Explaining Successful Docker Images using Pattern Mining Analysis

Riccardo Guidotti[1,2], Jacopo Soldani[1], Davide Neri[1], and Antonio Brogi[1]

[1] University of Pisa, Largo B. Pontecorvo, 3, Pisa, Italy, name.surname@di.unipi.it,
[2] KDDLab, ISTI-CNR, Via G. Moruzzi, 1, Pisa, Italy, guidotti@isti.cnr.it

Abstract. Docker is on the rise in today's enterprise IT. It permits shipping applications inside portable containers, which run from so-called Docker images. Docker images are distributed in public registries, which also monitor their popularity. The popularity of an image directly impacts on its usage, and hence on the potential revenues of its developers. In this paper, we present a frequent pattern mining-based approach for understanding how to improve an image to increase its popularity. The results in this work can provide valuable insights to Docker image providers, helping them to design more competitive software products.

1 Introduction

Docker images are the de-facto standard for container-based virtualization in enterprise IT [14]. The aim of container-based virtualization is to provide a simple yet powerful solution for running software applications in isolated virtual environments called *containers* [21]. Containers have faster start-up time and less overhead than other existing visualization approaches, like virtual machines [10]. Docker permits building, shipping, and running applications inside portable containers. Docker containers run from Docker images, which are the read-only templates used to create them. A Docker image packages a software together with all the dependencies needed to run it (e.g., binaries, libraries).

Docker also provides the ability to distribute and search images through so-called *Docker registries*. Through Docker registries any developer can create and distribute its own created images, so that other users have at their disposal plentiful repositories of heterogeneous, ready-to-use images. In this scenario, public registries as the official Docker Hub are playing a central role in the distribution of Docker images.

DockerFinder [4] enhances the support for searching Docker images. DockerFinder allows to search for images based on multiple attributes. These attributes include (but are not limited to) the name and size of an image, its popularity within the Docker community (measured in terms of so-called *pulls* and *stars*), the operating system distribution they are based on, and the software distributions they support (e.g., `java 1.8` or `python 2.7`). DockerFinder automatically crawls all such information from the Docker Hub and by directly inspecting the Docker containers that run from images. In this way, DockerFinder builds its own dataset of Docker images.

The popularity of an image directly impacts on its usage [13]. Understanding the reputation and usage of an image is important as for every other kind of open-source software. The higher is the usage and the endorsement of an open-source software, the higher are the chances of revenue from related products/services, the self-marketing and the peer recognition for its developers [8].

The main objective of this paper is to understand which are the *rules* characterizing the patterns leading to popular images both in terms of registered pulls and explicit endorsement from the users. With pattern we refer to the typical image composition in terms of operating system distribution, installed softwares, number of layers, size of the image, etc. In order to perform pattern analysis we develop an accurate data-to-model transformation, which considers the possible data types and distributions of variables. Finally, we extract itemsets and rules using the well known FP-Growth algorithm [6].

The analysis of such rules and itemsets highlights that most of the Docker images follow rules which are very common but that do not lead to a consistent level of popularity, while some rules only used by a small portion of the images are very stable and predictive of high level of popularity both in terms of pulls and stars. Moreover, most of the rules leading to the highest success are satisfied only by images officially supporting commercialized software distributions, and as we proof in the experiments this does not happen by chance.

The rest of the paper is organized as follows. Sect. 2 provides background on Docker. Sect. 3 formalizes the data type, the data transformation and the popularity rules used to study Docker images. Sect. 4 presents a dataset of Docker images and the analysis illustrating the main patterns hidden in the data. Sects. 5 and 6 discuss related work and draw some concluding remarks.

2 Background

Docker is a platform for running applications in isolated user-space instances, called *containers*. Each Docker *container* packages the applications to run, along with all the software support they need (e.g., libraries, binaries, etc.).

Containers are built by instantiating so-called Docker *images*, which can be seen as read-only templates providing all instructions needed for creating and configuring a container (e.g., software distributions to be installed, folders/files to be created). A Docker image is made up of multiple file systems layered over each other. A new Docker image can be created by loading an existing image (called *parent* image), by performing updates to that image, and by committing the updates. The commit will create a new image, made up of all the layers of its parent image plus one, which stores the committed updates.

Existing Docker images are distributed through Docker *registries*, with the Docker Hub (`hub.docker.com`) being the main registry for all Docker users. Inside a registry, images are stored in *repositories*, and each repository can contain multiple Docker images. A repository is usually associated to a given software (e.g., *Java*), and the Docker images contained in such repository are different versions of such software (e.g., *jre7*, *jdk7*, *open-jdk8*, etc.). Repositories are di-

vided in two main classes, namely *official* repositories (devoted to curated sets of images, packaging trusted software releases — e.g., *Java*, *NodeJS*, *Redis*) and *non-official* repositories, which contain software developed by Docker users.

The success and popularity of a repository in the Docker Hub can be measured twofold. The number of *pulls* associated to a repository provides information on its actual usage. This is because whenever an image is downloaded from the Docker Hub, the number of pulls of the corresponding repository is increased by one. The number of *stars* associated to a repository instead provides significant information on how much the community likes it. Each user can indeed "star" a repository, in the very same way as eBay buyers can "star" eBay sellers.

DockerFinder is a tool for searching for Docker images based on a larger set of information with respect to the Docker Hub. DockerFinder automatically builds the description of Docker images by retrieving the information available in the Docker Hub, and by extracting additional information by inspecting the Docker containers. The Docker image descriptions built by DockerFinder are stored in a JSON format[3], and can be retrieved through its GUI or HTTP API.

Among all information retrieved by DockerFinder, in this work we shall consider the size of images, the operating system and software distributions they support, the number of layers composing an image, and the number of pulls and stars associated to images. A formalization of the data structures considered is provided in the next section. Moreover, in the experimental section we will also observe different results for official and non-official images.

3 Proposed Analytical Model

We hereafter provide a formal representation of Docker images, and we then illustrate how to model docker patterns and popularity/endorsement rules.

A *Docker image* can be represented as a tuple indicating the operating system it supports, the number of layers forming the image, its compressed and actual size, and the set of software distributions it supports. For the sake of readability, we shall denote with $\mathbb{U}_{os}$ the finite universe of existing operating system distributions (e.g., "`Alpine Linux v3.4`","`Ubuntu 16.04.1 LTS`"), and with $\mathbb{U}_{sw}$ the finite universe of existing software distributions (e.g., "`java`", "`python`").

Definition 1 (Image). *Let $\mathbb{U}_{os}$ be the finite universe of operating system distributions and $\mathbb{U}_{sw}$ be the finite universe of software distributions. We define a Docker* image *I as a tuple $I = \langle os, layers, size_d, size_a, \mathcal{S} \rangle$ where*
- $os \in \mathbb{U}_{os}$ is the operating system distribution supported by the image I,
- $layers \in \mathbb{N}$ is the number of layers stacked to build the image I,
- $size_d \in \mathbb{R}$ is the download size[4] of I,
- $size_a \in \mathbb{R}$ is the actual size[5] of I, and
- $\mathcal{S} \subseteq \mathbb{U}_{sw}$ is the set of software distributions supported by the image I.

[3] An example of raw Docker image data is available at `https://goo.gl/hibue1`.

[4] As images are downloaded as compressed archives, their download size correspond to their compressed size (in GBs).

[5] The actual size of an image corresponds to its decompressed size (in GBs).

A concrete example of a Docker image I is the following
$I = \langle$`Ubuntu 16.04 LTS`, `6`, `0.78`, `1.23`, {`python,perl,curl,wget,tar`}$\rangle$

A *repository* contains multiple Docker images, and it stores the amount of pulls and stars associated to the images it contains. The pulls highlights the popularity of a repository, while the stars its endorsement. The main difference between pulls and stars is that stars are a *direct* appreciation of the users, while pulls are an *indirect* appreciation because a repository can be downloaded but not appreciated.

Definition 2 (Repository). *Let $\mathbb{U}_I$ be the universe of available Docker images. We define a* repository *of images as a triple $R = \langle p, s, \mathcal{I} \rangle$ where*
- $p \in \mathbb{R}$ is the number (in millions) of pulls from the repository R,
- $s \in \mathbb{N}$ is the number of stars assigned to the repository R, and
- $\mathcal{I} \subseteq \mathbb{U}_I$ is the set of images contained in the repository R.

For each repository, the number of pulls and stars is not directly associated with a specific image, but it refers to the overall repository. We hence define the notion of *imager*, viz., an image that can be used as a "representative image" for a repository. An imager essentially links the pulls and stars of a repository with the characteristic of an image contained in such repository.

Definition 3 (Imager). *Let $R = \langle p, s, \mathcal{I} \rangle$ be a repository, and let $I = \langle os, layers, size_d, size_a, \mathcal{S} \rangle \in \mathcal{I}$ be one of the images contained in R. We define an* imager I_R *as a tuple directly associating the pulls and stars of R with I, viz.,*

$$I_R = \langle p, s, I \rangle = \langle p, s, \langle os, layers, size_d, size_a, \mathcal{S} \rangle \rangle.$$

A concrete example of imager I_R is the following: I_R=$\langle$`1.3`, `1678`, $\langle$`Ubuntu 16.04 LTS`, `6`, `0.7`, `1.2`, {`python,perl,curl,wget`}$\rangle\rangle$. It is worth highlighting that an imager can be formed by picking any image I contained in R, provided that I can be considered a "medoid" [22] representing the set of images contained in R.

In order to perform frequent pattern mining analysis on imagers, we must "flatten" their representation and turn them into itemsets [1, 22]. We hence provide a translation from the tuple representing an imager into a set of items, taken from discrete domains. The latter also means that the numerical domains of pulls, stars, layers and sizes have to be discretized into intervals, which will be considered instead of the concrete numeric values. The notion of *imagerset* is defined precisely to accomplish to this purpose.

Definition 4 (Imagerset). *Let I_R=$\langle p, s, I \rangle$=$\langle p, s, \langle os, layers, size_d, size_a, \mathcal{S} \rangle \rangle$ be an imager. Let also $\mathbb{P}$, $\mathbb{S}$, $\mathbb{L}$, $\mathbb{S}_b$, $\mathbb{S}_a$ be the discretizations of the numeric domains of pulls, stars, layers, download sizes and compressed sizes, respectively. The* imagerset $\mathsf{I_R}$ *corresponding to I_R is defined as follows: $\mathsf{I_R} = \{\overline{p}\} \cup \{\overline{s}\} \cup \{os\} \cup \{\overline{layers}\} \cup \{\overline{size_d}\} \cup \{\overline{size_a}\} \cup \mathcal{S}$ where $\overline{x}$ denotes the interval corresponding to the value x in its discretized domain (e.g., $\overline{p}$ denotes the class p in $\mathbb{P}$).*

According to this definition, and assuming a given discretization, the previous imager I_R taken as example becomes the following imagerset $\mathsf{I_R}$:

$$\mathsf{I_R} = \{\texttt{1.0} \leq p < \texttt{3.0}, \texttt{1200} \leq s < \texttt{1800}, \texttt{Ubuntu 16.04 LTS}, \\ \texttt{5} \leq layers < \texttt{10}, \texttt{0.4} \leq size_d < \texttt{0.8}, \texttt{1.0} \leq size_a < \texttt{1.5}, \\ \texttt{python}, \texttt{perl}, \texttt{curl}, \texttt{wget}\}.$$

Imagersets can then be exploited to determine popularity patterns, expressed as rules determining the popularity of an imager based on its technical contents. Following [1,22], each rule is of type $\mathsf{X} \rightarrow y$, where X is an itemset containing an operating system distribution, a class of layers, a compressed size, a download size and/or a set of supported software distribution. y is instead the popularity of an imager, expressed in terms of either pulls or stars.

Definition 5 (Popularity Rule). *Let $\mathbb{P}$, $\mathbb{S}$, $\mathbb{L}$, $\mathbb{S}_b$, $\mathbb{S}_a$ be the discretizations of the numeric domains of pulls, stars, layers, download sizes and compressed sizes, respectively. A* pulls popularity rule *is a pattern $\mathsf{X} \rightarrow y$ where*
- $\mathsf{X} \subseteq \mathbb{U}_{os} \cup \mathbb{L} \cup \mathbb{S}_b \cup \mathbb{S}_a \cup \mathbb{U}_{sw}$ is an itemset, and
- $y \in \mathbb{P}$ is the popularity level expressed as pulls.
A stars popularity rule *is defined analogously (with $y \in \mathbb{S}$).*

We will now exploit our modelling to analyse concrete data.

3.1 Implementing Models Transformation

In order to transform the continuous attributes $size_d, size_a, p, s$ into corresponding, discretized intervals, it is important to consider their distributions. Indeed, these attributes have a long tailed distribution with few imagers having a small set of high values, while most of the imagers are characterized by a large and various set of low values. A traditional *natural binning* [22] would result in a discretization placing most of different low values in the long tail in a single bin. This would annihilate any difference, hence resulting in a biased data model.

In order to overcome this issue we exploit the "knee method" [22] that first sorts a variable x, then considers the curve described by the sorted x, and after that it selects a threshold point pt on such curve. The latter is the point having the maximum distance from the closest point on the straight line passing through the minimum and the maximum values of x on the considered curve (examples in Figure 2). We can then apply the natural binning only on the values lower than the threshold pt, as on these values the long tail distribution effect is less present or not present at all. Finally, the set of obtained bins is extended by including an additional bin containing all the values higher than the threshold pt.

3.2 Implementing Pattern Extraction

The transformation described in the previous section allows us to turn a given set of imagers $I_R^{\{\}} = \{I_{R1}, ..., I_{Rn}\}$ into a set of imagersets $\mathsf{I}_\mathsf{R}^{\{\}} = \{\mathsf{I}_{\mathsf{R}1}, ..., \mathsf{I}_{\mathsf{R}n}\}$. This transformation enables the usage of common algorithms for frequent pattern mining like Apriori, Eclat and FP-Growth [1, 6, 22]. All these approaches extract the rules form the retrieved frequent itemsets. Since we are interested in analyzing also the frequent itemsets besides the popularity rules we do not

	$size_d$	$size_a$	*layers*	$\lvert\mathcal{S}\rvert$	*pulls*	*stars*
$\tilde{x}$	0.16	0.41	10.00	8.00	0.06	26.0
μ	0.27	0.64	12.67	7.82	6.70	134.46
σ	0.48	1.11	9.62	2.26	46.14	564.21

Table 1: Statistics of imagers: median $\tilde{x}$, mean μ and standard deviation σ.

considers algorithms able to directly extract rules that have a strong relationship with a target attribute (i.e., the popularity in our case) such as algorithms for subgroup discovery [9] and contrast sets detection [2].

Given as input a set of sets of items (viz., $\mathsf{I}_\mathsf{R}^{\{\}}$), such algorithms can be exploited to determine *(i)* the set of *itemsets* whose *support* σ is higher or equal than a user defined threshold *min_sup*, and *(ii)* the set of *rules* whose *confidence* c is higher or equal than a user defined threshold *min_conf*.

Given an itemset X, its support $\sigma(\mathsf{X})$ with respect to a set of imagersets $\mathsf{I}_\mathsf{R}^{\{\}}$ is defined as the proportion of imagers that contain the itemset X [22], namely $\sigma(\mathsf{X}) = \frac{|\{\mathsf{I}_\mathsf{R} \in \mathsf{I}_\mathsf{R}^{\{\}} \mid \mathsf{X} \subseteq \mathsf{I}_\mathsf{R}\}|}{|\mathsf{I}_\mathsf{R}^{\{\}}|}$. The confidence of a rule instead indicates how often such rule is true. Given a rule $\mathsf{X} \rightarrow y$, its confidence $c(\mathsf{X} \rightarrow y)$ with respect to a set of imagersets $\mathsf{I}_\mathsf{R}^{\{\}}$ is defined as the proportion of the imagers that contains X which also contains y. We recall that, in this paper, we shall consider popularity rules, i.e., $y \in \mathbb{P} \cup \mathbb{S}$. $c(\mathsf{X} \rightarrow y) = \frac{\sigma(\mathsf{X} \cup \{y\})}{\sigma(\mathsf{X})}$. We also recall two indicators that can be observed from the output of the above mentioned algorithms, viz., *coverage* and *lift*. The rule *coverage* is the proportion of records that satisfy the antecedent X of a rule: $coverage(\mathsf{X} \rightarrow y) = \sigma(\mathsf{X})$. The *lift* is the ratio of the support to that expected if X and y were independent: $lift(\mathsf{X} \rightarrow y) = \frac{\sigma(\mathsf{X} \cup \{y\})}{\sigma(\mathsf{X}) \cdot \sigma(y)}$. A lift equals to 1 implies that the probability of occurrence of the antecedent and that of the consequent are independent of each other. A lift strictly higher than 1 indicates the degree to which those two occurrences are dependent on one another, and makes the rule potentially useful for predicting the popularity.

It is finally worth recalling the definition of two particular types of itemsets, as they will be used in the following section. An itemset X is *maximal* if none of its supersets has a support greater or equal than *min_sup*, while it is *closed* if all its supersets have a lower support than $\sigma(\mathsf{X})$. We will not consider normal frequent itemsets, because maximal and closed itemsets generalize and capture variegate compositions while maintaining a better/higher level of support.

4 Experiments

4.1 Dataset and Experimental Setting

DockerFinder autonomously collects information on all the images available in the Docker Hub that are contained in official repositories or in repositories that have been starred by at least three different users. The datasets collected

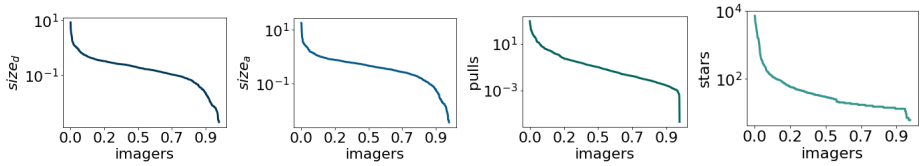

Fig. 1: Semilog distribution of $size_d$, $size_a$, pulls and stars.

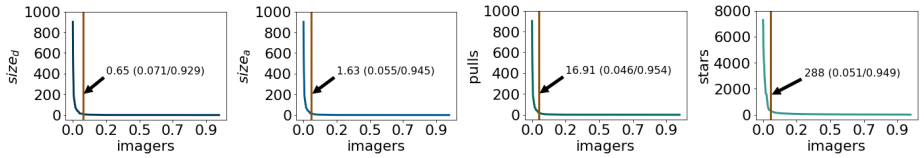

Fig. 2: Knee method effect. Numbers in parentheses indicate data used for the bin of the highest values (left) and all the rest using equal width binning (right).

by DockerFinder[6] ranges from January 2017 to March 2018 at irregular intervals. If not differently specified in this work we refer to the most recent backup where 132,724 images are available. Since performing frequent pattern mining with the aim of understanding the rules leading to successful imagers requires a notion of popularity, i.e., pulls or stars, from the available images we select 1,067 imagers considering for each repository the "latest" image. We leave as future work the investigation of the effect of considering other extraction of imagers. Some examples can be the smallest image, the one with more softwares, or a medoid or centroid of each repository.

Statistical details of the imagers extracted from the principal dataset analyzed can be found in Table 1. As anticipated in the previous section, $size_d$, $size_a$, p and s follow a long tailed distribution highlighted by the large difference between the median $\tilde{x}$ and the mean μ in Table 1. The power-law effect is stronger for *pulls* and *stars* (see Fig. 1). There is a robust Pearson correlation between pulls and stars of 0.76 (p-value 1.5e-165). However, saying that a high number of pulls implies a high number of stars (or vice versa) could be a tall statement. For this reason we report experiments for both popularity measures. There are no other relevant correlations. We highlight that there are 50 different *os* and the most common ones are `Debian GNU/Linux 8 (jessie)`, `Ubuntu 14.04.5 LTS` and `Alpine Linux v3.4`. The six most common software distributions among the 28 available (without considering the version) are `erl`, `tar`, `bash`, `perl`, `wget`, `curl`, and they appear in more than 55% of the imagers. In order to avoid considering the obvious itemsets always containing such software distributions, we remove them for the imagers.

Fig. 2 highlights the long tail of the aforementioned variables and which is the portion of data used for the bin containing the highest values (left of the parentheses) and all the rest using equal width binning (right of the parentheses).

[6] Publicly available at `https://goo.gl/ggvKN3`.

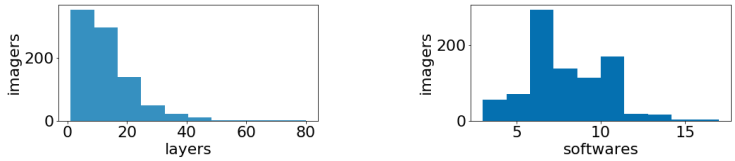

Fig. 3: Distributions of the number of layers (left) and of that of softwares (right).

{$0.0037 \leq size_a < 0.0993$, $0.0019 \leq size_d < 0.0419$, httpd, ash, unzip} (0.1047)
{ping, git, python} (0.0956)
{pip, git, python} (0.0853)
{ping, unzip, python} (0.0751)
{npm, node, git, python} (0.0728)
{$9 \leq softwares < 10$, Debian GNU/Linux 8 (jessie), git, python} (0.0660)
{java, Debian GNU/Linux 8 (jessie), unzip} (0.0648)
{Alpine Linux v3.4, httpd, ash, unzip} (0.0637)
{java, git, unzip, python} (0.0626)
{$3 \leq layers < 5$, ash, unzip} (0.0569)

Fig. 4: Maximal itemsets top ten sorted by support.

This operation removes the bias before the discretization that can be applied with natural binning on more than 94% of the variables. On the other hand, the number of *layers* and the number of softwares $|\mathcal{S}|$ do not suffer of this problem and thus can be directly discretized (see Fig. 3).

The imagers to imagerset transformation and the cleaning steps return an imagerset $\mathsf{I}_{\mathsf{R}}^{\{\}}$ with an average imagerset size $|\mathsf{I}_{\mathsf{R}}|$ of 19.05 ± 6.32. We underline that the high variability of the imagersets is given by the softwares components $\mathcal{S}$ as all the other characteristics are fixed. These imagersets $\mathsf{I}_{\mathsf{R}}^{\{\}}$ are given in input to a frequent pattern mining algorithm.

Since we do not focus on particular types of itemsets or rules, a "classic" pattern mining algorithm is suitable for assessing this task. Even though performance is not an issue in this application, among the existing frequent pattern mining algorithms we selected *FP-Growth* as from the state-of-the-art it shown to have the best performances [6]. Thus, it would be the best choice in presence of larger datasets. Other pattern mining algorithms would have returned very similar results. We leave the study of the impact of the selected frequent pattern mining algorithm for future works. In particular, we used the FP-Growth Python implementation of the `pyfim` library[7]. As threshold parameters we fixed $min\_sup = 0.05$ and $min\_conf = 0.1$. We exploited such low values because we are interested not only in unveiling the most common patterns and rules, but especially those leading to the highest target values in terms of pulls and stars.

4.2 Pattern Mining Analysis

We hereby report on some of the most interesting results of the frequent pattern mining analysis performed specifically for itemsets and popularity rules[8].

[7] `http://www.borgelt.net/pyfim.html`

[8] The python code and the list of all the itemsets and popularity rules extracted can be found at `https://github.com/di-unipi-socc/DockerImageMiner`.

{httpd, ash, unzip} (0.1695)
{$0.0019 \leq size_d < 0.0419$, ash, unzip} (0.1286)
{$0.0037 \leq size_a < 0.0993$, ash, unzip} (0.1251)
{$0.0037 \leq size_a < 0.0993$, $0.0019 \leq size_d < 0.0419$, ash, unzip} (0.1229)
{git, unzip, python} (0.1229)
{Debian GNU/Linux 8 (jessie), git, python} (0.1149)
{$0.0019 \leq size_d < 0.0419$, httpd, ash, unzip} (0.1104)
{$0.0037 \leq size_a < 0.0993$, httpd, ash, unzip} (0.1069)
{$0.0037 \leq size_a < 0.0993$, 0.0019 <= size < 0.0419, httpd, ash, unzip} (0.1047)
{$9 \leq |\mathcal{S}| < 10$, git, python} (0.1035)

Fig. 5: Closed itemsets top ten sorted by support.

{$0.00 \leq p < 0.07$} $\leftarrow$ {$8 \leq |\mathcal{S}| < 9$, ping, unzip, python} (1.00, 1.94)
{$0.00 \leq p < 0.07$} $\leftarrow$ {$10 \leq |\mathcal{S}| < 11$, pip, git, unzip, python} (1.00, 1.94)
{$0.00 \leq p < 0.07$} $\leftarrow$ {Ubuntu 16.04.2 LTS, git, python} (1.00, 1.94)
{$0.00 \leq p < 0.07$} $\leftarrow$ {$size_d > 0.6419$, java, git, unzip} (0.94, 1.83)
{$0.00 \leq p < 0.07$} $\leftarrow$ {$size_d > 0.6419$, java, git, unzip, python} (0.93, 1.82)

Fig. 6: Pulls popularity rules top five sorted by confidence (first value in parentheses).

Itemsets. FP-Growth with $min\_sup = 0.05$ retrieved 21 maximal itemsets and 45 closed itemsets having at least three components (i.e., $|\mathsf{X}| \geq 3$).

Figs. 4 and 5 report the top ten of the extracted patterns sorted by decreasing support σ (in parentheses, on the right). We underline that in these itemsets we do not consider pulls and stars as they are accurately analyzed in the popularity rules. We can notice that the closed itemset {`httpd`, `ash`, `unzip`} has a high support and it is contained in the maximal itemsets. Thus, it is a very typical and common pattern. Something similar happens for the pair {`git`, `python`}.

Some itemsets are also augmented with $9 \leq |\mathcal{S}| < 10$, signaling that very commonly there are nine softwares and among them, besides the very common six filtered out, there are also `git` and `python`. Other common softwares combinations can be read in Fig. 4. By looking at sizes, we find $0.0037 \leq size_a < 0.0993$, $0.0019 \leq size_d < 0.0419$. This highlights that many Docker images are "light images" with an average compression ratio of 0.4. Finally, we underline that the maximal itemsets are more related to the software composition, while the closed itemsets to the Docker images size. Furthermore, `java`, one of the most common programming language and tool is present only in two of reported and most supported itemsets. The reason could be that a key feature of Docker images is lightness that is generally not a prerogative of `java`.

Popularity Rules. Using FP-Growth with $min\_sup = 0.05$ and $min\_conf = 0.1$ and considering only rules having the antecedent part containing at least three components (i.e., $|\mathsf{X}| \geq 3$), we extracted $9,325$ popularity rules where the target is the number of pulls $y \in \mathbb{P}$, and $12,900$ where the target is the number of stars $y \in \mathbb{S}$. In the following we analyze these rules with respect to the indicators previously presented (confidence and lift) and also by focusing only the rules predicting the highest values of popularity in terms of both pulls and stars. Confidence and lift are reported in the parentheses following this order.

$\{13 \leq s < 19\} \leftarrow \{0.2019 \leq size_d < 0.2419,$ `ping, unzip`$\}$ (0.77, 2.39)
$\{13 \leq s < 19\} \leftarrow \{0.5774 \leq size_a < 0.6730,\ 10 \leq layers < 12\}$ (0.72, 2.23)
$\{13 \leq s < 19\} \leftarrow \{0.0819 \leq size_d < 0.1219,\ 5 \leq |\mathcal{S}| < 6,$ `python`$\}$ (0.71, 2.19)
$\{13 \leq s < 19\} \leftarrow \{10 \leq |\mathcal{S}| < 11,\ size_d > 0.6419,$ `unzip`$\}$ (0.70, 2.15)
$\{13 \leq s < 19\} \leftarrow \{0.9599 \leq size_a < 1.0555,$ `ping`$\}$ (0.70, 2.15)

Fig. 7: Stars popularity rules top five sorted by confidence (first value in parentheses).

Figs. 6 and 7 illustrate the five most interesting rules among the ten popularity rules with the highest confidence sorted by decreasing confidence[9]. We recall that pulls are expressed in millions. The first thing we notice is that these rules with high confidence predict low popularity levels. Thus, the most common Docker image building patterns among Docker images developers perhaps do not lead to good results in terms of popularity. This confirms the idea that many users design Docker images for private usage and they are not interested in obtaining a public recognition. A second interesting aspect is that the pull popularity rules have a higher confidence than the stars popularity rules and in general (not only looking at these top fives), stars popularity rules involves the image sizes ($size_a, size_d$). Hence, pulls popularity rules are more common than stars popularity rules and they are generated by different patterns of image development. This observation is confirmed by the data because the intersection of the imagersets covered by the stars popularity rules which are covered also by the pull popularity rules is only 0.09. Moreover, since stars are given as a direct endorsement it means that image size is a very relevant aspect for Docker users.

Even though confidence highlights common rules, it does not provide an indication of how much these rules are reliable. Indeed, all the rules reported in Figs. 6 and 7 have a high confidence but a low positive lift. This indicates that the imagerset composition suggested by the itemset X is not very predictive of the outcome y. To overcome this limitation we analyze in Figs. 8 and 9 the five most interesting rules among the ten popularity rules with the highest confidence sorted by decreasing lift. This time we can observe rules with a markedly high lift and a low confidence. By analyzing the target we notice that these rules predict high values (not the maximum value) of both pulls and stars. Hence, the rules, and consequently the patterns, which are predictors of a certain degree of success, cover less imagerset but are strongly more stable than those covering many imagerset with low popularity. The content of the itemsets of the reported rules, both for pulls and stars, is mainly related to the software composition and it is very common among the two sets of rules. The indication of these rules, which are markedly different from the previous ones, is that assembling a Docker image with these characteristics, i.e., `Ubuntu 14.04.5 LTS`, `nginx`, `ping`, `unzip`, `python`, `ruby` and an actual size of about 500MB may provide a good level of success in the community of Docker users.

Up to this point we filtered the popularity rules with respect to confidence and lift, letting emerge the most common patterns. We now wish to understand which are the itemsets leading to the highest values of pulls and stars. We report

[9] We discarded very similar rules in order to have a broader overview.

$\{8.47 \leq p < 8.55\} \leftarrow$ {nginx, ping, unzip, python} (0.11, 97.66)
$\{8.47 \leq p < 8.55\} \leftarrow$ {Ubuntu 14.04.5 LTS, ping, git, unzip, python} (0.11, 97.66)
$\{8.47 \leq p < 8.55\} \leftarrow \{0.5774 \leq size_a < 0.6730$, ping, git} (0.11, 97.66)
$\{8.47 \leq p < 8.55\} \leftarrow$ {ruby, ping, git, unzip, python} (0.11, 97.66)
$\{1.69 \leq p < 1.77\} \leftarrow \{0.3862 \leq size_a < 0.4818,\ 5 \leq layers < 7\}$ (0.11, 97.66)

Fig. 8: Pulls popularity rules top five sorted by lift (second value in parentheses).

$\{256 \leq s < 263\} \leftarrow$ {nginx, ping, unzip, python} (0.11, 97.66)
$\{256 \leq s < 263\} \leftarrow$ {Ubuntu 14.04.5 LTS, ping, git, unzip, python} (0.11, 97.66)
$\{256 \leq s < 263\} \leftarrow \{0.5774 \leq size_a < 0.6730$, ping, git} (0.11, 97.66)
$\{256 \leq s < 263\} \leftarrow$ {ruby, ping, unzip, python} (0.11, 97.66)
$\{256 \leq s < 263\} \leftarrow$ {ruby, ping, unzip} (0.11, 97.66)

Fig. 9: Stars popularity rules top five sorted by lift (second value in parentheses).

in Figs. 10 and 11 the five rules with the highest lift returning the highest values of pulls and stars, i.e., $p > 16.72$ and $s > 283$ respectively. Note that these values are those retrieved by the knee method in the imageset to imagerset transformation. First of all, we highlight that for the first time we have a high presence of the number of layers and the number of softwares of component of the itemsets. Therefore, these elements are becoming particularly interesting in defining very popular and successful images. Secondly, we notice that these rules have a confidence a bit higher than the previous set of rules observed, but also a lift coefficient markedly lower. Thus, the predictive power of these rules leading to the maximum success is not as strong as the one of the rules with the highest lift. This is because the consequence of such a success is not entirely related to the image composition but rather depends on other external and not observed factors. Something we can observe for the data we have is the fact that an imagerset is generated from an image of an official or not official repository. Examples of official images are `alpine`, `ubuntu`, `mongo`, `postgres`, `openjdk`, etc.

We underline that respecting the reported popularity rules does not automatically imply a certain degree of popularity. In other words, it is not sufficient to assemble a Docker image as described by the rules extracted to ensure successful images, as there are some external factors that can undoubtedly affect the popularity, e.g., whether a repository is official or not, or whether it uses novel, upgraded software distributions.

In order to quantitatively assess this point we perform an experiment using a null random model. We randomly select 1000 times 10 rules among all those extracted, both for pulls and stars. Then we calculate the average coverage of the selected rules among all the imagerset and among the imagerset referring only to official repositories. Finally, we compare these numbers with the average coverage of the ten rules with the highest lift returning the highest values of pulls and stars. Results are reported in Table 2. Both for pulls and stars a random selection of rules has a coverage considerably lower than the selection of the rules leading to maximum popularity values for official repositories. On the other hand, this phenomenon is not registered when all the repositories are considered. In conclusion, we can state that official repositories follow the rules reported in Figs. 10 and 11 not by chance and that in general they are less followed than

$\{p > 16.72\} \leftarrow \{3 \leq |\mathcal{S}| < 4,\ 0.12 \leq size_d < 0.16\}$ (0.4444, 9.3016)
$\{p > 16.72\} \leftarrow \{0.1949 \leq size_a < 0.2906,\ 0.0819 \leq size_d < 0.1219,$ ash, unzip$\}$ (0.33, 6.97)
$\{p > 16.72\} \leftarrow \{3 \leq |\mathcal{S}| < 4,\ 3 \leq layers < 5\}$ (0.33, 6.97)
$\{p > 16.72\} \leftarrow \{$Debian GNU/Linux 9 (stretch), java, unzip$\}$ (0.33, 6.97)
$\{p > 16.72\} \leftarrow \{0.2906 \leq size_a < 0.3862,\ 0.1219 \leq size_d < 0.1619,\ 7 \leq layers < 10\}$ (0.25, 5.23)

Fig. 10: Pulls popularity rules predicting the highest value of pulls.

$\{s > 283\} \leftarrow \{$Debian GNU/Linux 9 (stretch), java, unzip$\}$ (0.44, 8.31)
$\{s > 283\} \leftarrow \{0.1949 \leq size_a < 0.2906,\ 3 \leq layers < 5\}$ (0.33, 6.23)
$\{s > 283\} \leftarrow \{0.2419 \leq size_d < 0.2819,\ 8 \leq |\mathcal{S}| < 9,$ git, python$\}$ (0.33, 6.23)
$\{s > 283\} \leftarrow \{3 \leq |\mathcal{S}| < 4,\ 0.1219 \leq size_d < 0.1619\}$ (0.33, 6.23)
$\{s > 283\} \leftarrow \{$Alpine Linux v3.7, $0.0037 \leq size_a < 0.0993,\ 0.0019 \leq size_d < 0.0419,$
ash, unzip$\}$ (0.27, 5.10)

Fig. 11: Stars popularity rules predicting the highest value of stars.

a random selection of rules. Hence, despite the low values of confidence and rules, the rules reported in Figs. 10 and 11 are part of the reasons why official repositories are more successful besides hidden and unobserved factors.

5 Related Work

The estimation and analysis of popularity of Docker images resembles the analysis of success performed in various other domains.

A well-known domain is related to quantifying the changes in impact and productivity throughout a research career in science. [24] defines a model for the citation dynamics of scientific papers. The results uncover the basic mechanisms that govern scientific impact, and they also offer reliable measures of influence that may have potential policy implications. [18] points out that, besides dependent variables, also contextual information (e.g., prestige of institutions, supervisors, teaching and mentoring activities) should be considered. The latter holds also in our context, where we can observe that official images behave differently with respect to non-official images. Sinatra et al. [20] recently designed a stochastic model that assigns an individual parameter to each scientist that accurately predicts the evolution of her impact, from her h-index to cumulative citations, and independent recognitions (e.g., prizes). The above mentioned approaches analyze the success phenomena by assuming the existence of a mathematical formulation that try to fit on the data. In our proposal, we are not looking for just an indicator but for an explainable complex model that not only permits analyzing a population, but also to reveal suggestions for improvements.

Another domain of research where the analysis of success is relevant is sport. In [3] the level of competitive balance of the roles within the four major North American professional sport leagues is investigated. The evidence in [3] suggests that the significance of star power is uncovered only by multiplicative models (rather than by the commonly employed linear ones). As shown by our experiments, this holds also in our context, where we explain with multi typical items the co-occurrences and interdependencies that lead to a certain level of popularity or endorsement. In [5], Franck et al. provide further evidence on contextual factors, by showing that the emergence of superstars in German soccer depends

	pulls		**stars**	
	all repositories	officials	all repositories	officials
random	0.18 ± 0.45	0.14 ± 0.38	0.17 ± 0.44	0.14 ± 0.37
max popularity	0.11 ± 0.39	0.59 ± 0.79	0.12 ± 0.44	0.83 ± 1.15

Table 2: Comparison of average coverage (± standard deviation) between random selection of rules and rule predicting the maximum popularity values for all the repositories and for official repositories for pulls and stars.

not only on their investments in physical talent, but also on the cultivation of their popularity. An analysis of impact of technical features on performances of soccer teams is provided in [15]. The authors find that draws are difficult to predict, but they obtain good results in simulating (and consequently quantifying) the overall championships. Instead, the authors of [16] try to understand which are the features driving human evaluation with respect to performance in soccer.

Another field of research where the study of success and popularity is quite useful is that one of online social networks, like Facebook, Instagram, Twitter, Youtube, etc. The authors of [12] propose a method to predict the popularity of new hashtags on Twitter using standard classification models trained on content features extracted from the hashtag and on context features extracted from the social graph. The difference with our approach is that we try to extract patterns to explain the reasons of a certain degree of popularity. For understanding the ingredients of success of fashion models, the authors of [17] train machine learning methods on Instagram images to predict new popular models. Instead, in [23] a regression method to estimate the popularity of an online video (from YouTube or Facebook) measured in terms of its number of views is presented. Results show that, despite the visual content can be useful for popularity prediction before content publication, the social context represents a much stronger signal for predicting the popularity of a video.

Closer to our context, some forms of analytics have been recently applied to GitHub repositories. The authors of [25] study GitHub software version evolution by developers' activities. They define four metrics to measure commit activity and code evolution and then they adopt visualization techniques to analyze the commit logs. The authors of [26] instead study popularity of GitHub developers on a sociological basis. The study is based on follow-networks built according to the follow behavior among developers in GitHub, which allows to the authors of [26] to identify and present a set of typical patterns determining a growth of developers' popularity in social coding networks. The contextual dimension given by the social network is considered in [26] find an explosive growth of the users in GitHub and construct follow-networks according to the follow behaviors among developers in GitHub. Using this network delineates four typical social behavior patterns. Further domains where the analysis and prediction of success is a challenging task are music [19], movies [11] and school performances [7]. However, to the best of our knowledge, our approach is the first that is based on complex descriptions such as those of Docker images, and which tries to understand the reasons of popularity and endorsement.

6 Conclusion

In this paper we have proposed a methodology based on frequent pattern mining to retrieve the hidden patterns leading to the popularity of Docker images. In particular, we developed an approach to use common frequent pattern mining algorithms (such as FP-Growth), which discretizes continuous variables by taking into account their distributions. The main findings highlight that most of the images follow rules which are very common but that do not lead the Docker image to a relevant level of popularity. On the other hand, we have found some rules satisfied only by a small portion of the images, which are however very stable and predictive of a consistent level of popularity in terms of pulls and stars. Finally, we have observed that the most successful rules are followed only by so-called official Docker images.

As future work, besides testing the proposed frequent pattern mining analytical framework on other domains, we would like to strengthen the experimental section by means of a real validation which involve the usage of the rules we observed in this paper. The idea is to release on DockerHub a set of images following the aforementioned rules, and to observe the level of popularity they will be obtaining in a real case study, and how long it takes to reach the estimated values. Time is indeed another crucial component that was not considered because the current version of DockerFinder is not updating the status of a repository at constant time intervals. Another extension of this study involves to also consider the temporal dimension and the evolution of the patterns. Moreover, while in this paper we propose a reasonable analysis of Docker images using basic existing approaches, as future work we would like to consider advanced *multi-instance learning* techniques [27]. These methods allow to overtake the problem of having multiple Docker images for a single repository as they takes as input a set of labeled *bags*, each containing many instances. Finally, a natural extension of this work is to build a predictor/regressor either from scratch or on top of the popularity rules extracted and observe to which extent is possible to infer the popularity of a Docker image.

Acknowledgements. Work partly supported by the EU H2020 Program under the funding scheme "INFRAIA-1-2014-2015: Research Infrastructures" grant agreement 654024 *"SoBigData"* `http://www.sobigdata.eu`.

References

1. R. Agrawal, R. Srikant, et al. Fast algorithms for mining association rules. In *Proc. 20th int. conf. very large data bases, VLDB*, volume 1215, pages 487–499, 1994.
2. S. D. Bay and M. J. Pazzani. Detecting group differences: Mining contrast sets. *Data mining and knowledge discovery*, 5(3):213–246, 2001.
3. D. J. Berri, M. B. Schmidt, and S. L. Brook. Stars at the gate: The impact of star power on nba gate revenues. *Journal of Sports Economics*, 5(1):33–50, 2004.
4. A. Brogi, D. Neri, and J. Soldani. DockerFinder: Multi-attribute search of docker images. In *IC2E*, pages 273–278. IEEE, 2017.

5. E. Franck and S. Nüesch. Mechanisms of superstar formation in german soccer: Empirical evidence. *European Sport Management Quarterly*, 8(2):145–164, 2008.
6. J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *ACM sigmod record*, volume 29, pages 1–12. ACM, 2000.
7. J. M. Harackiewicz et al. Predicting success in college: A longitudinal study of achievement goals and ability measures as predictors of interest and performance from freshman year through graduation. *JEP*, 94(3):562, 2002.
8. A. Hars and S. Ou. Working for free? - Motivations of participating in open source projects. *IJEC*, 6(3):25–39, 2002.
9. F. Herrera, C. J. Carmona, P. González, and M. J. Del Jesus. An overview on subgroup discovery: foundations and applications. *KAIS*, 29(3):495–525, 2011.
10. A. Joy. Performance comparison between linux containers and virtual machines. In *ICACEA*, pages 342–346, March 2015.
11. B. R. Litman. Predicting success of theatrical movies: An empirical study. *The Journal of Popular Culture*, 16(4):159–175, 1983.
12. Z. Ma, A. Sun, and G. Cong. On predicting the popularity of newly emerging hashtags in twitter. *JASIST*, 64(7):1399–1410, 2013.
13. I. Miell and A. H. Sayers. *Docker in Practice*. Manning Publications Co., 2016.
14. C. Pahl, A. Brogi, J. Soldani, and P. Jamshidi. Cloud container technologies: a state-of-the-art review. *IEEE Transactions on Cloud Computing*, 2017. *[In press]*.
15. L. Pappalardo and P. Cintia. Quantifying the relation between performance and success in soccer. *Advances in Complex Systems*, page 1750014, 2017.
16. L. Pappalardo, P. Cintia, D. Pedreschi, F. Giannotti, and A.-L. Barabasi. Human perception of performance. *arXiv preprint arXiv:1712.02224*, 2017.
17. J. Park et al. Style in the age of instagram: Predicting success within the fashion industry using social media. In *CSCW*, pages 64–73. ACM, 2016.
18. O. Penner, R. K. Pan, A. M. Petersen, K. Kaski, and S. Fortunato. On the predictability of future impact in science. *Scientific reports*, 3:3052, 2013.
19. L. Pollacci, R. Guidotti, et al. The fractal dimension of music: Geography, popularity and sentiment analysis. In *GOODTECHS*, pages 183–194. Springer, 2017.
20. R. Sinatra, D. Wang, P. Deville, C. Song, and A.-L. Barabási. Quantifying the evolution of individual scientific impact. *Science*, 354(6312):aaf5239, 2016.
21. S. Soltesz et al. Container-based operating system virtualization: A scalable, high-performance alternative to hypervisors. *SIGOPS*, 41:275–287, 2007.
22. P.-N. Tan et al. *Introduction to data mining*. Pearson Education India, 2006.
23. T. Trzciński and P. Rokita. Predicting popularity of online videos using support vector regression. *IEEE Transactions on Multimedia*, 19(11):2561–2570, 2017.
24. D. Wang, C. Song, and A.-L. Barabási. Quantifying long-term scientific impact. *Science*, 342(6154):127–132, 2013.
25. Y. Weicheng, S. Beijun, and X. Ben. Mining github: Why commit stops–exploring the relationship between developer's commit pattern and file version evolution. In *APSEC*, volume 2, pages 165–169. IEEE, 2013.
26. Y. Yu, G. Yin, H. Wang, and T. Wang. Exploring the patterns of social behavior in github. In *CrowdSoft*, pages 31–36. ACM, 2014.
27. Z.-H. Zhou and M.-L. Zhang. Multi-instance multi-label learning with application to scene classification. In *NIPS*, pages 1609–1616, 2007.