. Before spliting, 15 columns were dropped, containing information not available prior to the project's state change. Dimensions of training set are 89490 rows and 97 columns, and dimensions of test set are 33100 rows and 97 columns.

- **Generation of the frequency score feature**.

  The implementation is available at `pre_text_mining.py`.

  Title and descriptions of the project were processed to obtain an score based on word frequencies in succesful and failed projects by principal category ('category_parent_name', 15 categories). Word frequencies were computed using both 'blurb' and 'name' variables on the training set. Punctuation signs were replaced by blank spaces, words were tokenized (word_tokenize from nltk package), stop words were filtered (stopwords from nltk), and stemmed (PorterStemmer from nltk). Word frequency was computed using FreqDist (nltk package). The frequency score was computed for every project by adding the successful project frequency of every word present in its 'blurb' and 'name' variables and substracting the failed project frequencies. The score was normalized by text lenght.

3.2 Exploratory data analysis

- **Profile reports** generated multiple times during data pre-processing using the pandas profile package (https://pypi.org/project/pandas-profiling/) to help with data visualization. Raw, train and test data htlm profiling reports can be found at my professional website.

  The implementation is available at `EDA_profiling.py` and `EDA_tt_profile.py`.

  Pandas profiling generates an statistical description of the data and each variable, computes interaction and correlation plots and warns about potential sources of problems such as high correlation, duplicates, high cardinality for categorical variables and skewed or zero containing variables. These reports (available at http://gloriagcurto.info/EDA_profiles) were extremely useful to quickly identify duplicated rows, variables with high numbers of missing values and also non numerical variables present in the train and test sets that are incompatible with XGBoost modeling.

- **Word clouds** were generated from the train and test sets by principal category and successful or failed projects using WordCloud.

  The implementation is available at `pre_text_mining.py`.

  Word clouds were representative of each category and sometimes were helpful to identify subcategories with high rates of failed projects (such as food trucks within Food, see Figure 3p). The majority of the most frequent 200 words were shared by successful and failed projects within categories but they also always present especific more frequent words, see Figure 3 and 13). If we take as an example the Comic word clouds (Figure 3c, 3d), we can see that both successful and failed projects contain graphic novel, adventure and comic with aproximatively the same frequency, but horror, fantasy and manga, are overrepresented in the failed projects. At the same time, anthologies, sci fi and girl terms are overrepresented in the succesful projects, suggesting possible differences in the succesful rate of different comic genres. Other words were consistenly found associated to successful projects across

categories. This is the case, for example, of enamel pin and hard enamel that were associated to succesful projects from the crafts (Figure 3e) and fashion categories (Figure 3k), and absent from the 200 most frequent words (see Figures 3f and 3l).

For clarity purpose, test set word clouds can be found at the end of the document, see Figure 13.

Word clouds were useful to understand project dinamics within categories and to evaluate the pertinency of the frecuency score feature. As we can identify trends in the word clouds associated to the target variable, the frequency score could potentialy help increase the evaluation metrics of our model.

The rest of data visualization was performed as a guide during data pre-processing (see Sections 3.1, 5 and Figures 1, 2, 9, 10, 11).

4 Data modeling

- **Data processing prior to XGBoost training** The implementation is available at `pre_data_model_train.py` and `pre_data_model_test.py`. First, the target variable in numeric format ('state_code') was isolated from the features and written to a file. Second, non numeric features were identified thanks to the pandas-profiling report and removed. Features were written to a file.
- **Methodological choices** The following packages were used for data modeling, evaluation and interpretation:
  - xgboost. Extreme Gradient Boosting classifier [1] was chosen due to its performance, escalabilty and built-in model interpretation capabilities. The tree based booster was prefered because of the intuitive interpretation of tree models.
  - Scikit learn matrics: classification_report, confusion_matrix [2].
  - bayesian-optimization [3] Bayesian optimization with cross validation was prefered as hyperparameter tuning method over a grid or a random search because it has been shown to obtain better results in fewer evaluations [4, 5].
  - SHAP SHAP [6, 7] was prefered over other model interpretation methods due to its mathematical strenght.
- **Model training, first iteration**

  The implementation is available at `train_first_model_wo_ts.py`.

  As a first attempt to model the data, a XGboost tree bosster classifier was initialized with default parameter and fitted to the training data. Evaluation metrics for predictions using both the training and test set were consistently found to show a score of 1, see tables 1, 2.

  In order to diagnose the reason behind such a perfect metrics, training set label randomization was performed using shuffle from scikit learn utils package. A new model was fitted with the shuffled labels and predictions for both the training and test sets were evaluated. As shown in tables 3, 5 and in their corresponding confusion matrices 4, 6, predictions were randomized after shuffling the labels, as proben by an accuracy of 50% in predictions for the test set. These results suggest that the perfect scores are not due to overfitting.

(a) Successful Art.

(b) Failed Art.

(c) Successful Comics.

(d) Failed Comics.

(e) Successful Crafts.

(f) Failed Crafts.

(g) Successful Dance.

(h) Failed Dance.

(i) Successful Design.

(j) Failed Design.

(k) Successful Fashion.

(l) Failed Fashion.

(m) Successful Film & Video.

(n) Failed Film & Video.

(o) Successful Food.

(p) Failed Food.

(q) Successful Games.

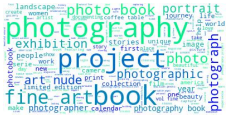
(r) Failed Games.

(s) Successful Journalism.

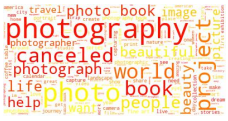
(t) Failed Journalism.

(u) Successful Music.

(v) Failed Music.



(w) Successful Photography.



(x) Failed Photography.

(y) Successful Publishing.

(z) Failed Publishing.

(aa) Successful Technology.

(ab) Failed Technology.

(ac) Successful Theater.

(ad) Failed Theater.

Figure (3) Wordclouds of successful and failed projects by principal category in the training set.

	precision	recall	f1-score	support
0	1.0	1.0	1.0	41522.0
1	1.0	1.0	1.0	47968.0
accuracy	1.0	1.0	1.0	1.0
macro avg	1.0	1.0	1.0	89490.0
weighted avg	1.0	1.0	1.0	89490.0

Table (1) Classification report for training set.

	precision	recall	f1-score	support
0	1.0	1.0	1.0	15423.0
1	1.0	1.0	1.0	17677.0
accuracy	1.0	1.0	1.0	1.0
macro avg	1.0	1.0	1.0	33100.0
weighted avg	1.0	1.0	1.0	33100.0

Table (2) Classification report for test set.

	precision	recall	f1-score	support
0	0.726857	0.325442	0.449587	41522.000000
1	0.604945	0.894138	0.721647	47968.000000
accuracy	0.630272	0.630272	0.630272	0.630272
macro avg	0.665901	0.609790	0.585617	89490.000000
weighted avg	0.661510	0.630272	0.595415	89490.000000

Table (3) Classification report for predictions of the train set using the model trained with shuffled labels.

	0	1
0	13513	28009
1	5078	42890

Table (4) Confusion matrix for predictions of the train set using the model trained with shuffled labels.

	precision	recall	f1-score	support
0	0.419684	0.184141	0.255971	15423.000000
1	0.522159	0.777847	0.624858	17677.000000
accuracy	0.501208	0.501208	0.501208	0.501208
macro avg	0.470921	0.480994	0.440415	33100.000000
weighted avg	0.474410	0.501208	0.452975	33100.000000

Table (5) Classification report for predictions of the test set using the model trained with shuffled labels.

Tree and importance plots for model interpretation implemented in the XGBoost package solved the mistery, see Figure 4. Interpretation plots showed that the decision tree had a

	precision	recall	f1-score	support
0	0.419684	0.184141	0.255971	15423.000000
1	0.522159	0.777847	0.624858	17677.000000
accuracy	0.501208	0.501208	0.501208	0.501208
macro avg	0.470921	0.480994	0.440415	33100.000000
weighted avg	0.474410	0.501208	0.452975	33100.000000

Table (6) Confusion matrix for predictions of the test set using the model trained with shuffled labels.

unique branch and the predictions were solely based in the value of the feature 'spotlight', that was identified as highly correlated with the target variable by the pandas-profiling reports.

After a search in the kickstater's website, I found a more exact difinition for 'spotlight' that the one provided with the data. Spotlight is a place for the projects on Kickstarter were the creators can communicate their project's progress, after they've been successfully funded.

I removed 'spotlight' from the features files and proceed to train a new model.

- **Model training, second iteration**

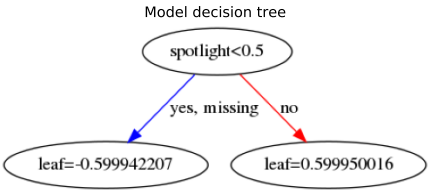
  The implementation is available at `train_wo_spotlight.py`.

  A bayesian Optimization function for xgboost was defined with the following parameters:

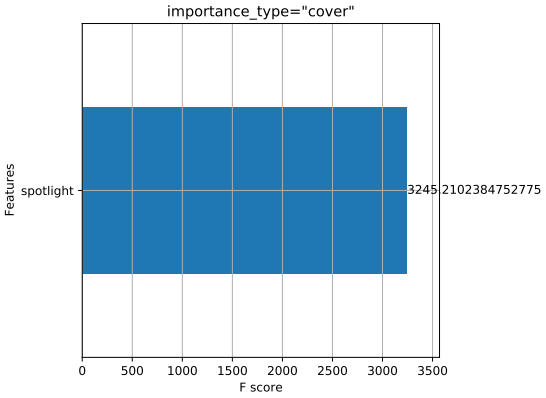
  - 'objective': 'binary:logistic'. Represents the learning task and the corresponding learning objective for a binary classification.
  - 'max_depth': int(max_depth). Maximum depth of a tree. Higher values make the model more complex and more likely to overfit. Range: $(0, \infty)$.
  - 'gamma': gamma. Minimum loss reduction required to partition a leaf node of the tree. Higher values make the algorithm more conservative. Range: $(0, \infty)$.
  - 'learning_rate':learning_rate. Step size shrinkage used in boosting iterations. Higher values make the process more conservative. Range: (0,1).
  - 'subsample': subsample. Subsample ratio of the training instancesprior to growing trees. Subsampling will occur once in every boosting iteration and prevents overfitting. Range: (0,1).
  - 'eval_metric': 'auc'. AUC measures the quality of the model's predictions irrespective of what classification threshold is chosen and their absolute values. For our classification task, we don't need well calibrated outputs and the cost of classification errors in our case is not critical. Moreover, a paralel hyperparameter optimization was evaluated using 'error' as metric with similar results. Range: (0,1).

  Cross validation was performed in 5 folds and 500 iterations, with and early stop at 100 rounds. Hyperparameter search was evaluated using "test-auc-mean" as metric.

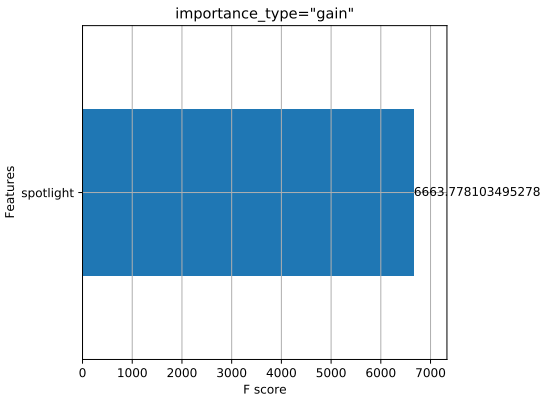
  The bayesian hyperparameter space was designed in a conservative way, to avoid overfitting of the data.



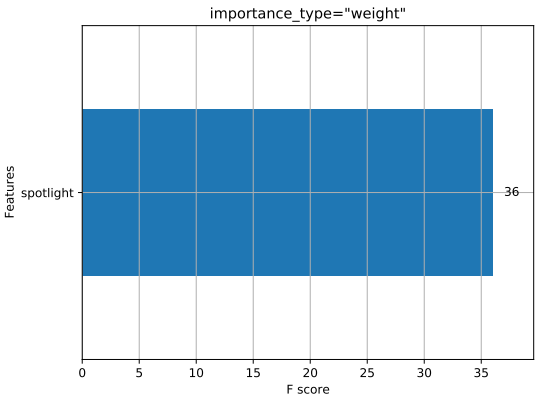
(a) Decision tree plot.



(b) Importance cover plot.



(c) Importance gain plot.



(d) Importance weight plot.

Figure (4) XGBoost built-in model interpretation plots.

– 'max_depth': (3, 8). Default value is 6.

– 'gamma': (0, 5). Default value is 0.

– 'learning_rate':(0, 1). Default value is 0.3.

– 'subsample':(0, 1). Default value is 1.

Bayesian optimization was performed for 5 iterations with 8 steps of random exploration and an acquisition function of expected improvement ('ie'). Best parameters were optimized to {'gamma': 3.262977074427203, 'learning_rate': 0.08113939174319618, 'max_depth': 7.84163489438336, 'subsample': 0.9915277034216099}.

The following tables contain evaluation metrics obtained by training the model with the best hyperparameters defined by bayesian optimization:

	precision	recall	f1-score	support
0	0.833378	0.896802	0.863927	41522.000000
1	0.904370	0.844792	0.873566	47968.000000
accuracy	0.868924	0.868924	0.868924	0.868924
macro avg	0.868874	0.870797	0.868747	89490.000000
weighted avg	0.871431	0.868924	0.869094	89490.000000

Table (7) Classification report for predictions of the train set using bayesian optimization hyperparameters.

	0	1
0	37237	4285
1	7445	40523

Table (8) Confusion matrix for predictions of the train set using bayesian optimization hyperparameters.

	precision	recall	f1-score	support
0	0.829042	0.885107	0.856157	15423.00000
1	0.893471	0.840754	0.866311	17677.00000
accuracy	0.861420	0.861420	0.861420	0.86142
macro avg	0.861256	0.862930	0.861234	33100.00000
weighted avg	0.863450	0.861420	0.861580	33100.00000

Table (9) Classification report for predictions of the test set using bayesian optimization hyperparameters.

	0	1
0	13651	1772
1	2815	14862

Table (10) Confusion matrix for predictions of the test set using bayesian optimization hyperparameters.

If we compare these classification reports, in tables 7, 9 to the ones of a XGBoost classifier with default parameters, in tables, 11, 13, we can observe similar metrics. However, the confusion matrices for the default hyperparameters model, see tables 12, 14, were slighly better than for the bayesian optimization model, see tables 8, 10. Therefore, I decided to save the default parameters model to a file for further exploitation. The following step was model interpretation.

	precision	recall	f1-score	support
0	0.835575	0.906652	0.869664	41522.000000
1	0.912773	0.845564	0.877884	47968.000000
accuracy	0.873908	0.873908	0.873908	0.873908
macro avg	0.874174	0.876108	0.873774	89490.000000
weighted avg	0.876955	0.873908	0.874070	89490.000000

Table (11) Classification report for predictions of the train set using default hyperparameters.

	0	1
0	37646	3876
1	7408	40560

Table (12) Confusion matrix for predictions of the train set using default hyperparameters.

	precision	recall	f1-score	support
0	0.828747	0.892369	0.859382	15423.000000
1	0.899351	0.839113	0.868188	17677.000000
accuracy	0.863927	0.863927	0.863927	0.863927
macro avg	0.864049	0.865741	0.863785	33100.000000
weighted avg	0.866453	0.863927	0.864085	33100.000000

Table (13) Classification report for predictions of the test set using default hyperparameters.

	0	1
0	13763	1660
1	2844	14833

Table (14) Confusion matrix for predictions of the test set using default hyperparameters.

5 Model interpretation

- **XGBoost built-in model interpretation**

  The implementation is available at `train_wo_spotlight.py`.

  According to the model's decision tree in Figure 5, the first variable that is considered for classification is 'profile', the second is 'usd_goal'. If we look at the importance plots (Figure 6), we can see that even though 'profile' is always important, its order of appearance is not conserved. Differences are due to different methods to compute the importance:

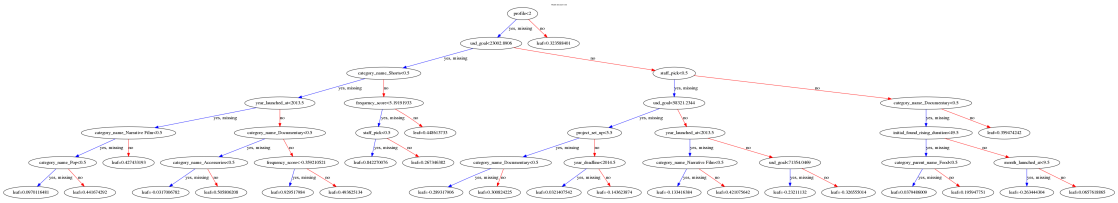


Figure (5) Decision tree plot.

- "weight": number of times a feature appears in a tree
- "gain": average gain of splits which use the feature
- "cover": average coverage of splits which use the feature where coverage is defined as the number of samples affected by the split

These model interpretation plots are not consistent, and conclusions about the contribution of each feature to the model are not reliable. In order to obtaint reliability in model interpretation, especially if we want to interpret the contribution of the features to single results , we need a method that is both consistent and accurate. SHAP plots [6, 7] are then the ideal choice.

- **SHAP model interpretation**

  The implementation is available at `SHAP_error_evaluation.py` and `EDA_features_important.py`. General SHAP model interpretation plots, such as the bar chart or the summary plot, allow to see wich of the features are contributing to the predictions and their ordering. We can observe in both plots (Figures 7,8, that 4 out of the first seven features were generated during data preprocessing. In particular, the profile and frequency scores make important contributions to the model.

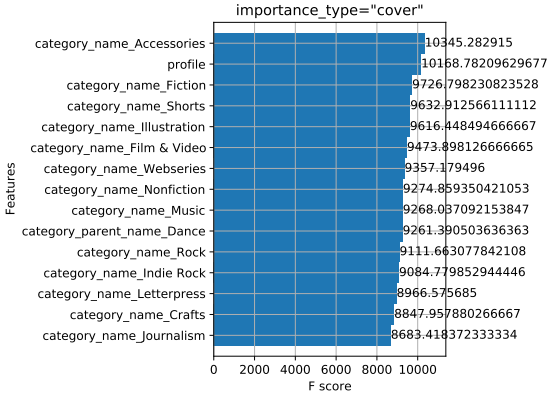
  Summary plot (Figure 8) also represents how the values of each feature contribute to the classification. We can see, for example, that high economic goals apper to be more frequently associated to failed projects. In fact, if we represent the distribution of 'usd_goal' in successful or failed projects [Figure 9), we can see that succesful projects have economic goals that are centered in lower goal values.

  We can also observe that higher values of the profile and frequency scores are associated to successful outcomes, see Figures 10 and 11.

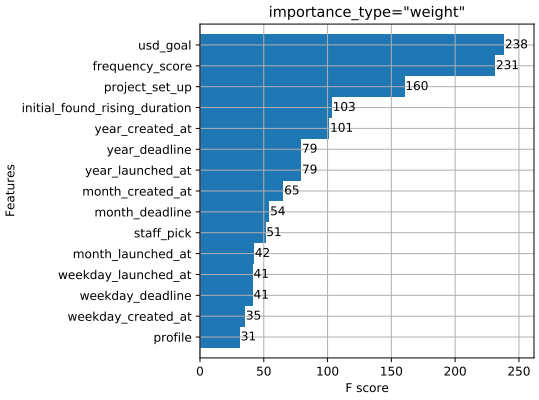
- **Prediction error evaluation using SHAP force plots**

  The implementation is available at `SHAP_error_evaluation.py`.

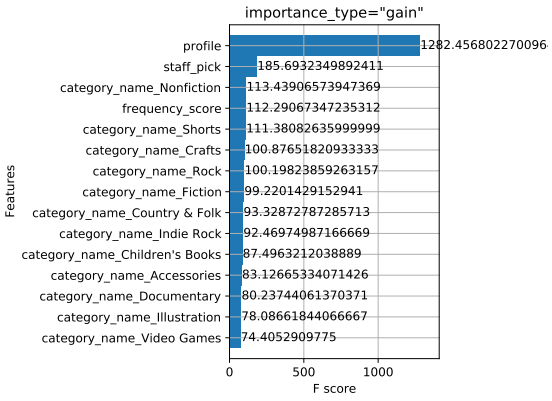
  SHAP force plots are representations of feature contribution for individual projects. If we compute force plots for missclassfied projects, we can easily evaluate prediction errors and potentially increase model performance. In fact, the analyzed missclassified projects showed a common pattern (Figures 12a, 12b). The analyzed prediction errors were found to have low profile scores and frequency scores that corresponded to the predicted class. Therefore, if this diagnosis holds true after evaluation of a bigger representation of the prdiction errors, we might introduce new features to buffer the predictive power of both scores (see 7).



(a) Importance cover plot.



(b) Importance weight plot.



(c) Importance gain plot.

Figure (6) XGBoost built-in model interpretation plots.

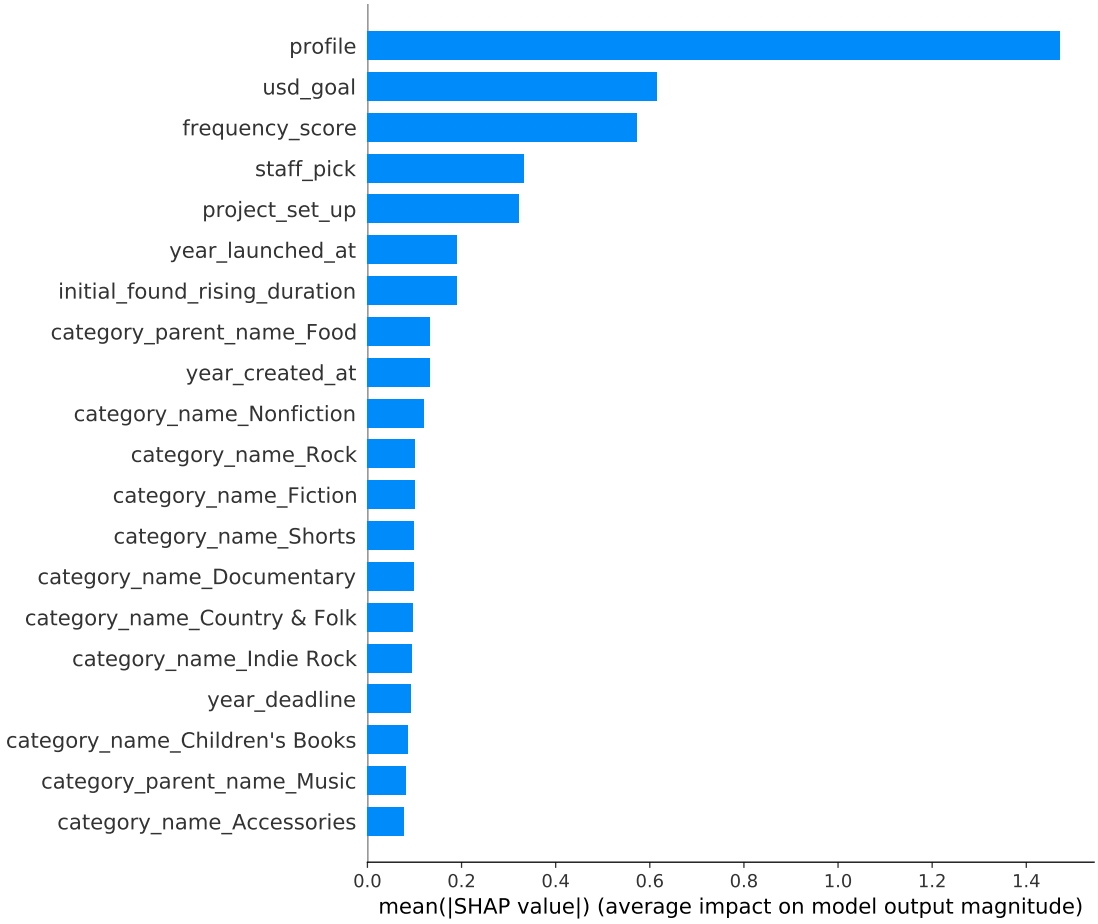


Figure (7) Mean importance chart.

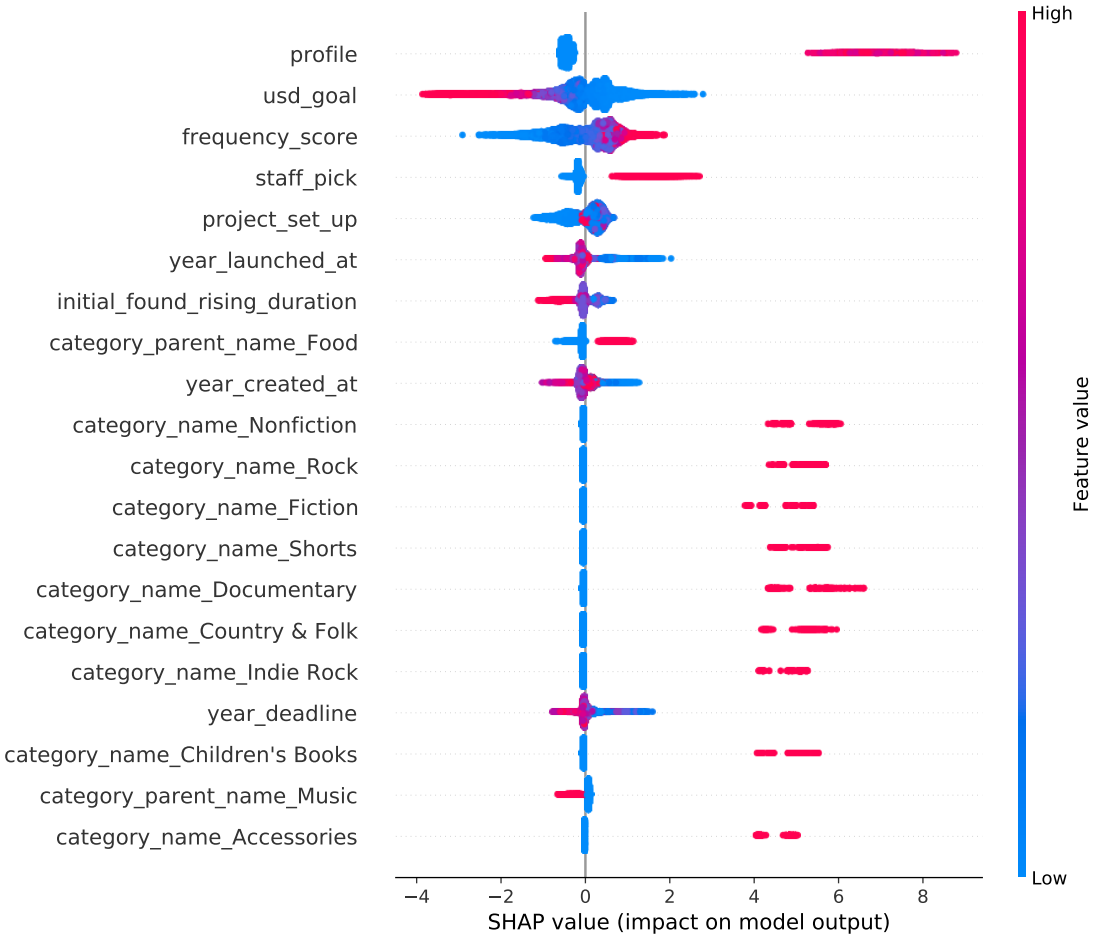


Figure (8) Summary plot.

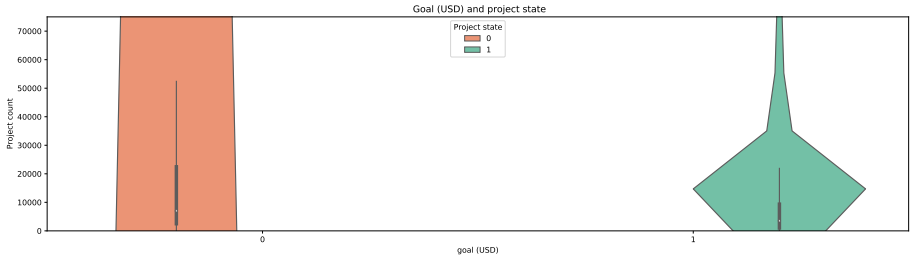


Figure (9) Goal distribution in successful and failed projects.

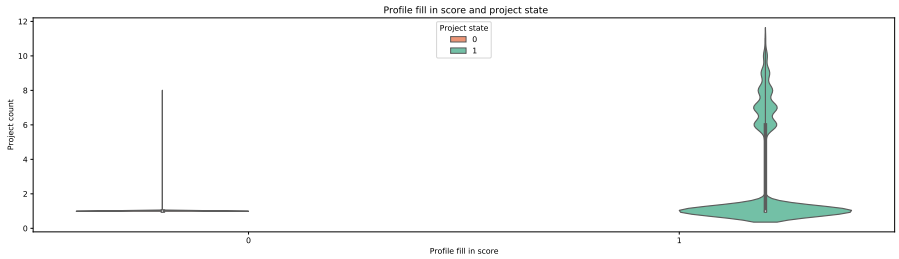


Figure (10) Profile score distribution in successful and failed projects.

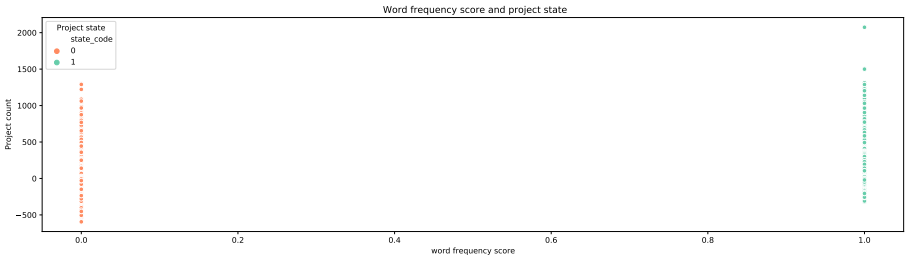


Figure (11) Frequency score distribution in successful and failed projects.

6 Development

6.1 Tools

The following development tools were employed:

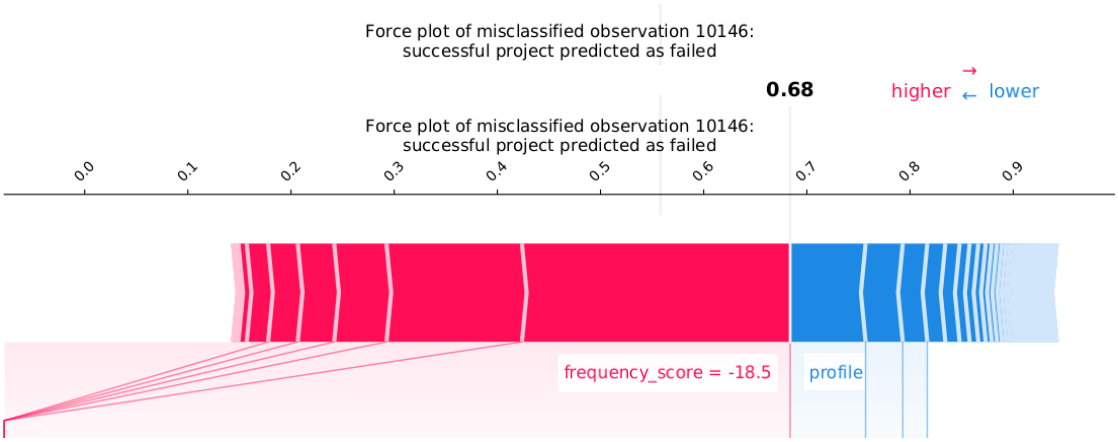
- Python
- Git and GitHub
- Visual Studio Code
- iPython and Jupyter Notebooks
- LaTeX

6.2 Implementation of the processing chain

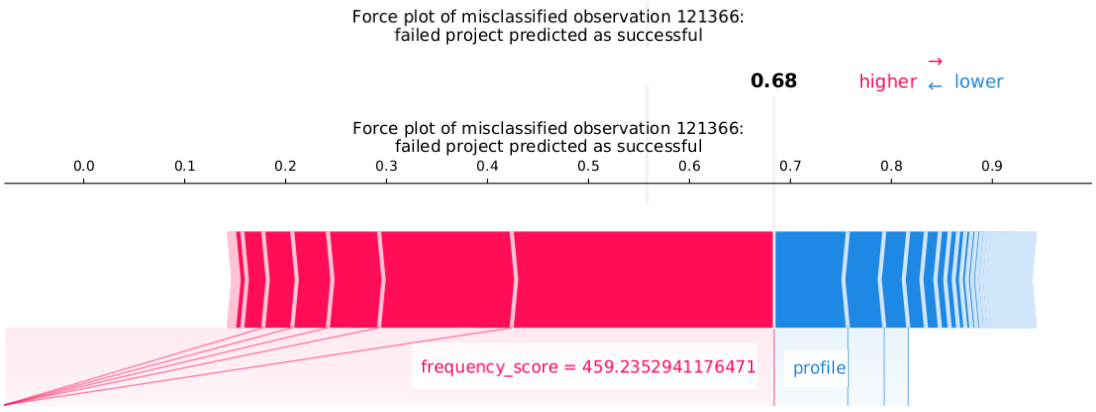
The project is made of several steps of different nature (such as data minin,g, feature extraction, applying several classifier, and assessment of model, among others). The structure of the code reflects this structure in the form of a processing chain. Each of the steps of the chain is processed by a different Python script, that must be run in the corresponding order. Splitting the whole processes in a chain made of independent modules also helps debugging and eventually maintaining this software due to component isolation[1].

The processing chain is made of the following Python scripts:

[1]The opposite case would be a monolithic system with complex interactions between modules, which is clearly harder to debug and maintain because of its inherent complexity.



(a) Successful predicted as failed.



(b) Failed predicted as successful.

Figure (12) Force plots for misclassified projects.

1. **pre_join.py**
2. **pre_decode_JSON.py**
3. **pre_deserialized_to_one_hot_encoding.py**
4. **pre_date_format_and_derived_variables.py**
5. **pre_currency_and_other_variables.py**
6. **pre_currency_and_other_variables.py**
7. **pre_join_and_drop_live_state.py**
8. **pre_duplicates.py**
9. **pre_profile_pruning.py**
10. **pre_category_name_pruning.py**
11. **pre_country_pruning.py**
12. **pre_currency_pruning.py**
13. **pre_words.py**
14. **pre_split_train_test.py**
15. **pre_year_EDA.py**
16. **pre_data_prep_temporal_and_classification.py**
17. **pre_text_mining.py**
18. **EDA_profiling.py**
19. **EDA_tt_profile.py**
20. **pre_data_model_train.py**
21. **pre_data_model_test.py**
22. **train_first_model_wo_ts.py**
23. **train_wo_spotlight.py**
24. **SHAP_error_evaluation.py**
25. **EDA_features_important.py**

Each of these scripts is decribed in its corresponding section in this report.

7 Conclusions

- Creators should focalize their efforts in their project's profile, because is determinant for the success of their campaign. In future iterations of this project, the generation of more profile related features could help improve the model's performance. Information coded as urls in the JSON encoded variables could be valuable in this context. For example, processing the information contained in the images related to the project, could potentialy increase model accuracy.

- Keywords were also valuable for the model performance, but might be the source of prediction errors. Other text mining methods could help to mitigate this effect, such as word embedding techniques. Incorporation of information related to rewards (available from a URL) could also be valuable. In addition, contextualizing keywords by combining Google Trends interest metrics, Instagram and, Twitter tags in months before and for the duration of a project's crowdfunding could also help to mitigate prediction errors. An evaluation of simultaneuos similar projects might also contribute to reduce errors.
- Information about creators and their previous projects could also be retrieved and contribute to model performance.

References

[1] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. *CoRR*, abs/1603.02754, 2016.

[2] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.

[3] Fernando Nogueira. Bayesian Optimization: Open source constrained global optimization tool for Python, 2014–.

[4] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical bayesian optimization of machine learning algorithms, 2012.

[5] James S. Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 2546–2554. Curran Associates, Inc., 2011.

[6] Scott Lundberg and Su-In Lee. A unified approach to interpreting model predictions. *CoRR*, abs/1705.07874, 2017.

[7] Scott M. Lundberg, Gabriel Erion, Hugh Chen, Alex DeGrave, Jordan M. Prutkin, Bala Nair, Ronit Katz, Jonathan Himmelfarb, Nisha Bansal, and Su-In Lee. From local explanations to global understanding with explainable ai for trees. *Nature Machine Intelligence*, 2(1):2522–5839, 2020.

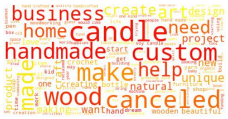
(a) Successful Art.

(b) Failed Art.

(c) Successful Comics.

(d) Failed Comics.

(e) Successful Crafts.



(f) Failed Crafts.

(g) Successful Dance.

(h) Failed Dance.

(i) Successful Design.

(j) Failed Design.

(k) Successful Fashion.

(l) Failed Fashion.

(m) Successful Film & Video.

(n) Failed Film & Video.

(o) Successful Food.

(p) Failed Food.

(q) Successful Games.

(r) Failed Games.

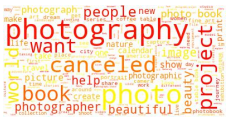
(s) Successful Journalism.

(t) Failed Journalism.

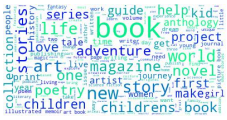
(u) Successful Music.

(v) Failed Music.

(w) Successful Photography.



(x) Failed Photography.



(y) Successful Publishing.

(z) Failed Publishing.

(aa) Successful Technology.

(ab) Failed Technology.

(ac) Successful Theater.

(ad) Failed Theater.

Figure (13) Wordclouds of successful and failed projects by principal category in the test set.