

IONIAN UNIVERSITY

DEPARTMENT OF INFORMATICS



Master's Thesis

Analysing Crowd-Funding Campaign Success Factors using Machine Learning

Constantine Xipolitopoulos

Supervisor: Katia Lida Kermanidis

Corfu

January 2021

Supervisor

Katia Lida Kermanidis, *Associate Professor*
Ionian University

Grading Committee

Adamantia Pateli, *Associate Professor*
Ionian University

Katia Lida Kermanidis, *Associate Professor*
Ionian University

Emmanouil Maragoudakis, *Professor*
Ionian University

This work is dedicated to a better 2021.

Abstract

In this past decade, an ever increasing number of businesses, small time creators, hobbyists, and entrepreneurs started using online crowd-funding platforms such as Kickstarter as a tool for funding their start-ups, kick start their ideas, or pre-sell their product to an audience in order to gauge interest. Consequently, there is a growing need for information that will allow these creators and entrepreneurs to plan, launch, and pivot their campaigns successfully. The aim of this work is to define the most important success factors for crowd-funding campaigns using machine learning and deep learning techniques. Towards this, a web scraper for gathering Kickstarter campaign data was developed and as a result, a novel dataset on this research topic was created, including information on descriptive, financial and linguistic features, of around 2500 finished crowd-funding campaigns that had been run on Kickstarter. Machine learning experiments that included various algorithms such as Decision Trees and Support Vector Machines, but also Deep Learning Neural Networks were employed in an attempt to predict the successful funding of the campaigns of the dataset.

Keywords: *crowd-funding, web-scraping, machine learning, deep learning*

Preface

The present work was developed during the years 2020-2021 in the Department of Informatics of the Ionian University in Corfu, and is submitted in partial fulfilment of the requirements for the Master's degree in Computer Science of the postgraduate "Research directions in Informatics" programme.

First of all, I would like to thank the professors of the "Research Directions in Informatics" programme and especially Ms. Katia Lida Kermanidis for her great help and guidance this past year. I also thank my parents Aristidis and Evangelia, my sister Eumorphia, my late grandparents, the Mpikou family and my extended family for their love and support. Finally, I thank my friends Theodosios, Vangelis, Sarantis, Giannis, Kosmas and Bryan for helping me get through 2020 in one piece.

Corfu, January 2021

Constantine Xipolitopoulos

Contents

1	Introduction	1
1.1	Crowd-funding & Kickstarter	1
1.2	Goal	2
1.3	Structure	3
1.4	Source Code	3
2	Literature review	5
3	The data set	9
3.1	Data	9
3.2	Feature Extraction	10
3.3	Statistical Information	11
4	The web scraper	17
4.1	Environment Preparation	17
4.2	The Layout	18
4.3	Selenium	19
4.4	Navigating	20
4.5	XPath	22
4.6	Scraping with BeautifulSoup	24
5	Exploring the data	27
5.1	Exploratory Data Observations	27
6	Experiments	35
6.1	Algorithms	35
6.2	Removing Features	36
6.3	Results	37

7	Deep Learning	39
7.1	Normalisation	39
7.1.1	Data Normalisation	39
7.1.2	Batch Normalisation	40
7.2	Designing the A.N.N.	40
7.2.1	Control	40
7.2.2	The Second Network	41
7.2.3	Activation Function	42
7.3	Training and Optimizers	43
7.3.1	Parameter Studies	45
7.4	Evaluation	46
7.4.1	Final Models	49
7.4.2	Deep Learning Methodology Analysis & Conclusions .	49
8	Conclusions	51

List of Figures

4.1	Here we can see the top banner section of a typical Kickstarter campaign.	18
4.2	In order to get to the comments we need to click on the Comments tab on the bottom part of the banner.	19
4.3	The comments left by people who in the end cancelled their pledge are hidden and need to be revealed using Selenium automation before scraping can begin.	23
4.4	Every 25 comments this button needs to be pressed using JavaScript, then 25 more comments are loaded in using an AJAX call by Kickstarter.	23
5.1	Average goal in USD (\$) for the two classes.	28
5.2	Average FAQ and Tier numbers on each class.	28
5.3	Success rate for campaigns launched during specific days of the week.	29
5.4	Success rate for campaigns launched during every month of the year.	29
5.5	Campaign length and incubation periods stats.	30
5.6	Staff picks and number of updates.	30
5.7	Success rate for the different countries Kickstarter operates in.	31
5.8	Average VADER comment score for both classes.	32
5.9	Success rate for the different main categories Kickstarter has.	33
7.1	The architecture of the layers of the first neural network (control) using a 10:1 scale.	41
7.2	The architecture of the layers of the second neural network using a 10:1 scale.	42

7.3	The ReLU activation function.	43
7.4	The flow of data and calculations inside every hidden layer of the neural network.	45
7.5	Parameter study of the percentage of probability for drop-out in each hidden layer output.	47
7.6	Parameter study of the number of training epochs.	47
7.7	Parameter study of the number of hidden layers in the network.	48
7.8	Parameter study of the learning rate of the optimiser algo- rithm "Adam".	48

List of Tables

2.1	Related work.	6
3.1	The features used directly from the Webrobots data set without any processing or alteration thereof.	11
3.2	The new features extracted from the Webrobots data set after processing.	12
3.3	The new features extracted from the new scraping data collected with the web scraper after processing.	14
3.4	Subset of the monetary statistics of the data set.	15
3.5	Subset of the descriptive statistics of the data set.	15
3.6	Subset of the linguistic statistics of the data set.	15
6.1	Features per category.	37
6.2	Naive Bayes classifier results matrix.	37
6.3	C4.5 Decision Tree classifier results matrix.	37
6.4	Ada-Boost C4.5 Ensemble Decision Tree classifier results matrix.	37
6.5	Performance of each algorithm. Internal 5-fold cross validation accuracy (CV 5-fold), and the final external validation metrics (Precision, Recall, F-1 Score).	38

Chapter 1

Introduction

1.1 Crowd-funding & Kickstarter

Online crowdfunding is the practice of funding an idea by pitching it directly to the end users through an internet platform, getting a large number of people to contribute to the idea, helping them realise it via the internet. Platforms such as Kickstarter and IndieGoGo have flourished using this idea, and helped a lot of aspiring creators realise their dream projects, their platforms giving access to a large number of small-time investors, and the general public. The term "crowd-funding" has largely become a synonym of "Kickstarter" the main platform we will be using to gather our data for this work, it is by far the largest platform for this reason.

These platforms also bypass the limiting legal hurdles that must be overcome to get funding from a traditional venture capital raising way. This can be both good (for the creators) but also bad (for the backers). Also it makes it simple for creators to fund their ideas without actually giving away a part of the company/idea as it is usually done in the venture capital world where investors fund a company through a venture capital firm, for exchange of a stake at the company. Because there have been a ton of examples of crowd-funding campaigns going down without any product being shipped to the backers, and the creators simply running away with the money. However with the amount of new campaigns being created every day, analysis of campaign success factors is increasingly needed to aid in the design and launch process of a campaign. However, because not all campaigns launched get funded, and because if they don't reach their funding goal they get no

funding at all, it's all or nothing. In this paper we define success factors for crowdfunding campaigns using machine and deep learning by analysing data gathered from finished Kickstarter campaigns and trying to predict whether a campaign was successful or not.

1.2 Goal

The goal of this thesis is to define the success factors for crowd-funding campaigns by using Artificial Intelligence (A.I.) techniques, such as Machine Learning (M.L.) and Deep Learning (D.L.). A novel web scraper for collecting Kickstarter campaign comment data was developed using the Python programming language, and several libraries such as Selenium, and BeautifulSoup, and made publicly available at [2] with an open source license for everyone to be able to use and expand upon. A novel data set was also created, consisting of 2,500 campaigns that run on Kickstarter and have concluded their campaign run. The features extracted from this data can be classified into three different categories: descriptive, financial and linguistic features. The descriptive category includes data that describe aspects of the campaign itself, such as the length of the campaign's funding round, the number of updates the campaign posted, the weekday it launched, the category that it was a part of, and the number of comments made on the campaign. The financial category concerns parts of a campaign related to monetary and performance statistics such as the number of reward tiers of a campaign, the amount of money it had as a funding goal, whether it was staff picked by Kickstarter to appear on the front page of the website, the average pledge of its backers, and how successful their crowd-funding was compared to what they were aiming for. The linguistic category is made up of features that were extracted from the campaign's pitch, long-form text of the campaign description, and comment made on the campaign page by backers of the campaign, such as the number of big words, average sentiment values, size of the campaign's pitch, size of the description length, and average size of the comments. Flesch–Kincaid readability and grade level complexity scores were also used for the user comments, and the campaign descriptions, as well as VADER sentiment scores for the same parts.

1.3 Structure

The structure of the rest of this work is organised as follows. Chapter 2 presents a literature review of the field and of related works. Chapter 3 describes the data scraping process alongside the feature extraction process through which new features were created by processing of the collected data. Chapter 4 describes the technical aspects of creating a web scraper for Kickstarter and building the data set. Chapter 5 is an exploratory analysis of various statistical facts about the data. Chapter 6 describes the machine learning experiments and analyses their results. Chapter 7 describes the deep learning part of the experiments, parameter studies on the neural networks, and their results. Finally, Chapter 8 draws conclusions based on the previous experiments and their results, and discusses future directions for the work and the field as a whole.

1.4 Source Code

The code implemented for this dissertation used the *git* code version control tool and can be found and downloaded openly from the git repository storage platform *GitHub* at the following link:

<https://github.com/constantinexisc/kickstarter-comment-scraper>

The repository has multiple files in it, the file "scraper.ipynb" is a *Jupyter Notebook* written in *Python*, it is meant to be an interactive ground for code where the user can manipulate cells, re-order them, execute them at their liking, and see the results inline with the code blocks. The file contains the Python source code for the web scraper that collected the data that went in the dataset that was used in this work. The file "feature-extraction.ipynb" contains the source code for extracting features from the raw web scraping data and turning the dataset into a readable ".csv" file format to be used in machine learning experiments. The file "experiments.ipynb" contains the code for every machine learning and deep learning experiment that was run for this work. The file "deeplearning.ipynb" has the code for the deep learning neural network experiments and parameter studies. And lastly the file "parameterstudies.ipynb" has the code for producing the bar graphs on the second to last chapter. The same repository also contains the raw data

collected by the scraper (in json format), as well as the final dataset (in csv format) used for the machine learning experiments, as well as an open source license and a convenient "readme" file with instructions on how to run the web scraper. The entirety of the source code for this work can also be found in printed form in the appendix.

Chapter 2

Literature review

Online crowd-funding platforms offer new opportunities for project creators to publish their creative ideas for the entire world to see, and more importantly share, gathering money in a new way that was not possible just a decade before. Consequently, the question of what influences the successful funding of projects, i.e. reaching the target amount of money, is a very interesting topic and also can prove quite fruitful for many people around the world if research can help with finding ways to better exploit this novel system. To tackle that fast, just in the last decade, quite a few studies were focused on identifying success factors of crowd-funding campaigns using either statistic analysis or machine learning methods.

The most common machine learning algorithms used were Decision Trees, Artificial Neural Networks, Genetic Algorithms, Bayesian Networks and Support Vector Machines as well as ensemble methods of some of the above algorithms. Current research on the specific research field is not sufficient enough for clearly identifying success factors. However, there are certain works that offer some insight. The most commonly investigated success variable included the attainment of the funding goal, while the most widely investigated determinant factors involved the campaign duration, the existence of the type of advertising made on social networks as well as linguistic features of the campaign description, such as blurb length and sentiment. Recent related works were listed and organised by dependent and independent variables in Table 2.1. Information regarding the independent variables: descriptive (abbreviated as "D"), financial (abbreviated as "F"), and linguistic (abbreviated as "L") features, the research methods, the number

of campaigns as data examples, accuracy (abbreviated as "Acc").

Table 2.1: Related work.

Papers	Class	Features	Method	Data	Acc
[23]	Pledged vs Goal	L	SVM + POS	-	71%
[10]	Backers, Pledged vs Goal	F, D	A.N.N.	-	94%
[14]	Pledged vs Goal	D, L	A.N.N.	216,136	91%
[8]	Pledged vs Goal	D, L	Logistic regression	698	-
[4]	Pledged	F, D, Social	SVM	16,042	76%
[1, 22]	Pledged, Goal	F, D, L	Random Forest + Ada-Boost	151,608	77%
[5]	Goal	F, D, Social	Random Forest	13,000	68%
[11]	Goal	F, D, Social	Logistic regression	4,121	77%
[21]	Goal	F, D	Logistic regression	9,962	-
[3]	Goal	D, L	SVM	26,543	-
[Kuo]	Goal	F, D, L	Logistic regression	132,853	76%

Etter et al. proposed a method in [4] for predicting the success of Kickstarter campaigns by using both direct information and social features. They used two sets of predictor features, the first one was a set of predictors that used a time series of money pledges to classify campaigns as probable success or failure and the second was a feature set that uses information gathered from tweets and Kickstarter's campaigns and backers graph. They showed that even though the feature sets that are based only on the amount of money pledged reach a high accuracy, combining them with features sets that use social media features enabled then to improve the performance significantly.

Mitra et al. tried exploring phrases that could predict success in [16]. They studied a data set of 45 thousand Kickstarter campaigns, analysing

phrases and other variables commonly present on different crowd-funding sites. They found that language used in the campaign description has surprising predictive levels, accounting for 58% of the variance around a campaign achieving its funding goal.

Ryoba et al. in [19] investigated campaign success contribution of various combinations of phased communication aspects like updates and comments. Machine learning algorithms were used to search and evaluate the best combination of phased communication aspects for predicting success using a Kickstarter campaigns dataset. Their study found that the number of updates, the sentiment of comments, readability of updates were the most important aspects in predicting campaign success.

Moutinho et al. in [17] provided a comprehensive review on factors influencing crowdfunding success by both focusing on campaign specific factors as well as creator/founder specific factors. Analysing a data set of projects from Kickstarter, they found that the campaign description, related images and videos as well as the creator having made campaigns before on Kickstarter influenced the campaign funding success.

Previous works have thus used various feature categories, though they only focused on a limited number of features (as the factors of success) per feature category. Despite the large number of previous works that used linguistic features as success factors, these features were extracted exclusively from the campaign description [3, 8, Kuo, 18, 20, 23], and not from the comments of each campaign which is a pretty big differentiating factor between this work and theirs.

Chapter 3

The data set

3.1 Data

Using the latest version (December of 2020) Webrobots Kickstarter data set [24] as a starting point, a new data set was created for the purposes of this work. The Webrobots data set consisted of one very large JavaScript Object Notation (JSON) file containing information about each campaign, each line itself consisting of a separate valid JSON object that had to be parsed and processed separately in a loop through the entire file.

Kickstarter has a lot of different data for each of the campaigns it hosts in their platform. A web scraper was developed that scraped a lot of different data from the website. In particular, the Webrobot's data set is critically missing textual information like the content of the comments of a campaign, or the text of the description of a campaign. This textual content was something that had not yet been used by other works to judge a campaign's success or failure so it became a focus for this work. Contrary to the usual web scraping of textual content and the difficulties of trying to use that content for machine learning purposes these comments contained no bad grammar, were of a fairly high spelling and grammar level, there was virtually no spam, malformed text, or incoherent writing as well. No bot accounts that spammed comments were found, by a process of manually looking through the comments, this can be easily explained by the fact that compared to Twitter, Reddit, and other social media platforms that are usually considered for doing machine learning on textual data, Kickstarter needs someone to actually pay and back a campaign for them to be able to

comment on it, it is not free, so the incentive to bot creators simply is not there.

After the scraping and preprocessing steps were finished, the data set consisted of 2,500 distinct examples (campaigns) and 35 features in total, including the feature label with the class being either successful or unsuccessful, depending on whether the campaign goal in United States Dollars (USD), was respectively reached, or not. The classes were very balanced, with 1230 examples labelled as successful and 1220 examples labelled as unsuccessful, so no up-sampling nor down-sampling techniques were needed at all. Data outliers were discarded after the feature extraction process described below, the limits were arbitrarily set manually and were based on each feature's characteristics to determine the outlier values, for example a 10 billion U.S. dollar fund goal was deemed to be an outlier, as well as a campaign who had zero funding, only campaigns with funding of over 1 dollar were considered in the work and included in the final data set of 2,500 examples.

3.2 Feature Extraction

Feature extraction or feature engineering is the process of creating "features", meaning usable representations of something, from a more abstract (higher level) thing. For the machine learning experiments to be successful and possible even, we need to work on the raw data we have and create features to feed into the models for training and testing. In the final data set we can sort the resulting features into three categories: features that were taken from the Webrobots data set as is, meaning without any processing, features that were created from the Webrobots data set, meaning some sort of processing was done on the raw data available from the Webrobots dataset to create features, and lastly, alongside the previous features, a custom web scraper [2] was developed in order to expand the existing data set with an array of textual information about the comments of each campaign, the number of updates and the number of backer reward tiers.

Table 3.1 contains the features used that were taken straight off of the Webrobots data set with only minimal processing mostly in formatting. Table 3.2 contains the resulting features from processing various data from the Webrobots data set. Finally Table 3.3 contains the features extracted

from the novel data produced from the Kickstarter web scraping code.

Name	Description	Type
Category	Campaign's root category	Nominal
Subcategory	Campaign's subcategory	Nominal
Country	Country the campaign was filed in from	Nominal
Staff Pick	Selected by the Kickstarter staff for promotion for being an "exceptional project" [12]	Binary
Backers	The number of unique people that backed the campaign	Discrete
Goal	The campaign goal (U.S. dollars)	Continuous
Pledged	The amount of funds pledged towards the campaign goal (U.S. dollars)	Continuous
Blurb Words	The number of words in campaign blurb (quick cover description)	Discrete
Blurb Length	The length of campaign blurb (characters)	Discrete
Successful	Whether the campaign reached its set goal or not	Binary

Table 3.1: The features used directly from the Webrobots data set without any processing or alteration thereof.

3.3 Statistical Information

Some statistical information was extracted about a select subset of features shown in Table 3.4 and Table 3.5 in order to provide a more detailed analysis of the proposed data set. Mean is the average value of all the examples, std is the standard deviation of the examples' values, min is the minimum value found in all of the examples, max is the respective maximum value, and finally, the 25th, 50th and 75th percentiles were included to provide a picture about the distribution density of each of the features' values.

To have a consistent value range in the monetary features, including the campaign goal, the average amount of pledge, the amount of funds, etc.

Name	Description	Type
Pledged vs Goal	The ratio of pledged funds to campaign goal	Continuous
Average Pledge	The mean pledge of campaign	Continuous
Campaign Days	The duration of campaign (days)	Discrete
Camp. Incubation	The time period between campaign creation and launch (days)	Discrete
Launch Month	The month the campaign was launched	Ordinal
Launch Weekday	The day of the week the campaign was launched	Ordinal

Table 3.2: The new features extracted from the Webrobots data set after processing.

we converted all of these values to dollars, since the Webrobots data set includes historic information about the conversion rate between USD and the currency the campaign was in at the time it was scraped, we can quite easily bring all the currencies to the same format and be very accurate about it at the same time. With the date information being in UNIX time format a converter function was developed to facilitate addition and subtraction of time stamps in order to get the campaign duration in days, and the campaign incubation period, the weekday of the campaign launch, or the month of the campaign launch. For example, to get the campaign duration, we subtracted the date that the campaign was launched from the date the campaign was created to get the length of the "incubation" period of the campaign (converted in number of days). Additionally, we subtracted the date the campaign ended from the date the campaign was launched, to get the duration the campaign had been active and converted it to the number of days.

Regarding the comments of each campaign, various features were extracted from them. For starters, a sentiment feature was extracted by use of the Valence Aware Dictionary and Sentiment Reasoner (VADER) algorithm, implemented in the Python language. VADER is a a dictionary and rule-based sentiment evaluation tool that was specifically created to assert sentiment of social media posts [9] so it fits really well with the comments

of Kickstarter. More specifically, VADER uses a combination of a sentiment dictionary (a list of words that have been labelled by experts for the sentiment they express) and various heuristics, such as exclamation marks, amplifying the sentiment either negatively or positively. VADER is very adaptable to machine learning algorithms due to its continuous value range from -1 to +1, values that we want anyways in a machine learning model, especially when we talk about deep learning and their neurons behaving a lot better with values in this range. For extracting the number of syllables of a word, the CMU Pronouncing Dictionary data set [15] was used. A wrapper function that makes use of the CMUDict data was created that counted the syllables of each word in a sentence. Also features arising from this previous data extracted from the comments were created, such as the average number of syllables per word in a comment.

Further features were extracted from the Kickstarter campaign pages, these include: the number of images that were included in the campaign's main description area in the landing page of the campaign. The length of the landing page description text of a campaign. The number of the Frequently Asked Questions (FAQ) entries in the campaign's FAQ section. The number of comments the creator made, as well as the average VADER sentiment score of those comments. The number of "new backers" of a campaign, new backers is the name Kickstarter gives to people who haven't backed any project before the one they are designated as "new backer" on. The ratio of new backers to all backers is also calculated as a feature. Flesch-Kincaid scores were calculated for every comment of every campaign and averaged out to provide a single number as a feature for each example in the data set. Flesch-Kincaid has two different types of scores, one is readability level, which means how easy something can be read, and the other is grade level complexity, which denotes at what educational level someone should be in order to properly understand a text. These scores were also calculated for each campaign's description text in addition to the comments.

Name	Description	Type
New Backers	The number of backers that have never backed a campaign previously	Discrete
New Backers Ratio	The ratio of new backers to all backers of a campaign	Continuous
Images	The number of images in a campaign's main page	Discrete
Tiers	The number of different backer reward tiers	Discrete
FAQs	The number of FAQ sections in a campaign	Discrete
Updates	The number of times the campaign was updated	Discrete
Comments	The number of comments posted	Discrete
Creator Comments	The number of comments posted by the creator of the campaign	Discrete
VADER Score	The mean VADER score of campaign comments	Continuous
VADER Cr. Score	The mean VADER score of campaign comments made by the creator	Continuous
Fl-Kn Readability	The average Flesch-Kincaid score of readability ease of the campaign's comments' text	Continuous
Fl-Kn Grade	The average Flesch-Kincaid score of grade level complexity of the campaign's comments	Continuous
Syllables	The mean number of syllables of campaign comments	Continuous
Words p. Comment	The mean number of words of campaign comments	Continuous
Comment to Word	The mean comment to word ratio of campaign comments	Continuous
Big Words	The mean big word ratio of campaign comments (more than 5 syllables)	Continuous
Desc. Text Length	The length of the description text of a campaign	Discrete
Desc. Fl-Kn Ease	The Flesch-Kincaid score of readability ease of the campaign's description text	Continuous
Desc. Fl-Kn Grade	The Flesch-Kincaid score of grade level complexity of the campaign's description text	Continuous

Table 3.3: The new features extracted from the new scraping data collected with the web scraper after processing.

	Backers	Goal	Pledged	Pledged/Goal	Mean Pledge
mean	80.6	28198	8404	1.49	70.53
std	318	316345	87414	6.13	256.17
min	1	0	0	0.00	0.00
25%	3	1111	62	0.01	17.50
50%	10	4000	566	1.00	38.80
75%	63	10000	4530	1.19	74.23
max	9100	1000000	3511060	140.44	1000.00

Table 3.4: Subset of the monetary statistics of the data set.

	Campaign Days	Campaign Incubation	Tiers	Updates
mean	33.39	31.10	7.26	5.03
std	12.57	56.02	5.77	8.16
min	1.00	0.00	0.00	0.00
25%	29.00	2.00	4.00	0.00
50%	30.00	9.00	6.00	1.00
75%	36.00	32.00	10.00	7.00
max	90.00	365.00	100.00	68.00

Table 3.5: Subset of the descriptive statistics of the data set.

	VADER	Cmnt. Words	Fl-Kn Ease	Fl-Kn Grade	Desc. Length
mean	0.19	9.44	62.26	8.35	2474.37
std	0.30	20.63	21.10	3.44	2718.99
min	-0.74	0.00	-301.79	-15.70	0.00
25%	0.00	0.00	60.00	7.94	780.00
50%	0.00	0.00	60.00	9.00	1619.00
75%	0.38	15.00	65.01	9.00	3323.00
max	1.00	525.00	206.84	55.60	24793.00

Table 3.6: Subset of the linguistic statistics of the data set.

Chapter 4

The web scraper

4.1 Environment Preparation

One big downside of the Web Robots Kickstarter dataset, is that it doesn't contain the comments for each of the campaigns, which means that if we want to do any kind of natural language processing for each campaign's comments, well, we are out of luck if we just use this dataset. To go further, we will have to collect them manually for every campaign with web scraping. Because Kickstarter does not offer any public API that we can use to get the information in a formatted and quick way this will have to be custom code that automatically operates a browser window, we will use Selenium for this, and scrapes the appropriate data that we want, for this task we will use the HTML scraping library BeautifulSoup.

We first have to install Python 3. For Linux we installed python using the distribution's package manager with the command

```
apt install python
```

After we have installed python, we run

```
pip3 --version
```

to make sure pip is installed and attached to the correct python version.

Then we install the required packages for the web scraping code to be able to run with

```
pip install selenium beautifulsoup4 pandas
```

We used a Python virtual environment in Ubuntu because we didn't want to clutter up the main Python installation with all these different packages.

Finally, we downloaded the latest Mozilla "geckodriver" and the latest Mozilla Firefox browser for the appropriate system. We placed the gecko-driver executable in a folder and then added the folder to the PATH environmental variable, depending on the operating system (for Ubuntu Linux we used *usr/local/bin* for an example).

4.2 The Layout

In Figure 4.2 we can see the structure of a typical Kickstarter campaign's comment section. There will always be a top-level comment from a backer and below there can be replies from the same backer, from the creator of the campaign, or from other collaborators to the campaign. This is the data that we want to collect in order to be able to process them into useful features, for use in a machine learning algorithm or neural networks.

Now that we know what we want to scoop up from the website, it's time to get into browser automation with Selenium and scraping with BeautifulSoup.

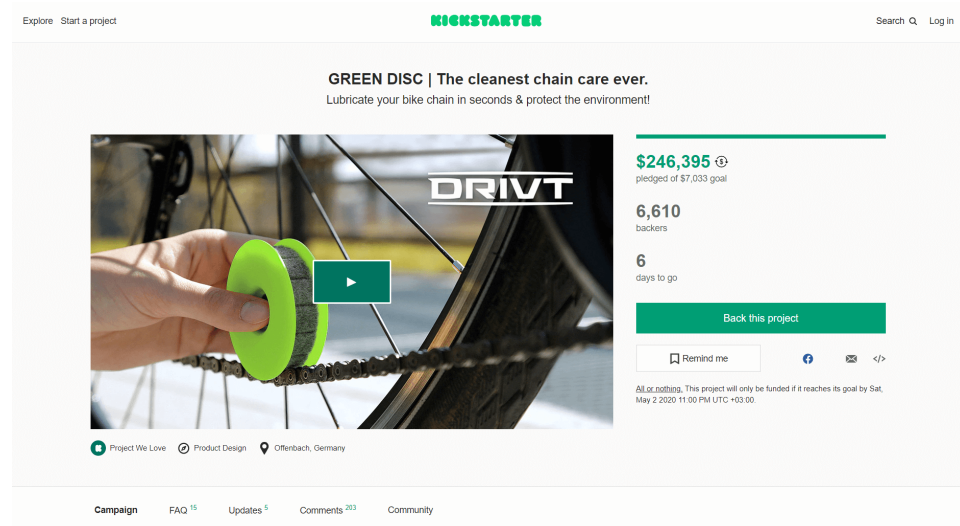


Figure 4.1: Here we can see the top banner section of a typical Kickstarter campaign.

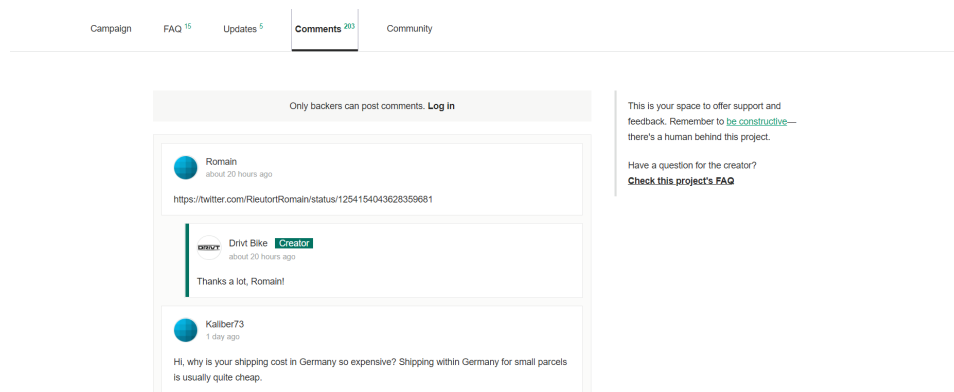


Figure 4.2: In order to get to the comments we need to click on the Comments tab on the bottom part of the banner.

4.3 Selenium

Now that we know what we want to scrape from the website, it's time to get into browser automation programming with Selenium and into HTML content scraping with BeautifulSoup.

Selenium is a browser automation software that interfaces with browser drivers such as `geckodriver` for Mozilla Firefox or `ChromeDriver` for Google Chrome. On its own Selenium doesn't do anything, it's just API bindings for external software that you can access through python code, so to continue you really need to install `geckodriver` and Firefox, which we will use for the code samples below.

We import the Selenium `WebDriver`, then we instantiate it into a browser variable, then specify a URL to it and open it with Mozilla Firefox by calling the `get` method of the `WebDriver`, this opens a Firefox browser window on the desktop environment and navigates automatically to the given URL. We then execute an `assert` statement in python, in order to be sure that we actually opened a Kickstarter web page, before continuing. This minimal example can be seen in Listing 4.1.

The WebRobots dataset we downloaded has the `.json` filename extension, but it's not a proper JSON object. As a whole it's not really a JSON file, as it's not a JSON array, nor a normal JSON object, instead it has thousands of JSON objects separated by newlines, which means we can't use normal JSON loading methods on it because they will not work. We need to separate the file into different strings for each of its lines and parse each of those

```
from selenium import webdriver

browser = webdriver.Firefox()
url =
↳ "https://www.kickstarter.com/projects/drivt/green-disc"
browser.get(url)
assert "Kickstarter" in browser.title # quit if error
```

Listing 4.1: A minimal working example, opening the browser on a specific web page, using Selenium.

strings into a JSON object separately, using the JSON module's `loads` (load string) method, then add all those objects into a list.

We will expand the previous example to be able to navigate programmatically inside the web page once it's been opened. But first we must work with the Web Robots dataset, in order to be able to add data we scrape to it, for that we will use pandas. The extended Selenium code can be seen in Listing 4.2.

4.4 Navigating

In order to do navigation and some other things like error handling and timeout checking with Selenium, we're importing some additional functions from the Selenium Python package as seen in Listing 4.3.

One of the problems that is immediately apparent is that for campaigns that have a lot of comments, we can only see the most recent 25 of them when we first open up the *Comments* tab. This is a problem because we cannot scrape data that doesn't exist in the web page yet. We will first have to load all the comments available in a campaign, and then scrape them from the HTML using BeautifulSoup.

There's one big loop in the code already, the one that goes through all the campaigns in the dataset one by one, we will add another loop inside that one, this nested loop will act as a way for expanding the comments, because we might need to click the "Load More" button at the bottom of the comments many times before all the comments have been loaded by the JavaScript. We add the following snippet of code inside the big campaign

```
from selenium import webdriver
import json
import pandas as pd

data = []
filename = "latest-webrobots-dataset.json"

# Opening the latest Web Robots json data set and reading
↪ line by line:
with open(filename, "r", encoding="utf8") as temp:
    for line in temp:
        data.append(json.loads(line)) # string to json
        ↪ loading function

# For every json object (campaign) in the list:
for campaign in data:
    campaign_url =
    ↪ campaign['data']['urls']['web']['project']
    .split('?')[0]
    browser.get(campaign_url)
    assert 'Kickstarter' in browser.title
```

Listing 4.2: The expanded Selenium code.

loop seen in Listing 4.4.

The loop in Listing 4.4 tries to find the "Load More" button at the bottom of the comment section and then clicks it. We give it a 6 second timeout window in order to find it, if it passes that it means that either the internet is very slow, or that the button has disappeared which means all the comments have been loaded. If instead we find the button, we click it, and the loop waits for 3 seconds, and starts looking for the button again. This 3 second delay is of course, arbitrary, and is about ten times longer than the time 25 new comments take to load and for the site to reach a steady state again.

TimeoutException is an exception that Selenium raises whenever the *WebDriverWait* takes more than the given timeout window to find the presence of an element in a *WebDriver* instance. Because JavaScript code and dynamic loading of content are not instantaneous, we need the delays in

```
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions
↪ as EC
from selenium.webdriver.common.by import By
from selenium.common.exceptions import TimeoutException
from selenium.common.exceptions import
↪ NoSuchElementException
```

Listing 4.3: The additional needed imports from Selenium for web scraping.

order to be sure we aren't missing content. And because the button that loads more comments disappears after all the comments have been loaded, we also need to handle that when dealing with the *TimeoutException*.

4.5 XPath

The huge strings seen in the above code are *XPath* specifiers that are used in the *findElementsByXpath* method, it's ideal to use XPath for these identical buttons that are repeated, because the way to find the correct XPath for the buttons is pretty simple, we basically need to find two of the same object, get their XPaths, and compare the two to see where they differ, and then remove that difference.

Now that we have loaded all the comments, we surely can just scrape it using BeautifulSoup. But not quite, we are not done yet, Kickstarter hides the long reply chains so we will need to also reveal all of those hidden comments, and we won't be able to click once for each comment chain, we will need to do a top to bottom sweep for all comment chain reveals, and then do it again until nothing is left in a recursive fashion. To add to that, comments made by people that have cancelled their pledge to the campaign, are also hidden, so we will need extra code for those ones too. The code to unfold all these hidden comments and comment chains is after the nested loop of comment loading but still in the big loop of campaigns.

Taking care of the cancelled pledge hidden comments is easy, no need for any kind of waiting, revealing the comment by clicking the *Show Comment* button is instantaneous since the content was already loaded by JavaScript

```

timeout_delay = 6 # seconds
# Loading Comments...
while True:
    try:
        myElem = WebDriverWait(browser,
                                ↪ timeout_delay).until(presence_of_element_located
                                ((By.XPATH, '/html/body/main/div/div/div[2]
                                /section[7]/div/div/div/div[2]/div/div/button')))
        myElem.click()
    except TimeoutException:
        print("end of comments reached") # button error
        ↪ (timeout or couldn't find it)
        break
    sleep(3)

```

Listing 4.4: The additional needed imports from Selenium for web scraping.

code, it was just being hidden. Tackling the hidden comment chain problem though we do need to wait for a few seconds to begin each iteration, because the button actually dynamically loads content from the server using AJAX calls, unlike the cancelled pledge comments which have already been loaded but simply hidden.

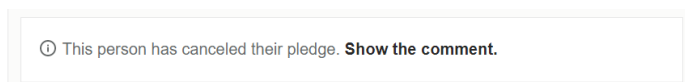


Figure 4.3: The comments left by people who in the end cancelled their pledge are hidden and need to be revealed using Selenium automation before scraping can begin.

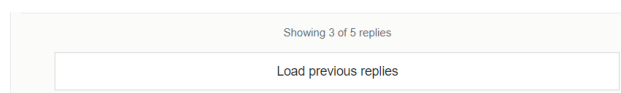


Figure 4.4: Every 25 comments this button needs to be pressed using JavaScript, then 25 more comments are loaded in using an AJAX call by Kickstarter.

4.6 Scraping with BeautifulSoup

BeautifulSoup is a Python library specialising in parsing HTML data into python data structures for easy access and processing. We will use it on the full HTML file after we have unfolded all the hidden comments using Selenium.

```
htmldata = BeautifulSoup(browser.page_source, 'html.parser')

# Comments
html_comments = htmldata.find_all('div', attrs={'class':
↪ 'border-box relative break-word border border-grey-400
↪ px3 pt3 pb2 o100p transition-all transition-delay-1000
↪ bg-white'})
comments = []
for comment in html_comments:
    comment_pieces = comment.find_all('p', attrs={'class':
↪ ['type-14 mb2', 'type-14 mb0']})
    stitched_comment = ''
    for text in comment_pieces:
        stitched_comment += text.text + ' '
    comments.append(stitched_comment)
```

Listing 4.5: Basic code for scraping the HTML comment data from Kickstarter.

First we load the HTML source code by using the BeautifulSoup constructor with *browser.pageSource* as the source code argument, and *html.parser* as an argument telling BeautifulSoup to use the default python HTML parser engine. Next, we use the *findAll* method to add all the comments in the *htmlComments* list. Using the inspection tool, we notice that all the comments, regardless of their nested positions, are all specified with the same class attribute, so that makes it easier for us to parse them all in one function call.

We create an empty array for the comments, then we notice that inside every one of the comments' bounding boxes that we used to load them into the *htmlComments* list, there are different paragraph elements that contain the fragmented text of the entire comment. So we have to call *findAll* again, this time on every *htmlComments* object, and finding every

paragraph of text. Finally we stitch the comments' text in one single string, remembering to put some spacing between the partial texts, we do this for every comment of the campaign append each of them into the comments list after the stitching, and then we're done.

Now that we are done with processing the comments into the comments list, we need to add the scraped data back to the original data structure that we loaded from the JSON. With this we are adding extra key and value pairs to the dictionary that holds the campaign data. Furthermore, we can selectively exclude comments based on them having been posted by the campaign's creator or a campaign collaborator, this is helpful because we want to work only with comments made by backers.

Now that all of the JSON objects are updated in memory, we will serialise them line by line to a new updated file of the exact same structure (albeit with additions) as the JSON we downloaded from WebRobots.

Chapter 5

Exploring the data

5.1 Exploratory Data Observations

Some features that we created in previous chapters were kept for exploratory data analysis purposes, but were then dropped in the actual training and evaluation of the machine learning models that we will be building in the next chapter. These included features that were related to the campaign being successful or not, like the amount of money pledged and the number of backers, rather than related to the properties of the project itself like the category, subcategory, goal amount, length of campaign, weekday of launch, etc.

Starting out, in Figure 5.1 we can very clearly see a huge divide in the average monetary goal a campaign creator sets for his campaign in if it's going to be successful or not. This can be quite easily explained with the fact that no matter the other factors, a lower goal will always be inherently easier to reach than a much higher one, no matter the quality of the campaign.

In Figure 5.2a it is shown that the successful campaigns have more than 3 times the FAQ number than failed ones. FAQs are an important part of a successful campaign because it gives insight into common questions a backer could have, before a new backer decides if he will support a campaign this could be a big help in that decision.

In Figure 5.2b it is shown that the successful campaigns have around 50% the reward tiers number than the failed ones. Reward tiers are yet another important part of a successful campaign because it gives backers more options on the monetary level they want to support a campaign, they

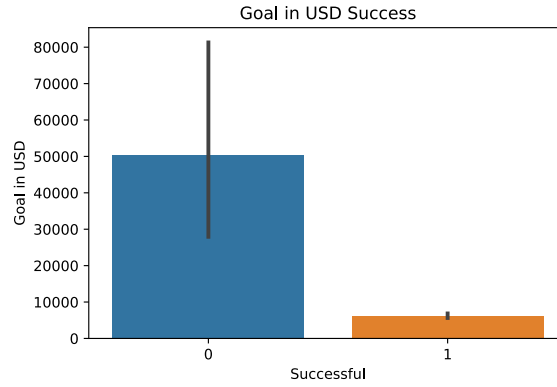
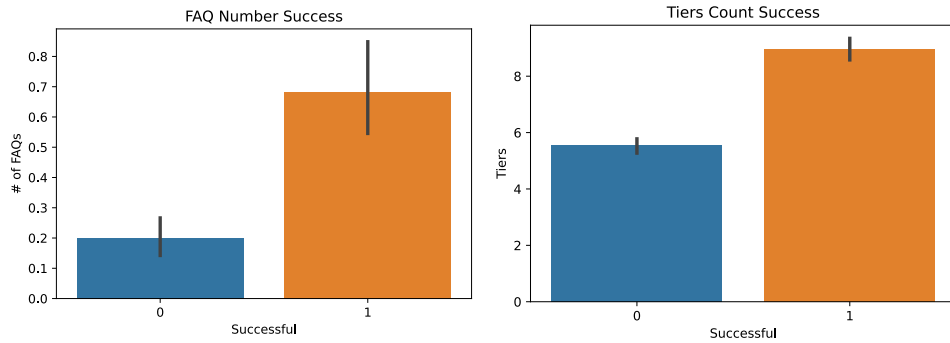


Figure 5.1: Average goal in USD (\$) for the two classes.



(a) Average reward tier count for each of the classes. (b) Average reward tier count for each of the classes.

Figure 5.2: Average FAQ and Tier numbers on each class.

give more flexibility to the reward structure and add opportunities for up-selling even in the campaign's funding.

An interesting fact comes up when we look at Figures 5.4 and 5.3, that is the point that there are measurable differences in success rates between campaigns launches that happen during certain days, or certain months, and all the others. For example, we can clearly see a higher rate of success during Tuesdays, and the least during Fridays, Saturdays, and Mondays, the difference is a lot too, around 10% higher during Tuesday than lets say during Saturday. The same can be seen if we take a look at months of launch, September holds a clear lead over every other month here, while December and January hold the two worst positions on when to launch a campaign with the difference being a noticeable percentage, over 15%.

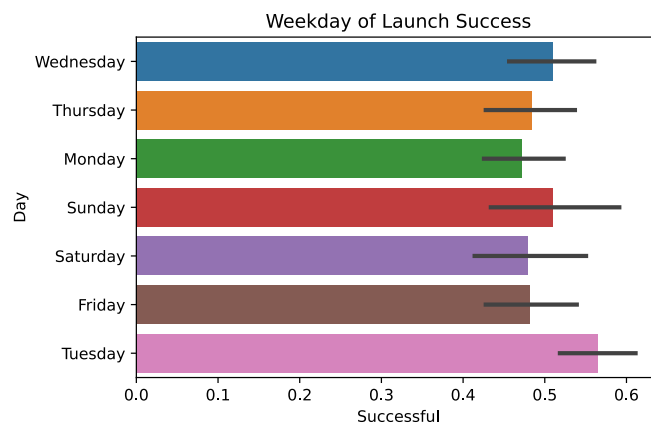


Figure 5.3: Success rate for campaigns launched during specific days of the week.

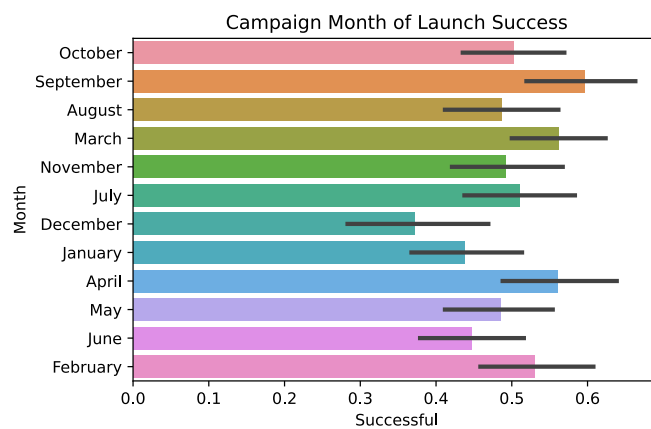
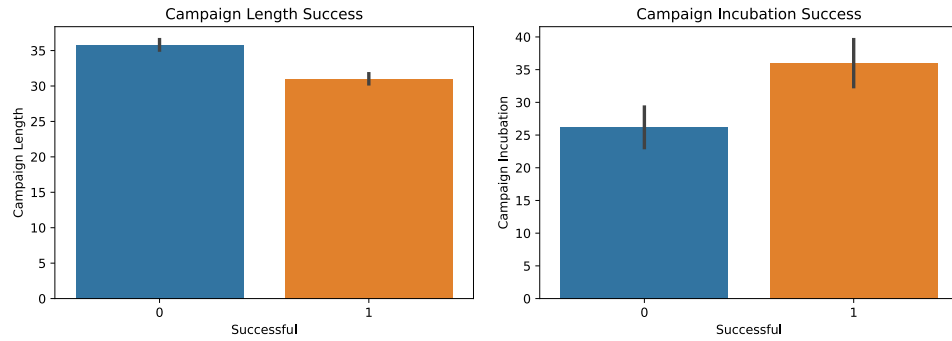
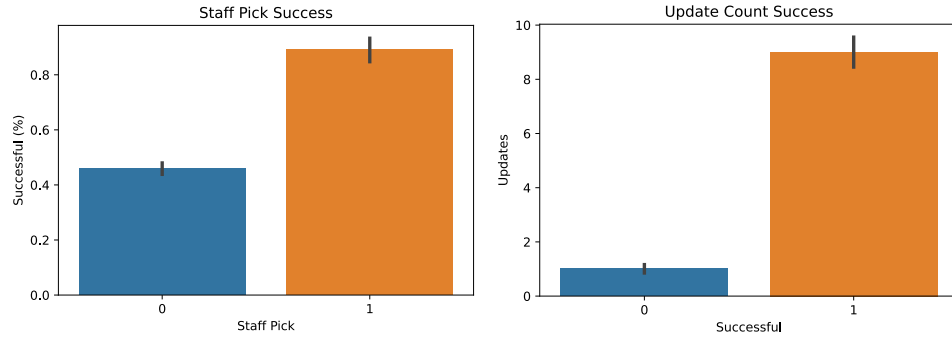


Figure 5.4: Success rate for campaigns launched during every month of the year.



(a) Average campaign lengths in days for each of the classes. (b) Average campaign incubation periods in days for each of the classes.

Figure 5.5: Campaign length and incubation periods stats.



(a) Staff pick rates for each of the two classes. (b) Average number of updates posted by a campaign's creator during its duration.

Figure 5.6: Staff picks and number of updates.

As far as periods of time goes, Figures 5.5a and 5.5b show that it's better to have a shorter campaign length (the period of time that the campaign is active on Kickstarter), as the successful campaigns have on average around 5 days less as a duration. Inversely however, the campaign incubation period (the period of time a campaign is up on the site, with all the information available for potential backers to explore, and link to, but without actually accepting backers actively) of successful campaigns is at least 10 days longer than the unsuccessful ones.

Moving on to "Staff Picked" campaigns, and number of updates. Figure 5.6a shows a pretty obvious thing, campaigns that have good presentation, reward structure, sensible goals, and a proven creator, get staff picked by Kickstarter and they get a premium position in the rankings, hence being

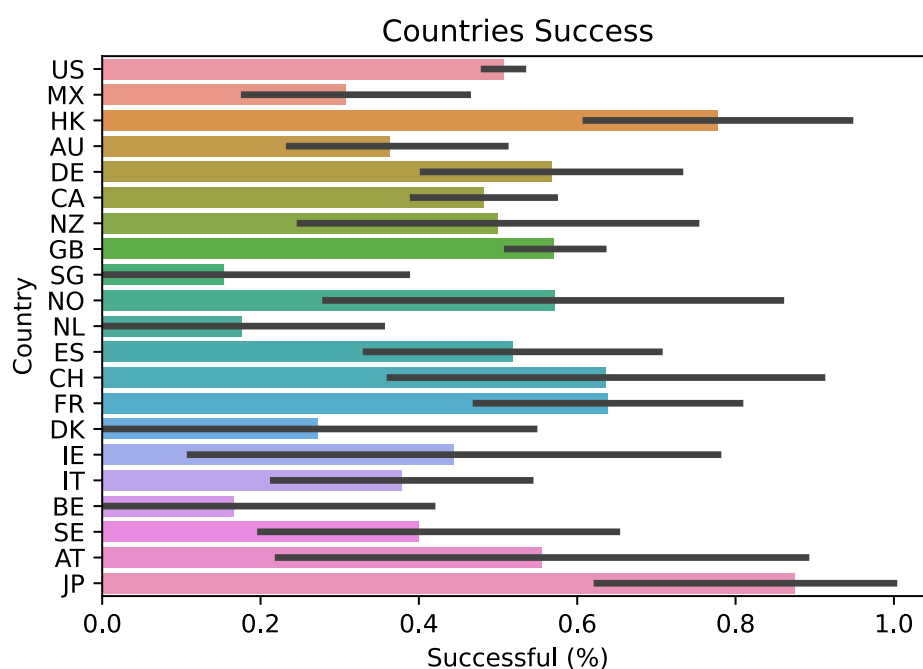


Figure 5.7: Success rate for the different countries Kickstarter operates in.

shown to more people, so they succeed more often. Specifically, campaigns that were staff picked, have a huge average success rate of being funded of 85%!

Figure 5.6b shows that campaigns that are successful post a lot more updates, 10 times as much in fact. Now this can go both ways, is it the fact that they are posting updates that's driving in more backers seeing as the campaign is active and the creator involved? Or is it the other way around, the campaign is successful so the creator is posting more updates? This statistic is kind of like what came first, the chicken or the egg.

Figure 5.7 shows that all countries are not created equal when it comes to getting a campaign funded through Kickstarter. There are some countries with severe disadvantages, mostly because Kickstarter and more generally crowd-funding are not well known enough in them so they attract way less traffic than let's say the US. Specifically, Japan and Hong Kong lead the way in percentage of campaigns funded, with Japan having 84% of campaigns funded, while Hong Kong trails that with 78%. It should be mentioned that there are error bars for a 95% confidence interval in these graphs. The more data we have on a country the shorter this bar interval becomes, so for a

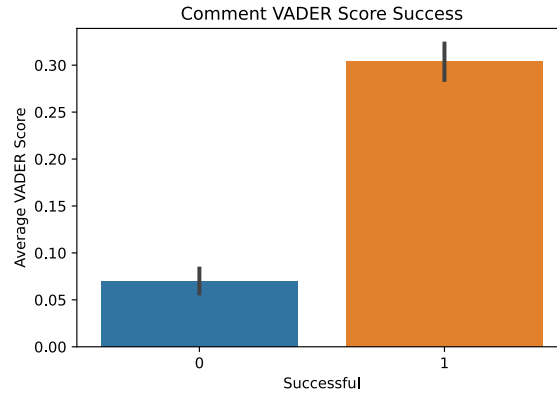


Figure 5.8: Average VADER comment score for both classes.

country like the U.K. we have a lot more statistics than a country like Hong Kong.

Yet another very interesting and expected result is the average VADER score for all the campaign comments as seen in Figure 5.8. We can see that the successful campaigns have on average a 4 times higher VADER sentiment score (higher means more positive), however even for the successful campaigns the overall VADER score is below 0, which means as a whole it is negative. This can of course be explained by the mere fact that there is always a bigger chance of someone displeased to write a comment than a happy backer leaving a nice comment.

Lastly let's take a look at the different Kickstarter categories and their success rates. Figure 5.9 shows all the different main categories (or parent categories) and their average successful funding rates, again, not all categories are created equal. Comics is by far and away the best category here, so to all aspiring comic book artists that's good news. Bad news for journalism however, since it is, alongside the crafts category, the worst category to launch a campaign in.

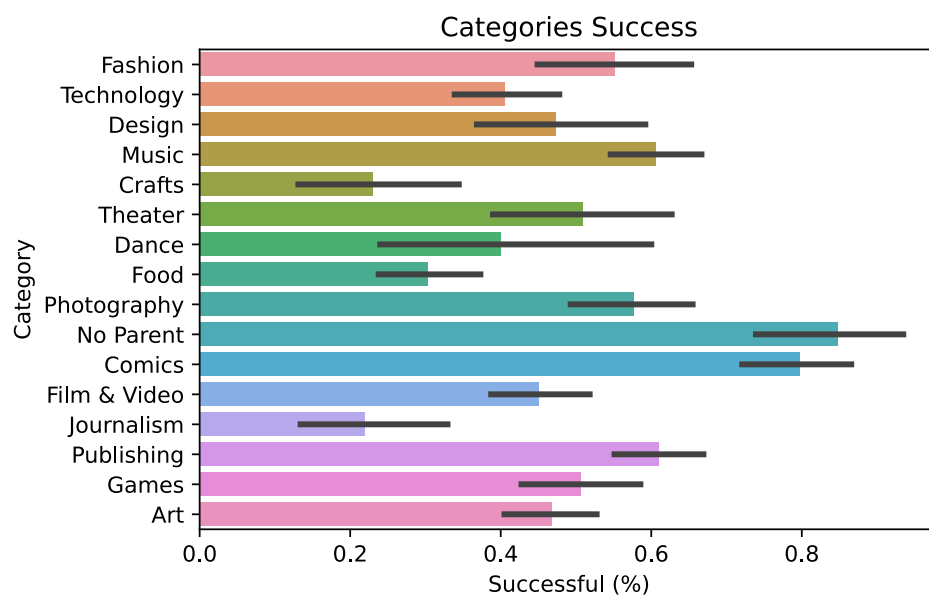


Figure 5.9: Success rate for the different main categories Kickstarter has.

Chapter 6

Experiments

6.1 Algorithms

In this section we will present the results of training different algorithms with our prepared data. We start with the most basic of the machine learning algorithms called Naive Bayes, it's called naive because it assumes no dependence between the different columns of our data. Then we will move on to more sophisticated algorithms like Decision Trees, SVMs, ensemble methods, and Deep Learning methods. We present the Precision and Recall metrics for each of the algorithms used below. Precision is the number of relevant predictions against retrieved instances, and Recall is the number of retrieved instances against all the relevant class instances.

Different sets of experiments were conducted with Naive Bayes, Decision Trees, Support Vector Machines (SVM), and ensemble methods such as AdaBoost with multiple Decision Trees. The machine learning algorithms were trained using the training set and then their performance was tested against the test set. We used a 5-fold cross validation [6] scheme with a leave-one-out approach for internally validating our models and afterwards we externally calculated the resulting performance metrics on the untouched test set. Cross-validating by leaving a pair out approach should also be considered as it's seen as more unbiased than other cross validation methods. For the Decision Tree model we trained a C4.5 Decision Tree classifier with maximum depth (the maximum depth of the graph of the generated tree structure) set to 10. There were also some performance enhancing features enabled, such as auto-pruning of leaves after a certain threshold [25]. For the

SVM algorithm we trained a simple linear kernel SVM, with a C parameter value of 1, C being a regularisation parameter that controls the balance between getting a low training error and a low testing error, that is the ability to generalise your classifier to the testing and entirely new, unseen data. We also employed an Ada-Boost ensemble classifier [7], using 100 C4.5 Decision Trees as the basis for the algorithm. Finally it should be mentioned that all of the machine learning experiments were conducted using Python version 3.8.6 on a Windows 10 x86-64 system and the latest versions of all the libraries were used.

6.2 Removing Features

Feature selection was performed on the data set by manual reviewing the effect features had on the results. More specifically, it was observed that the number of reward tiers feature of the financial category was extremely determinant for the prediction of a campaign's success and was as a result removed, so was the number of comments made to a campaign, we only kept the data coming from the comments, not the number of comments themselves. Additionally, the features "Backers Count", "New Backers Count", "Pledged vs Goal", "Pledged Amount" were also removed from the training/testing data set, since they are a direct pointer to success of the campaign and would completely skew the results of the experiments.

The feature columns "Weekday of Launch", "Month of Launch", "Country", "Category", and "Sub-category" were each replaced with one-hot encoded columns for each of their possible values, and 0 or 1 values for their existence or not in the respective columns that were created. One-Hot encoding exists because some learning algorithms cannot handle or "understand" categorical values, for example "Monday" or "Tuesday" have no meaning to a computer, we can remedy this by creating a new feature for each of the values that categorical value can take, for example a column named "Monday" will be created and for each example the value of the feature will be either 0 or 1 depending on if the value of the Weekday column was "Monday" or not.

Descriptive	Linguistic	Financial
Backers	Big Words	Tiers
New Backers	Desc. Length	Average Pledge
Campaign Days	Comment to Word	Pledged vs Goal
Campaign Incubation	Words per Comment	Staff Pick
Comments	Syllables	Goal
Country	VADER Score	Pledged
Month of Launch	VADER Cr. Score	
Category	Blurb Words	
Subcategory	Blurb Length	
Updates	F1-Kn Readability	
Weekday of Launch	F1-Kn Grade	
FAQs	F1-Kn Grade	

Table 6.1: Features per category.

	Precision	Recall	F1-Score
Success	76%	61%	68%
Fail	67%	81%	73%
Accuracy			71.0%

Table 6.2: Naive Bayes classifier results matrix.

	Precision	Recall	F1-Score
Success	91%	77%	83%
Fail	80%	92%	86%
Accuracy			85.0%

Table 6.3: C4.5 Decision Tree classifier results matrix.

	Precision	Recall	F1-Score
Success	92%	88%	90%
Fail	88%	91%	90%
Accuracy			90.6%

Table 6.4: Ada-Boost C4.5 Ensemble Decision Tree classifier results matrix.

6.3 Results

Onto the results: Table 6.5 presents the overall performance results of the algorithms. More specifically, Table 6.2 contains the results for the Naive Bayes algorithm. Even though Naive Bayes is not a very sophisticated machine learning algorithm, respectable accuracy scores were achieved, with

it reaching up to 71% accuracy, and posting numbers of 70% and 72% for precision and recall respectively. Noticeably improved results were achieved from the C4.5 Decision Tree algorithm, training of this algorithm was performed for varying levels of depth limits (depth of the resulting tree graph) and the classification error for each of the depth options was measured, noting that the "deeper" the Decision Tree is, the more overfitting exists to the training data, due to the inflexibility of the rules to the testing data. The final performance metrics occur for a depth limit of 9, as it achieved the best performance. We can see its performance in Table 6.3, good accuracy scores were achieved, with it reaching up to 85% accuracy, and posting numbers of 85% and 84% for precision and recall respectively. Meanwhile, the SVM algorithm reached a similar 86% accuracy score and more of the same precision with 84% and recall being at 85%.

For the Ada-Boost classifier, we used 100 C4.5 Decision Trees for training and applied pruning techniques in order to prevent overfitting and improve the overall performance. This algorithm was more robust to new data than a simple Decision Tree, since it outperformed the single C4.5 Decision Tree performance by a wide margin, having posted a 90.6% accuracy score, and 88% and 89% scores for precision and recall, respectively, as can be seen in Table 6.4.

Algorithm	CV 5-fold	Precision	Recall	F-1 Score
Naive Bayes	71%	70%	72%	71%
C4.5	87%	85%	84%	85%
SVM	86%	84%	85%	85%
Ada-Boost C4.5	89%	88%	89%	89%

Table 6.5: Performance of each algorithm. Internal 5-fold cross validation accuracy (CV 5-fold), and the final external validation metrics (Precision, Recall, F-1 Score).

Chapter 7

Deep Learning

7.1 Normalisation

7.1.1 Data Normalisation

In the data we have a lot of features with wildly different behaviours in their value distributions and value ranges, for example a feature may have a value range from 0 to 20, and another might have a value range from 10 to 25,000, this is very hard for a neural network to deal with since it is trying to combine those inputs through a series of linear combinations and non linear activation functions in order to try to make sense of their influence on the gradient of the loss function, and so the parameters associated with each input feature will also exist on different scales from each other, leading to weird loss function topologies and gradients.

By normalising all of the inputs to a standard scale common to all of the features in the data set, we are essentially allowing the neural network to more quickly learn the optimal parameters for each input feature and its correlation to the output. Also, it is generally useful to ensure that our inputs are in the range of $[-1, 1]$ in order to avoid weird mathematical errors associated with floating point number precision and arithmetic in systems like computers that use binary numbers to represent floating point numbers. Computers lose accuracy when performing mathematical operations on very big or very small numbers. Moreover, if the input features and target classes values range (in our case since it's a binary classification problem this is just 0 and 1) are on a completely different scale than the typical -1 to 1 range, the different parameters for the neural network (like the learning rate, batch

sizes) will likely be wrong for the type of data we are passing into it.

7.1.2 Batch Normalisation

In addition to the normalisation of the input feature values that happens right before we pass them to the first layer (the input layer) of our neural network, we will also employ something called *Batch Normalisation*. Batch normalisation normalises the output of a previous activation layer by subtracting the batch mean from each output value of a neuron and dividing by the batch standard deviation. Batch normalisation makes us able to use higher learning rates in our network parameters because it makes it so there is no activations going really high or really low respectively with the others in the specific layer we are normalising, with this effect it reduces overfitting of the model to the training data because it slightly regularises the learning of the layers to a more generalised version of the outputs of the previous layers, something similar to why drop-out is used on the hidden layers of artificial neural networks, to make the training "noisier" and to prevent overfitting.

7.2 Designing the A.N.N.

7.2.1 Control

We need to have a control network to act as a baseline for further results and analysis of different parameter studies for artificial neural networks. The control network will consist of an input layer the size of the input data first dimension, a hidden layer, and an output layer with a single neuron. The architecture model is a feed-forward neural network with 10% dropout, and batch normalisation. Figure 7.1 shows a scaled down model diagram of the control network. We can see the first layer having an input size of the vertical dimension of the data set, that is, the size of the feature vector. Second comes a 64 wide neuron layer which then outputs to the single neuron last layer which is the output. Same as with using a Naive Bayes algorithm as a baseline for comparing results to with normal Machine Learning experiments, such is the function for this network.

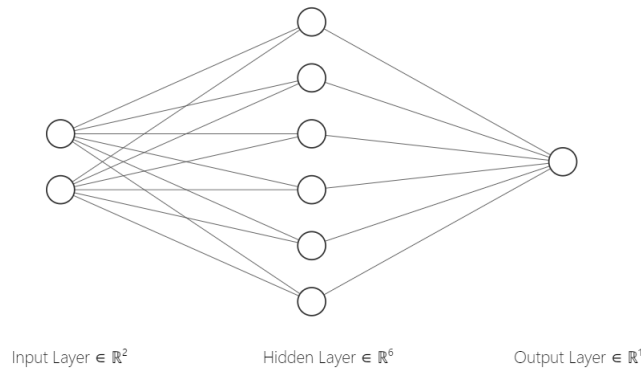


Figure 7.1: The architecture of the layers of the first neural network (control) using a 10:1 scale.

7.2.2 The Second Network

Let us now analyse the second artificial neural network we designed to feed our data into. Opting for 5 consecutive layers of neurons with feed-forward connections, i.e. the outputs of the neurons are directed only from their previous layer to the next, and do not create output feedback cycles. With the middle 3 of those layers being so-called "hidden" layers. Each layer of neurons is closely connected to its previous and next, each of its neurons is connected to each neuron in the two layers we mentioned. The input layer consists of 8 neurons which do not take any action except to pass their own corresponding characteristic to the neurons of the first hidden layer (hidden are all layers that are not input and output layers). The last layer is connected to the unique neuron of the network output layer that is used as the predicted value output of the network when in inference mode.

Listing 7.1 contains the code snippet that defines the above neural network in Facebook's Python Deep Learning Neural Network library called **PyTorch**, the code is generic, meaning we can add and remove layers as we want, we are also able to remove drop-out after each layer, use it selectively on layers, and also change its rate. In Figure 7.2 can see the first layer having an input size of 20, that is the size of the vertical dimension of the data set, the size of the feature vector. It then turns into a 256 neuron

wide hidden layer and then outputs its batch normalised values to the next hidden layer, and so on, until the last layer which is the output layer with a single neuron that has the prediction as its output value. We can also see the definition of the forward function which contains the activation function used (ReLU) and the drop-out probability of 10%. Also here we can see the batch normalisation happening after every layer except of course the last one.

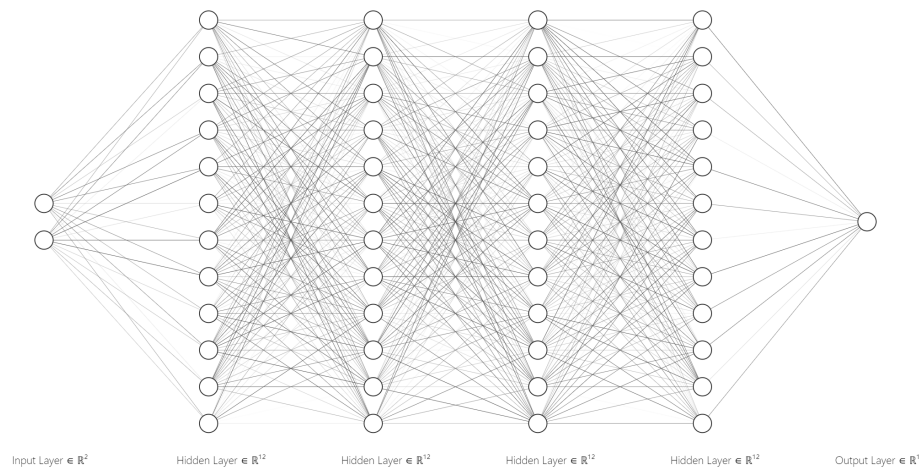


Figure 7.2: The architecture of the layers of the second neural network using a 10:1 scale.

7.2.3 Activation Function

Moving forward, with Figure 7.3 we see the graph of the *ReLU* activation function provided by PyTorch. Its output is the input value, if the input is positive, if the input is negative the function returns zero. It has been noted that the ReLU activation function is one of the most common in the field of neural networks for many reasons such as ease of calculation (the only arithmetic operation that is done is a comparison). This activation function is found in all but one of the network neurons, the output neuron, this is because PyTorch automatically uses a sigmoid function for the last output that we then convert to $[0, 1]$ with some easy code added to the end to do

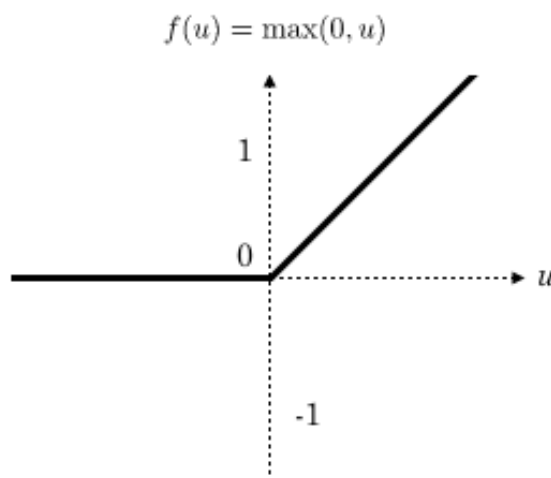


Figure 7.3: The ReLU activation function.

the conversion for us.

7.3 Training and Optimizers

What do we mean by the term *training* of a model? Neural networks are known for their almost magical ability to detect non-linear correlations between data, all this "magic" is done in the hidden layers of the network where neurons redefine their relationships (weights, polarizations) during training, and create various, fluid locality relationships. It is these relationships that demonstrate the ability to model networks that do computer vision, speech recognition, and non-linear functions amazingly well. The training algorithm we will use is the so-called *Adam*, a famous algorithm in the machine learning community, which comes with the PyTorch library, all we have to do is tell the library that Adam is the training algorithm we want for our model and pass it the learning rate that we want, for the first network we use a learning rate of 1%.

We will train our model using PyTorch's `fit` model library function, we have chosen to make 50 passes over the entire training data set as its training duration. Each pass over the data set during the training is called an *epoch*. Over the course of each epoch the training data (80% of the data) is shuffled before going through the network again, the validation data (20% of the overall data) is then used to score the performance of the network.

```
class binaryClassification(nn.Module):
    HIDDEN_LAYER_SIZE = 64
    DROPOUT = 0.1
    def __init__(self):
        super(binaryClassification, self).__init__()
        self.layer_1 = nn.Linear(len(df.columns),
            ↪ HIDDEN_LAYER_SIZE)
        self.layer_2 = nn.Linear(HIDDEN_LAYER_SIZE,
            ↪ HIDDEN_LAYER_SIZE)
        self.layer_out = nn.Linear(HIDDEN_LAYER_SIZE, 1)

        self.relu = nn.ReLU()
        self.dropout = nn.Dropout(p=DROPOUT)
        self.batchnorm1 = nn.BatchNorm1d(HIDDEN_LAYER_SIZE)
        self.batchnorm2 = nn.BatchNorm1d(HIDDEN_LAYER_SIZE)

    def forward(self, inputs):
        x = self.relu(self.layer_1(inputs))
        x = self.batchnorm1(x)
        x = self.relu(self.layer_2(x))
        x = self.batchnorm2(x)
        x = self.dropout(x)
        x = self.layer_out(x)

        return x
```

Listing 7.1: The code for defining the artificial neural network structure with the PyTorch library.

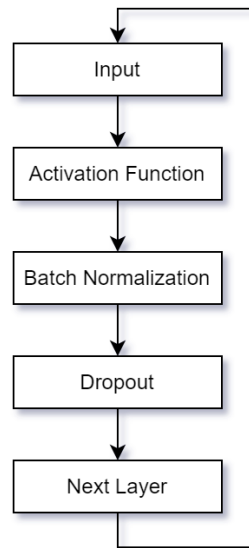


Figure 7.4: The flow of data and calculations inside every hidden layer of the neural network.

7.3.1 Parameter Studies

After examining the performance of the first and second networks we found it to be good, but not satisfactory compared to the traditional machine learning techniques we looked at previously. So we decided to design many different networks, each with a different number of layers, number of neurons per layer, different activation functions, etc. in order to find out how, and if, we will be able to surpass the performance of the first and second base models. This process is usually called a **parameter study**.

After hundreds of changes in every dimension and characteristic of an A.N.N. and its parameters we came up with the following tables, each one is a study of the influence a single parameter has on the performance of the network. The base neural network is the same as the second network design from the previous section in this chapter. We will now examine the differences in performance of all the different networks with the parameters tweaking as the differentiating factor between them.

7.4 Evaluation

In this section we will present the results of training different configurations of the second neural network architecture detailed in the previous section with our prepared data. On with the results, the following figures presents the overall performance results of the different configurations of the neural network parameters. More specifically, we changed the learning rate of the Adam optimiser algorithm, and played with the number of epochs the training went on for, and also took a look at the drop-out probabilities and hidden layer count of the network and how all of these affected performance.

In Figure 7.5 we can see a parameter study graph of the percentage of probability for a drop-out to occur in each and every neuron connection, on each hidden layer of a 3 hidden layer artificial neural network, with fixed parameters: the learning rate for Adam set at 0.001, 20 epochs, and batch size of 64. It is observed that a wide range of percentage probabilities for drop-out has a neutral effect on the performance of the network, and that drop-out rates of around 10% give a measurable benefit to accuracy. As expected, in high drop-out rates of over 50%, the network can still train on the data since the 20 epochs of training at 50% drop-out contains more or less the same information for the model as a 10 epoch 0% drop-out rate training cycle would. Further confirming the study number is the fact that with very high drop-out rates of over 90% we can see a sharp decline in performance as we would expect to happen.

In Figure 7.6 we can see a parameter study graph of the number of epochs of training. The 3 hidden layer neural network has the following fixed parameters: the learning rate for the Adam optimiser is set at 0.001, a batch size of 4, and a drop-out rate of 0%. It was observed that the best number of epochs for this particular model architecture, fixed parameters, and data set is around 10 to 20 epochs, anything above that and you are wasting time for marginal, if any, gains to performance. And if anything, on some cases might even lose performance as the model over fits to the training date, making it inflexible to adapt to the untouched test data hence lowering its accuracy numbers.

A rather bizarre finding is shown in Figure 7.7, we can see that adding extra hidden layers to the problem does not help the accuracy at all, in fact, when we add more than 6 hidden layers to our network the performance

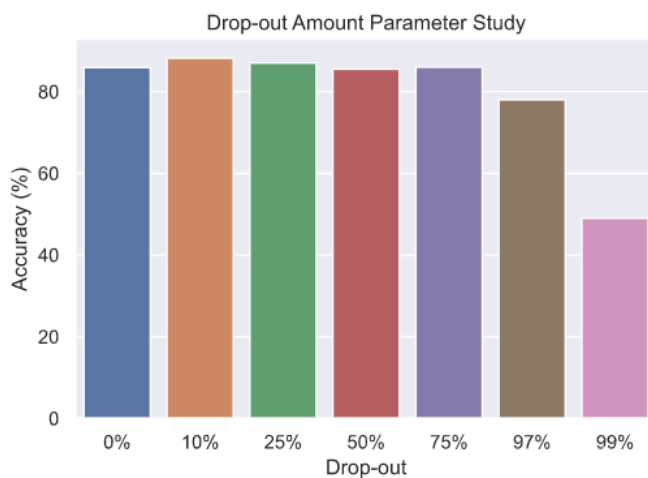


Figure 7.5: Parameter study of the percentage of probability for drop-out in each hidden layer output.

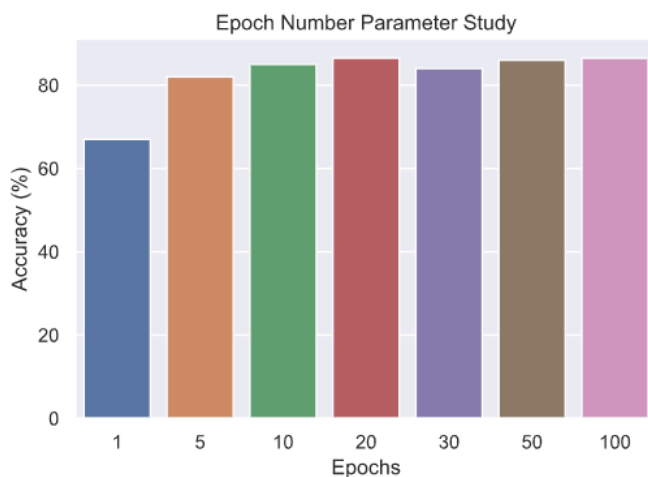


Figure 7.6: Parameter study of the number of training epochs.

starts to suffer considerably. This is most likely due to the fact that the problem we are dealing with is not predisposed to the deep learning concept, since it is already in a basic vector form. For a more in-depth look and discussion on the probable reasons for this behaviour refer to the *Deep Learning Methodology Analysis & Conclusions* section following this where we go in more detail about the usage of Deep Learning and its applications.

Lastly we take a look at the learning rate of the optimiser algorithm responsible for the back-propagation portion of the training, *Adam*. Figure

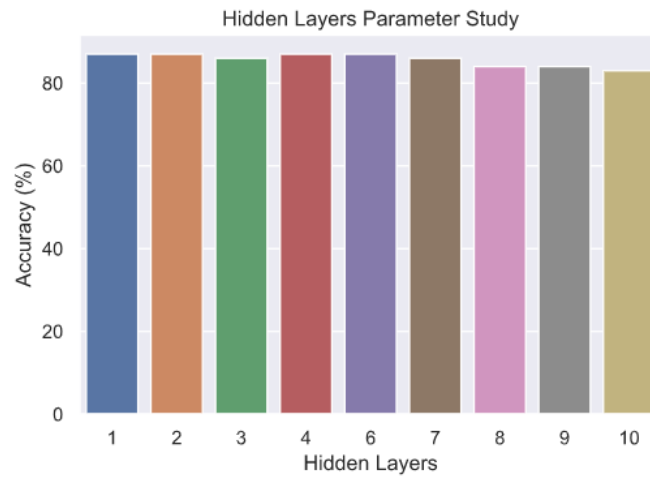


Figure 7.7: Parameter study of the number of hidden layers in the network.

7.8 shows that Adam does not really need tweaking, the default 1% learning rate can be safely left as is, since higher learning rates always have the danger of "overshooting" during the back-propagation process and ending up hurting performance between epochs. Also, high learning rates carry the risk of the algorithm not being able to reach the local minima of the loss function gradient because it just adjust weights too fast.

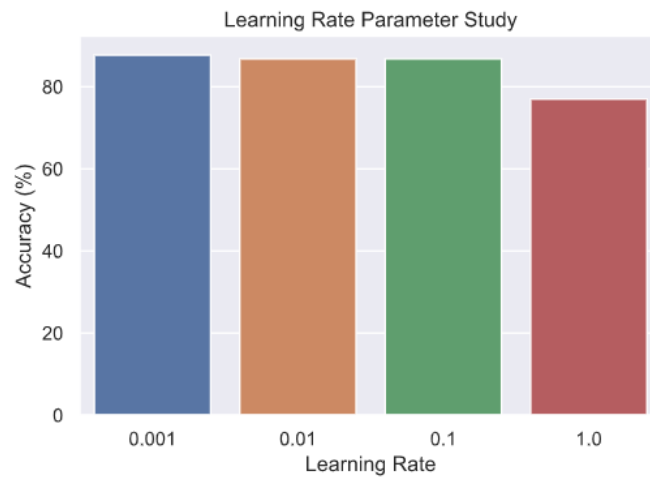


Figure 7.8: Parameter study of the learning rate of the optimiser algorithm "Adam".

7.4.1 Final Models

Taking into account all the previous findings of the parameter studies, a neural network was constructed with a 10% drop-out rate, 3 hidden layers, 256 neurons in each layer, with a 0.1% learning rate for the *Adam* optimiser, diagram can be seen in Figure 7.2. As a control network there is the 1 layer network with a 64 neuron layer and an output neuron with no drop-out, and the same 0.1% learning rate, diagram can be seen in Figure 7.1. Both of them using 20 epochs for training. These models will be compared to the traditional machine learning algorithms we saw previously.

Overall the first control neural network achieved 84.5% in accuracy, while the larger second tweaked network achieved 87.6% accuracy. This is disappointing because higher performance was expected from deep learning techniques, however it is explainable by simple deductions about the nature of the data being fed to the network.

7.4.2 Deep Learning Methodology Analysis & Conclusions

As we can see from the results in the previous section, it was very very difficult to get higher performance using a deep learning methodology with the same data we used on the previous chapter. There is a very clear reason for this result. Contrary to common knowledge and intuition, artificial neural networks are not inherently better at approximating functions and their results than other machine learning methods. The strength of deep learning lies in abstracting away complex things and data like images and not in some inherently better regression performance in the architecture of the model itself. A generally useful rule of thumb is that if the data is already in a tabular form like spreadsheets or data frames, then deep learning is probably not worth the time and complexity to get right. On the other hand for images, plain text, videos, graph networks, etc. deep learning may be the only way to do machine learning on that type of data.

This is because the main purpose of most deep learning models is to find a good vector representation of something that isn't intuitive for a human to do so, such as an image, a sentence in plain text, or an audio signal. The last layer of a deep learning model is usually just performing linear or logistic regression on the newly created/learned vector representation that now exists in the hidden layers of the network. However, if the data

is tabular, each row and each feature is essentially a good enough vector representation already, with most features unable to be processed further, so there is not a need for a deep learning model using artificial neural networks, at best they can achieve a marginally higher performance for hundreds or thousands of times the cost in time and processing power needs as a more traditional ensemble algorithm like Random Forests.

Using strong knowledge of the data and the domain of the data coupled with traditional machine learning algorithms may prove to be better than even the best artificial neural network models, by using a methodology called feature engineering. Trying to create new features out of already present ones, or from raw data, and squeezing out every last drop of statistical significance the data can provide to a learning algorithm. We did this already in the data processing and feature extraction chapters but there is a much deeper level that can be achieved with this technique, so that's left out as a future direction in this line of work.

Chapter 8

Conclusions

General Discussion

This thesis aimed at identifying success for crowd-funded campaigns by using machine learning techniques on 2,500 Kickstarter crowd-funded campaigns. To do so, we categorised, selected, and extracted data into three feature categories: descriptive features (i.e. category, campaign days), financial features (i.e. tiers, average pledge) and linguistic features (i.e. comment to word, VADER score). To the authors' best of knowledge, it is the first work that explores linguistic features extracted from campaign comment data as features in machine learning experiments on crowd-funding campaigns. Prior work has made linguistic analysis but focused on campaign descriptions only rather than using it on the campaign comments made by contributors to the campaign. We also identified problems in the usage of Deep Learning Neural Network techniques with this sort of "tabular" data that we were dealing with in this work, and laid out an argument on how and why this happened, and whether Deep Learning is even worth exploring as an option with similar data, or whether the authors of future work should just focus on better feature engineering instead of trying to get a neural network to adapt to their data.

The overall performance of the machine learning models throughout the experiments was fairly high. More specifically, it was observed that for the data set produced by the web scraper the best experiment results were obtained with the Ada-Boost ensemble classifier with accuracy up to 90.0%. Unfortunately weak performance was shown by all the feed-forward neural networks architectures, with this dataset that they simply could not quite

catch up to the ensemble models like the 100X C4.5 Ada-Boost model. While there are quite a few previous works having dabbled with the role of linguistic features as success factors for a crowd-funding campaign (e.g. syllables, VADER scores), they were just using data from the campaign description or title, or even the campaigns' short blurb sub-title. Instead, the data that was used to extract the linguistic features in this thesis was sourced from the comments made by backers of each campaign. This is a big separating factor of this work with previous research in the field of crowd-funding.

The web scraper is innovative in regards of it being able to scrape the entirety of a Kickstarter campaign's useful data in order to be used in machine learning experiments to create models that can do inference on different campaigns. Furthermore, the novel data set created through this effort has been made publicly available with a free use license for anyone to download, experiment, and expand upon. In addition to the data set, the entirety of the source code for the web scraper has also been made available in a public source code repository (GitHub) where anyone can download it and run it. Instructions regarding to set-up and usage of the code has been provided in the same public access repository. The goal is to allow others to collect and use a previously untapped source of textual information in hopes of producing great research going forward in the field of natural language processing and other related fields.

Future Work

In regards to future work in the field, more data needs to be collected, the nature of the web scraper and Kickstarter's I.P. banning of addresses that make too many requests is inherently limiting the amount of data a single researcher can produce at this point in time. While more publicly available datasets based on this are needed with more detailed features of natural language processing possibly. Analysis of a campaign's progression through its funding period as a time series is another way of studying this problem that can be considered in a future work. A custom web scraper must be built for this purpose that will regularly check and update data for crowd-funding campaigns within a certain time period (i.e. a week, a month or within the campaign duration). Finally, combining the use of machine learning methods along with statistical methods could enable defining the determinant role of each identified factor and improving prediction of the campaign's

success. A more interesting approach would be for a work that scrapes similar data points from other crowd-funding platforms and tries to compare the differences in user behaviour, commenting activity and sentiment, funding success statistics, et cetera.

Bibliography

- [1] Chung, J. and Lee, K. (2015). A long-term study of a crowdfunding platform: Predicting project success and fundraising amount. In *Proceedings of the 26th ACM Conference on Hypertext & Social Media*, pages 211–220.
- [2] Constantine Xipolitopoulos (2020). A scraper for kickstarter comments built with python, selenium, and beautifulsoup.
- [3] Desai, N., Gupta, R., and Truong, K. (2015). Plead or pitch? the role of language in kickstarter project success.
- [4] Etter, V., Grossglauser, M., and Thiran, P. (2013). Launch hard or go home! predicting the success of kickstarter campaigns. In *Proceedings of the first ACM conference on Online social networks*, pages 177–182.
- [5] Greenberg, M. D., Pardo, B., Hariharan, K., and Gerber, E. (2013). Crowdfunding support tools: predicting success & failure. In *CHI’13 Extended Abstracts on Human Factors in Computing Systems*, pages 1815–1820.
- [6] Guyon, I. (1997). A scaling law for the validation-set training-set size ratio. *AT&T Bell Laboratories*, 1(11).
- [7] Hastie, T., Rosset, S., Zhu, J., and Zou, H. (2009). Multi-class adaboost. *Statistics and its Interface*, 2(3):349–360.
- [8] Horvát, E.-Á., Wachs, J., Wang, R., and Hannák, A. (2018). The role of novelty in securing investors for equity crowdfunding campaigns. In *Sixth AAAI Conference on Human Computation and Crowdsourcing*.
- [9] Hutto, C. J. and Gilbert, E. (2014). Vader: A parsimonious rule-based model for sentiment analysis of social media text. In *Eighth international AAAI conference on weblogs and social media*.

- [10] Kamath, R. and Kamat, R. (2016). Supervised learning model for kickstarter campaigns with r mining. *International Journal of Information Technology, Modeling and Computing (IJITMC)*, 4(1):19–30.
- [11] Kaur, H. and Gera, J. (2017). Effect of social media connectivity on success of crowdfunding campaigns. *Procedia computer science*, 122:767–774.
- [12] Kickstarter (2015). How to get featured on kickstarter.
- [Kuo] Kuo, A. Predicting kickstarter success: An analysis of linguistic factors and model selection.
- [14] Lee, S., Lee, K., and Kim, H.-c. (2018). Content-based success prediction of crowdfunding campaigns: A deep learning approach. In *Companion of the 2018 ACM Conference on Computer Supported Cooperative Work and Social Computing*, pages 193–196.
- [15] Lenzo, K. et al. (2007). The cmu pronouncing dictionary. *Carnegie Mellon University*, 313.
- [16] Mitra, T. and Gilbert, E. (2014). The language that gets people to give: Phrases that predict success on kickstarter. In *Proceedings of the 17th ACM conference on Computer supported cooperative work & social computing*, pages 49–61.
- [17] Moutinho, N. and Leite, P. M. (2013). Critical success factors in crowdfunding: the case of kickstarter.
- [18] Parhankangas, A. and Renko, M. (2017). Linguistic style and crowdfunding success among social and commercial entrepreneurs. *Journal of Business Venturing*, 32(2):215–236.
- [19] Ryoba, M. J., Qu, S., Ji, Y., and Qu, D. (2020). The right time for crowd communication during campaigns for sustainable success of crowdfunding: Evidence from kickstarter platform. *Sustainability*, 12(18):7642.
- [20] Shafqat, W., Lee, S., Malik, S., and Kim, H.-c. (2016). The language of deceivers: Linguistic features of crowdfunding scams. In *Proceedings of the 25th International Conference Companion on World Wide Web*, pages 99–100.

- [21] Song, Y., Berger, R., Yosipof, A., and Barnes, B. R. (2019). Mining and investigating the factors influencing crowdfunding success. *Technological Forecasting and Social Change*, 148:119723.
- [22] Tran, T., Dontham, M. R., Chung, J., and Lee, K. (2016). How to succeed in crowdfunding: a long-term study in kickstarter. *arXiv preprint arXiv:1607.06839*.
- [23] Wang, W., Zhu, K., Wang, H., and Wu, Y.-C. J. (2017). The impact of sentiment orientations on successful crowdfunding campaigns through text analytics. *IET Software*, 11(5):229–238.
- [24] Webrobots (2020). Kickstarter datasets.
- [25] Weiss, S. M. and Indurkha, N. (1994). Decision tree pruning: biased or optimal? In *AAAI*, pages 626–632.

Appendix: Source Code

Web Scraper

```
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as
↳ EC
from selenium.webdriver.common.by import By
from selenium.common.exceptions import TimeoutException
from selenium.common.exceptions import NoSuchElementException
from time import sleep
from bs4 import BeautifulSoup
import re
import os
import pandas as pd
import json
from datetime import datetime

# determining starting and ending range to read from the 1.5GB
↳ file
with open('log.txt') as log:
    start = int(log.read()) + 1

assert type(start) == int
print(f'picking up from campaign #{start}')

end = start+1000
```

```

# Opening source json file and reading line by line:
data = []
filename = "Kickstarter_2020-12-17T03_20_12_051Z.json"

with open(filename, encoding='utf8') as f:
    for i, line in enumerate(f):
        if start < i < end:
            data.append(json.loads(line)) # parse line as json
            ↪ object
        if i > end: break
len(data)

browser = webdriver.Firefox()

updated_data = []

filename_write = "final_data.json"
with open(filename_write, "a") as updated_file:
    for k in range(len(data)):
        if k % 100 == 0 and k != 0:
            # every 100 (rejected included) campaigns, wait for a time
            ↪ to avoid getting IP banned
            sleep(300)
        try:
        try:
            assert data[k]['data']['state'] in ['canceled',
            ↪ 'successful', 'failed']
        except AssertionError:
            print(f"assertion error! {data[k]['data']['state']}")
            continue
        if data[k]['data']['state'] == "canceled":
            print("SKIPPED CANCELLED")
            continue
        if data[k]['data']['backers_count'] == 0:
            print("SKIPPED 0 BACKERS")

```

```

        continue
    campaign_url =
    ↪ data[k]['data']['urls']['web']['project'].split('?')[0]

    # Open web page
    browser.get(campaign_url)
    assert 'Kickstarter' in browser.title

    sleep(3)

    # add tier count
    try:
        # if new creator element exists in tier list
        browser.find_element_by_xpath('/html/body/main/div/div/div[2]
        .. /section[1]/div/div/div/div[2]/div/div[1]/button')
    try:
        data[k]['data']['tier_count'] =
        ↪ len(browser.find_elements_by_xpath('/html/body/main/div/div
        ..
        ↪ /div[2]/section[1]/div/div/div/div[2]/div/div[2]/div/ol/li/div[2]'))
    except NoSuchElementException:
        data[k]['data']['tier_count'] = 0
    except NoSuchElementException:
        # if not
        try:
            data[k]['data']['tier_count'] =
            ↪ len(browser.find_elements_by_xpath('/html/body/main/div
            ..
            ↪ /div/div[2]/section[1]/div/div/div/div[2]/div/div[1]/div/ol/li/div[2]'))
        except NoSuchElementException:
            data[k]['data']['tier_count'] = 0

    try: # add update count

```

```

data[k]['data']['update_count'] =
    ↪ int(browser.find_element_by_xpath('/html/body/main/div
    .. /div/div[2]/div[2]/div/div/div[1]/a[4]/span').text)
except NoSuchElementException:
    data[k]['data']['update_count'] = 0

try: # add number of images in description
data[k]['data']['image_count'] =
    ↪ len(browser.find_elements_by_xpath('/html/body/main/div/div
    .. /div[2]/section[1]/div/div/div/div[1]/div/div
    ..
    ↪ /section/div/div/div/div[1]/div/div/div[2]/div/div/figure/img'))
except NoSuchElementException:
    data[k]['data']['image_count'] = 0

try: # add description text
description_paragraphs = []
for p in
    ↪ browser.find_elements_by_xpath('/html/body/main/div/div
    .. /div[2]/section[1]/div/div/div/div[1]/div/div
    .. /section/div/div/div/div[1]/div/div/div[2]/div/p'):
    description_paragraphs.append(p.text)
data[k]['data']['description_text'] = '
    ↪ '.join(description_paragraphs)
#print(data[k]['data']['description_text'])
except NoSuchElementException:
    print("error on description text")
    data[k]['data']['description_text'] = ""

try: # add faq count
data[k]['data']['faq_count'] =
    ↪ int(browser.find_element_by_xpath('/html/body/main/div
    .. /div/div[2]/div[2]/div/div/div[1]/a[3]/span').text)
except NoSuchElementException:
    data[k]['data']['faq_count'] = 0

```

```

# Change to community section
try:
    browser.find_element_by_id('community-emoji').click()

    sleep(1)

try: # add new backers count
    data[k]['data']['new_backers_count'] =
        ↪ int(browser.find_element_by_xpath('/html/body/main/div
        .. /div/div[2]/section[8]/div/div[3]/div/div/div
        .. /div/div[1]/div[1]').text)
except NoSuchElementException:
    data[k]['data']['new_backers_count'] = 0

try: # add returning backers count
    data[k]['data']['returning_backers_count'] =
        ↪ int(browser.find_element_by_xpath('/html/body/main
        .. /div/div/div[2]/section[8]/div/div[3]/div/div
        .. /div/div/div[2]/div[1]').text)
except NoSuchElementException:
    data[k]['data']['returning_backers_count'] = 0
except NoSuchElementException:
    data[k]['data']['returning_backers_count'] = 0
    data[k]['data']['new_backers_count'] = 0

# Change to comment section
browser.find_element_by_id('comments-emoji').click()

timeout_delay = 6 # s
while True:
    # the big loop loading all the comments from javascript

```

```

try:
    myElem = WebDriverWait(browser,
        ↪ timeout_delay).until(EC.presence_of_element_located((By.XPATH,
        ↪ '/html/body/main/div/div/div[2]/section[7]/div
        .. /div/div/div[2]/div/div/button'))))
    myElem.click()
except TimeoutException:
    print("end of comments reached") # button error
    ↪ (timeout or couldn't find it)
    break
sleep(3)

cancelled_pledge_button =
    ↪ browser.find_elements_by_xpath('/html/body/main/div/div
    ..
    ↪ /div[2]/section[7]/div/div/div/div[2]/div/ul/li/div[1]/p/a')
for button in cancelled_pledge_button:
    button.click()

# run as long as you find no more buttons ( if the chain is
    ↪ long, it might need to push the button many times to
    ↪ reveal all the comment replies )
while True:
    show_more_buttons=
        ↪ browser.find_elements_by_xpath('/html/body/main/div
        .. /div/div[2]/section[7]/div
        .. /div/div/div[2]/div/ul/li/div[2]/div/button')
    if len(show_more_buttons) == 0: break
    for button in show_more_buttons:
        button.click()
    sleep(3)

# how to select which Xpath array [ ] to remove: find similar
    ↪ elements and see where they differ, and remove that.

print("Selenium engine stopped, beautifulsoup HTML comment
    ↪ scraping started..")

```

```

sleep(1)

# Beautiful Soup
htmldata = BeautifulSoup(browser.page_source, 'html.parser')

# comments
htmlcomments = htmldata.find_all('div', attrs={'class':
↪ 'border-box relative break-word border border-grey-400 px3
↪ pt3 pb2 o100p transition-all transition-delay-1000
↪ bg-white'})
comments = []
creator_comments = []
for i in range(len(htmlcomments)):
    creator_badge = htmlcomments[i].find_all('span',
↪ attrs={'class': 'bg-ksr-green-700 white px1 type-14 mr1'})
    collaborator_badge = htmlcomments[i].find_all('span',
↪ attrs={'class': 'bg-ksr-green-700 o50p white o0p px1
↪ type-14 mr1'})
    comment_pieces = htmlcomments[i].find_all('p', attrs={'class':
↪ ['type-14 mb2', 'type-14 mb0']})
    full_comment = ''

    for text in comment_pieces:
        full_comment += text.text + ' '

    if (len(creator_badge) > 0 or len(collaborator_badge) > 0):
        creator_comments.append(full_comment)
    else:
        comments.append(full_comment)

#print("COMMENT: " + full_comment)

# adding comment data to json
data[k]['data']['comments'] = comments
data[k]['data']['comment_count'] = len(comments)

```

```

# adding creator comments to json
data[k]['data']['creator_comments'] = creator_comments
data[k]['data']['creator_comment_count'] =
↳ len(creator_comments)

# debug prints
#print(f"Tier count: {data[k]['data']['tier_count']}")
#print(f"Update count: {data[k]['data']['update_count']}")
#print(f"Comment count: {data[k]['data']['comment_count']}")
#print(f"Comment count:
↳ {data[k]['data']['creator_comment_count']}")
#print(f"Image count {data[k]['data']['image_count']}")
#print(f"FAQ count {data[k]['data']['faq_count']}")
#print(f"new backers count
↳ {data[k]['data']['new_backers_count']}")
#print(f"returning backers count
↳ {data[k]['data']['returning_backers_count']}")

sleep(5)
updated_data.append(data[k])
except:
    continue

# finished with this campaign, write to log so we can pick up
↳ again later
with open('log.txt', mode='w') as log:
    log.seek(0)
    log.truncate(0)
    log.write(str(start+k))

updated_file.write(json.dumps(data[k])+'\n')

print(f"writing campaign ({start}+{k}=) #{start+k} to file,
↳ logging new campaign index")

```

Feature Extraction

```

import pandas as pd
import json
import os
from datetime import datetime
from vaderSentiment.vaderSentiment import
↳ SentimentIntensityAnalyzer
import cmudict
import textstat

data = []
filename = 'final_data.json'
with open(filename, "r") as f:
    for line in f:
        data.append(json.loads(line))

comments_sum = 0
num_successful = 0
num_failed = 0
remove_campaigns = []
for i, campaign in enumerate(data):
    comments_sum += campaign['data']['comment_count']
    if campaign['data']['state'] == 'failed':
        num_failed += 1
    if campaign['data']['state'] == 'successful':
        num_successful += 1

# delete live/cancelled campaigns, incubation periods over one
↳ year, campaign goal over 100 million dollars
if (campaign['data']['state'] != 'successful' and
↳ campaign['data']['state'] != 'failed') or
↳ (campaign['data']['goal'] > 20000000) or
↳ (int((float(data[i]['data']['launched_at'])-float(data[i]['data']['created_at'])))/(6
↳ > 365):
    remove_campaigns.append(i)

```

```

# delete in reverse order as to not mess with the indexing
↪ while deleting list objects
for i in reversed(remove_campaigns):
    #print(i)
    del data[i]
print("Average number of comments in campaigns:",
    ↪ round(comments_sum/len(data), 1))
print("Number of successful campaigns:", num_successful)
print("Number of failed campaigns:", num_failed)

# feature extraction
d = cmudict.dict()
def nsyl(word):
    return [len(list(y for y in x if y[-1].isdigit())) for x in
    ↪ d[word.lower()]]

vader = SentimentIntensityAnalyzer()
cleaned_data = []
for i, campaign in enumerate(data):
    #print(i)
    name = data[i]['data']['category']['name'] # (string)
    country = data[i]['data']['country'] # (string)
    try:
        parent_name = data[i]['data']['category']['parent_name'] #
        ↪ (string)
    except KeyError:
        parent_name = "No Parent"

    staff_pick = 0 if data[i]['data']['staff_pick'] == False else
    ↪ 1 # was bool (0 1 (int))
    successful = 0 if data[i]['data']['state'] == 'failed' else 1
    ↪ # was string (0 1 (int))

    backers_count = data[i]['data']['backers_count'] # (int)

```

```

usd_goal = int(float(data[i]['data']['goal']) *
    ↪ float(data[i]['data']['static_usd_rate'])) # (int)
usd_pledged = int(float(data[i]['data']['pledged']) *
    ↪ float(data[i]['data']['static_usd_rate'])) # (int) TODO:
    ↪ relation between pledges and goal
try:
average_pledge = usd_pledged / backers_count # average pledged
    ↪ amount per backer (int)
pledged_vs_goal = usd_pledged / usd_goal
except ZeroDivisionError:
average_pledge = 0
pledged_vs_goal = 0

blurb_words = len(data[i]['data']['blurb'].split(' ')) # words
    ↪ of blurb (int)
blurb_length = len(data[0]['data']['blurb']) -
    ↪ len(data[i]['data']['blurb'].split(' ')) # length of
    ↪ blurb's words (int)

campaign_days =
    ↪ int((float(data[i]['data']['deadline'])-float(data[i]['data']['launched_at']))/(60*60*24))
    ↪ # campaign length in days (int)
campaign_incubation =
    ↪ int((float(data[i]['data']['launched_at'])-float(data[i]['data']['created_at']))/(60*60*24))
    ↪ # days from launch to (int)

month_of_launch =
    ↪ datetime.utcfromtimestamp(int(data[i]['data']['launched_at'])).strftime('%B')
    ↪ # Month of campaign launch (string)
weekday_of_launch =
    ↪ datetime.utcfromtimestamp(int(data[i]['data']['launched_at'])).strftime('%A')
    ↪ # Weekday of campaign launch (string)

tier_count = data[i]['data']['tier_count'] # (int)
update_count = data[i]['data']['update_count'] # (int)
comment_count = data[i]['data']['comment_count'] # (string)

```

```

image_count = data[i]['data']['image_count'] # (int)
faq_count = data[i]['data']['faq_count'] # (int)
creator_comment_count =
    ↪ data[i]['data']['creator_comment_count'] # (int)
new_backers_count = data[i]['data']['new_backers_count'] #
    ↪ (int)
new_backers_ratio = new_backers_count / float(backers_count) #
    ↪ (float)

if len(data[i]['data']['creator_comments']) > 0:
    creator_vader_scores = 0
    for comment in data[i]['data']['creator_comments']:
        creator_vader_scores +=
            ↪ vader.polarity_scores(comment)['compound']
    avg_creator_vader_score =
        ↪ creator_vader_scores/len(data[i]['data']['creator_comments'])
else:
    avg_creator_vader_score = 0

description_flesch_kincaid_grade =
    ↪ textstat.flesch_kincaid_grade(data[i]['data']['description_text'])
description_flesch_kincaid_ease =
    ↪ textstat.flesch_reading_ease(data[i]['data']['description_text'])
description_length = len(data[i]['data']['description_text'])

# VADER sentiment score (includes CAPITALIZATION, slang terms,
    ↪ emoticons :D, emojis, !!!s and ???s)
if len(data[i]['data']['comments']) > 0:
    vader_scores = 0
    for comment in data[i]['data']['comments']:
        vader_scores += vader.polarity_scores(comment)['compound']
    avg_vader_score =
        ↪ vader_scores/len(data[i]['data']['comments'])
else: # to avoid division by zero error
    avg_vader_score = 0

```

```

# sentences ,syllables, and words
comment_syllables = 0
comment_words = 0
comment_big_words = 0
flesch_sum = 0
flesch_grade_sum = 0
for comment in data[i]['data']['comments']:
    syllable_count = 0
    word_count = 0
    big_words = 0
    sentence_number = comment.count('. ') + comment.count('? ') +
    ↪ comment.count('! ')
    if sentence_number == 0: sentence_number = 1
    for word in comment.strip().split(' '):
        word_count += 1
        cw = ''.join(filter(str.isalpha, word))
        sl = nsyl(cw)
        try:
            syllable_count += sl[0]
            if sl[0] >= 4: big_words += 1 # big words
        except:
            pass
    comment_syllables += syllable_count
    comment_words += word_count
    comment_big_words += big_words
    flesch_sum += textstat.flesch_reading_ease(comment)
    flesch_grade_sum += textstat.flesch_kincaid_grade(comment)
    try:
        avg_comment_words =
        ↪ comment_words/len(data[i]['data']['comments']) # average
        ↪ word count per comment
    avg_syllable_count =
    ↪ comment_syllables/len(data[i]['data']['comments'])
    ↪ #average syllable count per comment

```

```

avg_comment_syllable_to_word_ratio =
    ↪ comment_syllables/comment_words # syllables to words ratio
avg_big_word_ratio = comment_big_words / comment_words # big
    ↪ words to words ratio
flesch_kincaid_ease = flesch_sum /
    ↪ len(data[i]['data']['comments']) # average flesch-kincaid
    ↪ reading ease score of all comments
flesch_kincaid_grade = flesch_grade_sum /
    ↪ len(data[i]['data']['comments']) # average flesch-kincaid
    ↪ grade level of all comments

except ZeroDivisionError:
    avg_comment_words = 0
    avg_syllable_count = 0
    avg_comment_syllable_to_word_ratio = 0
    avg_big_word_ratio = 0
    flesch_kincaid_ease = 60
    flesch_kincaid_grade = 9

cleaned_data.append([
    name,
    parent_name,
    country, staff_pick,
    backers_count,
    usd_goal,
    usd_pledged,
    round(pledged_vs_goal,3),
    round(average_pledge,3),
    blurb_words,
    blurb_length,
    campaign_days,
    campaign_incubation,
    month_of_launch,
    weekday_of_launch,
    tier_count,
    update_count,

```

```

        comment_count,
        round(avg_vader_score, 3),
        round(avg_comment_words, 3),
        round(avg_syllable_count, 3),
        round(avg_comment_syllable_to_word_ratio, 3),
        round(avg_big_word_ratio, 3),
        flesch_kincaid_ease,
        flesch_kincaid_grade,
        image_count,
        faq_count,
        creator_comment_count,
        new_backers_count,
        new_backers_ratio,
        avg_creator_vader_score,
        description_flesch_kincaid_grade,
        description_flesch_kincaid_ease,
        description_length,
        successful
    ])
print(len(data), len(cleaned_data)) # Should be the same

col_names = [
    'name',
    'parent_name',
    'country',
    'staff_pick',
    'backers_count',
    'usd_goal',
    'usd_pledged',
    'pledged_vs_goal',
    'average_pledge',
    'blurb_words',
    'blurb_length',
    'campaign_days',
    'campaign_incubation',
    'month_of_launch',

```

```
'weekday_of_launch',  
'tier_count',  
'update_count',  
'comment_count',  
'avg_vader_score',  
'avg_comment_words',  
'avg_syllable_count',  
'avg_comment_to_word_ratio',  
'avg_big_word_ratio',  
'flesch_kincaid_ease',  
'flesch_kincaid_grade',  
'image_count',  
'faq_count',  
'creator_comment_count',  
'new_backers_count',  
'new_backers_ratio',  
'avg_creator_vader_score',  
'description_flesch_kincaid_grade',  
'description_flesch_kincaid_ease',  
'description_length',  
'successful']
```

```
df = pd.DataFrame(cleaned_data)  
df.columns = col_names  
df.describe().round(1)  
  
df.to_csv("machine_learning_final_dataset.csv")
```

Model Creation

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sb
from sklearn import tree
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import AdaBoostClassifier
from sklearn.svm import SVC
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report

df = pd.read_csv("machine_learning_final_dataset.csv")

#print(df['successful'].value_counts(), "\nBalanced classes!")
y = df.pop('successful').values # Popping the class out of the
↪ data

# One Hot Encoding of categorical data columns
df = pd.get_dummies(df, columns=['weekday_of_launch',
↪ 'month_of_launch', 'country', 'parent_name', 'name'])
#del df['weekday_of_launch']
#del df['month_of_launch']
#del df['country']
#del df['parent_name']
#del df['name']

del df['new_backers_count']
del df['backers_count']
del df['pledged_vs_goal']
del df['average_pledge']
del df['usd_pledged']
del df['comment_count']
del df['update_count']
```

```
# Splitting data to training and test sets
X_train, X_test, y_train, y_test = train_test_split(df, y,
    ↪ test_size=0.2, random_state=0) # X means features
    ↪ (dataframe), Y means class(list).

#Gaussian Bayes
gnb = GaussianNB()
gnb.fit(X_train, y_train)

# Cross-validation scoring
print(f"5-fold C-V score: {cross_val_score(gnb, X_train,
    ↪ y_train, cv=5, scoring='accuracy').mean().round(3)*100}%")

# Scoring against untouched test data
y_true = y_test
y_pred = gnb.predict(X_test)
print(classification_report(y_true, y_pred,
    ↪ target_names=['Success', 'Failure']))

gnb.score(X_test, y_test).round(3)

# C4.5 Decision Tree
dtree = tree.DecisionTreeClassifier(max_depth=2)
dtree.fit(X_train, y_train)

# Cross-validation scoring
print(f"5-fold C-V score: {cross_val_score(dtree, X_train,
    ↪ y_train, cv=5, scoring='accuracy').mean().round(3)*100}%")

# Scoring against untouched test data
y_true = y_test
y_pred = dtree.predict(X_test)
print(classification_report(y_true, y_pred,
    ↪ target_names=['Success', 'Failure']))
```

```
dtree.score(X_test, y_test).round(3)

# Ada-Boost C4.5
ada = AdaBoostClassifier(n_estimators=10)
ada.fit(X_train,y_train)

# Cross-validation scoring
print(f"5-fold C-V score: {cross_val_score(ada, X_train,
↪ y_train, cv=5, scoring='accuracy').mean().round(3)*100}%")

# Scoring against untouched test data
y_true = y_test
y_pred = ada.predict(X_test)
print(classification_report(y_true, y_pred,
↪ target_names=['Success', 'Failure']))

ada.score(X_test, y_test).round(3)

clf = SVC(kernel='linear', C=0.5)
clf.fit(X_train, y_train)

# Cross-validation scoring
print(f"5-fold C-V score: {cross_val_score(clf, X_train,
↪ y_train, cv=5, scoring='accuracy').mean().round(3)*100}%")

# Scoring against untouched test data
y_true = y_test
y_pred = clf.predict(X_test)
print(classification_report(y_true, y_pred,
↪ target_names=['Success', 'Failure']))

clf.score(X_test, y_test).round(3)
```

Deep Learning Model

```

import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix,
↳ classification_report

df = pd.read_csv("machine_learning_final_dataset.csv")
df = df.sample(frac=1, random_state=1112)    # shuffle the
↳ dataframe rows
y = df.pop('successful').values              # "pop" out the
↳ class column

#df = pd.get_dummies(df, columns=['weekday_of_launch',
↳ 'month_of_launch', 'country', 'parent_name', 'name']) #
↳ one-hot encoding
df = df.drop(columns=['weekday_of_launch', 'month_of_launch',
↳ 'country', 'parent_name', 'name']) # discard one-hot
↳ encoded features

#df = df[['blurb_words', 'blurb_length', 'avg_vader_score',
↳ 'avg_comment_words', 'avg_syllable_count',
↳ 'avg_comment_to_word_ratio', 'avg_big_word_ratio',
↳ 'flesch_kincaid_ease', 'flesch_kincaid_grade',
↳ 'avg_creator_vader_score',
↳ 'description_flesch_kincaid_grade',
↳ 'description_flesch_kincaid_ease', 'description_length',
↳ 'comment_count' ]] # keep linguistic features only

```

```
#df = df[['staff_pick', 'average_pledge', 'usd_goal',
↪ 'tier_count']] # keep financial features only

df = df.drop(columns=['backers_count', 'usd_pledged',
↪ 'pledged_vs_goal', 'comment_count', 'new_backers_count'])
↪ # drop features correlated with class

X_train, X_test, y_train, y_test = train_test_split(df, y,
↪ test_size=0.2, random_state=1113)

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.fit_transform(X_test)

EPOCHS = 20 # number of training epochs
BATCH_SIZE = 64 # number of training examples to
↪ process at the same time through the network
LEARNING_RATE = 0.001 # the learning rate of the optimizer
↪ algorithm
DROPOUT = 0.1

class trainData(Dataset):
    def __init__(self, X_data, y_data):
        self.X_data = X_data
        self.y_data = y_data
    def __getitem__(self, index):
        return self.X_data[index], self.y_data[index]
    def __len__(self):
        return len(self.X_data)

class testData(Dataset):
    def __init__(self, X_data, y_data):
        self.X_data = X_data
        self.y_data = y_data
```

```

def __getitem__(self, index):
    return self.X_data[index], self.y_data[index]
def __len__(self):
    return len(self.X_data)

train_data = trainData(torch.FloatTensor(X_train),
                        torch.FloatTensor(y_train))
test_data = testData(torch.FloatTensor(X_test),
                     torch.FloatTensor(y_test))

train_loader = DataLoader(dataset=train_data,
    ↪ batch_size=BATCH_SIZE, shuffle=True)
test_loader = DataLoader(dataset=test_data, batch_size=1)

# A.N.N. Model definition
class binaryClassification(nn.Module):
    def __init__(self):
        super(binaryClassification, self).__init__()
        self.layer_1 = nn.Linear(len(df.columns), 256)
        self.layer_2 = nn.Linear(256, 256)
        self.layer_3 = nn.Linear(256, 256)
        self.layer_out = nn.Linear(256, 1)

        self.relu = nn.ReLU()
        self.dropout = nn.Dropout(p=DROPOUT)
        self.batchnorm1 = nn.BatchNorm1d(256)
        self.batchnorm2 = nn.BatchNorm1d(256)
        self.batchnorm3 = nn.BatchNorm1d(256)

    def forward(self, inputs):
        x = self.relu(self.layer_1(inputs))
        x = self.batchnorm1(x)
        x = self.dropout(x)
        x = self.relu(self.layer_2(x))
        x = self.batchnorm2(x)

```

```
x = self.dropout(x)
x = self.relu(self.layer_3(x))
x = self.batchnorm3(x)

x = self.layer_out(x)

return x

device = torch.device("cpu")
model = binaryClassification()
model.to(device)
print(model)
criterion = nn.BCEWithLogitsLoss()
#criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=LEARNING_RATE)

def binary_acc(y_pred, y_test):
    y_pred_tag = torch.round(torch.sigmoid(y_pred))

    correct_results_sum = (y_pred_tag == y_test).sum().float()
    acc = correct_results_sum/y_test.shape[0]
    acc = torch.round(acc * 100)

    return acc

model.train()
for e in range(1, EPOCHS+1):
    epoch_loss = 0
    epoch_acc = 0
    for X_batch, y_batch in train_loader:
        X_batch, y_batch = X_batch.to(device),
        ↪ y_batch.to(device)
        optimizer.zero_grad()

        y_pred = model(X_batch)
```

```
loss = criterion(y_pred, y_batch.unsqueeze(1))
acc = binary_acc(y_pred, y_batch.unsqueeze(1))

loss.backward()
optimizer.step()

epoch_loss += loss.item()
epoch_acc += acc.item()

model.eval() # turn model to evaluation mode
epoch_acc = 0
with torch.no_grad():
    for X_batch, y_batch in test_loader:
        X_batch, y_batch = X_batch.to(device),
        ↪ y_batch.to(device)
        y_pred = model(X_batch)
        acc = binary_acc(y_pred, y_batch.unsqueeze(1))
        epoch_acc += acc.item()

print(f'Acc: {epoch_acc/len(test_loader):.3f}') # change to
↪ test_loader
```