Helping your Docker images to spread based on explainable models

Riccardo Guidotti[1,2], Jacopo Soldani[1], Davide Neri[1],
Antonio Brogi[1], and Dino Pedreschi[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 impacts on its actual usage, and hence on the potential revenues for its developers. In this paper, we present a solution based on interpretable decision tree and regression trees for estimating the popularity of a given Docker image, and for understanding how to improve an image to increase its popularity. The results presented in this work can provide valuable insights to Docker developers, helping them in spreading their images.

1 Introduction

Container-based virtualization provides a simple yet powerful solution for running software applications in isolated virtual environments, called *containers* [29]. Containers are rapidly spreading over the spectrum of enterprise information technology, as they feature much faster start-up times and less overhead than other existing visualization approaches, e.g., virtual machines [13].

Docker is the de-facto standard for container-based virtualization [20]. It 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 permits packaging 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 of) Docker containers through so-called Docker registries. Given that any developer can create and distribute its own created images, other users have at their disposal plentiful repositories of heterogeneous, ready-to-use images. In this scenario, public registries (e.g., Docker Hub) are playing a central role in the distribution of images.

DockerFinder [6] permits searching for existing Docker 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. With this approach, DockerFinder builds its own dataset of Docker images, which can be queried through a GUI or through a RESTful API.

The popularity of an image directly impacts on its usage [18]. Maximising the reputation and usage of an image is of course important, as for every other kind of open-source software. The higher is the usage of an open-source software, the higher are the chances of revenue from related products/services, as well as the self-marketing and the peer recognition for its developers [12].

In line with [9], the main objectives of this paper are (i) to exploit the features retrieved by DockerFinder to understand how the features of an image impact on its popularity, and (ii) to design an approach for recommending how to update an image to increase its popularity. In this perspective, we propose:

(i) DARTER (*Decision And Regression Tree EstimatoR*), a mixed hierarchical approach based on decision tree classifiers and regression trees, which permits estimating the popularity of a given Docker image, and
(ii) DIM (*Docker Image Meliorator*), an explainable procedure for determining the smallest changes that can be made to a Docker image to improve its popularity and usage.

It is worth highlighting that our approach is *explainable* by design [10]. That is, we can understand which features delineate an estimation, and we can exploit them to improve a Docker image. Besides being a useful peculiarity of the model, comprehensibility of models is becoming crucial, as the European Parliament in May 2018 adopted the GDPR for which a "right of explanation" will be required for automated decision making systems [8].

Our results show that (i) DARTER outperforms state-of-the-art estimators and that popular images are not obtained "by chance", and that (ii) DIM recommends successful improvements while minimizing the number of required changes. Thanks to the interpretability of both DARTER and DIM, we can analyze not-yet-popular images, and we can automatically determine the most recommended, minimal sets of changes allowing to improve their popularity.

The rest of the paper is organized as follows. Sect. 2 provides background on Docker. Sects. 3 and 4 illustrate the popularity problems we aim to target and show our solutions, respectively. Sect. 5 presents a dataset of Docker images and some experiments evaluating our solutions. Sects. 6 and 7 discuss related work and draw some conclusions, respectively.

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 divided 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 data structure 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 Docker Images and Popularity Problems

We hereafter provide a formal representation of Docker images, and we then illustrate the popularity problems we aim to target.

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`").

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

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 of I,
- $size_a \in \mathbb{R}$ is the actual size[4] of I, and
- $\mathcal{S} \subseteq \mathbb{U}_{sw}$ is the set of software distributions supported by the image I.

A concrete example of Docker image is the following:
⟨`Ubuntu 16.04.1 LTS`, `6`, `0.78`, `1.23`, {`python,perl,curl,wget,tar`}⟩.
A *repository* contains multiple Docker images, and it stores the amount of pulls and stars associated to the images it contains.

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 is the following:
⟨`1.3`, `1678`, ⟨`Ubuntu 16.04.1 LTS`, `6`, `0.7`, `1.2`, {`python,perl,curl,wget`}⟩⟩.
An imager can be obtained from any image I contained in R, provided that I can be considered a "medoid" representing the set of images contained in R.

We can now formalize the *popularity estimation problem.* As new Docker images will be released from other users, image developers may be interested in estimating the popularity of a new image in terms of pulls and stars.

Definition 4 (Popularity Estimation Problem). *Let $I_R = \langle p, s, I\rangle$ be an imager, whose values p and s of pulls and stars are unknown. Let also $\mathcal{I}_R$ be the context[5] where I_R is considered (viz., $I_R \in \mathcal{I}_R$). The* popularity estimation problem *consists in estimating the actual values p and s of pulls and stars of I_R in the context $\mathcal{I}_R$.*

[4] Images downloaded from registries are compressed. The download size of an image is hence its compressed size (in GBs), while its actual size is the disk space (in GBs) occupied after decompressing and installing it on a host.

[5] As I_R is the representative image for the repository R, $\mathcal{I}_R$ may be (the set of imagers representing) the registry containing the repository R.

Notice that, due to the currently available data, the popularity estimation problem is not considering time. For every image we only have a "flat" representation and, due to how DockerFinder currently works, we cannot observe its popularity evolution in time. However, the extension of the problem that considers also the temporal dimension is a future work we plan to pursue.

Image developers may also be interested in determining the minimum changes that could improve the popularity of a new image. This is formalized by the *recommendation for improvement problem.*

Definition 5 (Recommendation for Improvement Problem). *Let $I_R = \langle p, s, I \rangle$ be an imager, whose values p and s of pulls and stars have been estimated in a context X (viz., $I_R \in X$). The* recommendation for pulls improvement problem *consists in determining a set of changes C^* such that*
- $I_R \xrightarrow{C^*} I^*_R = \langle p^*, \cdot, \cdot \rangle$, with $I^*_R \in X \wedge p^* > p$, and
- $\nexists C^\dagger$ s.t. $|C^\dagger| < |C^*|$ and $I_R \xrightarrow{C^\dagger} I^\dagger_R = \langle p^\dagger, \cdot, \cdot \rangle$, with $I^\dagger_R \in X \wedge p^\dagger > p^*$
(where $x \xrightarrow{C} y$ denotes that y is obtained by applying the set of changes C to x). The recommendation for stars improvement problem *is analogous.*

In other words, a solution of the recommendation for improvement problem is an imager I'_R obtained from I_R such that I'_R is more likely to get more pulls/stars than I_R and that the number of changes to obtain I'_R from I_R is minimum.

4 Proposed Approach

We hereby describe our approaches for solving the popularity estimation problem and the recommendation for stars/pulls improvement problem.

4.1 Estimating Popularity

We propose DARTER (*Decision And Regression Tree EstimatoR*, Algorithm 1) as a solution of the popularity estimation problem[6]. From a general point of view the problem can be seen as a regression problem [30]. However, as we show in the following, due to the fact that few imagers are considerably more popular than the others and most of the imagers are uncommon, usual regression methods struggle in providing good estimation (popularity distributions are provided in Sect. 5.1). DARTER can be used to estimate both pulls p and stars s. We present the algorithm in a generalized way by considering a popularity target u.

DARTER can be summarized in three main phases. First, it estimates a popularity threshold *pt* (line 2) with respect to the known imagers $\mathcal{I}_R$. Then, it labels every image as *popular* or *uncommon* (equals to 1 and 0 respectively) with respect to this threshold (line 3). Using these labels, DARTER trains a *decision tree* Ψ (line 4). This phase could be generalized using multiple threshold and labels.

[6] DARTER is designed to simultaneously solve multiple instances of the popularity estimation problem, given a set $\mathcal{X}$ of imagers whose popularity is unknown.

```
Algorithm 1: DARTER($\mathcal{I}_R, \mathcal{X}, u$)
  Input  : $\mathcal{I}_R$ - context/set of imagers, $\mathcal{X}$ - set of images to estimate, $u$ -
           popularity type (can be equal to “pulls” or to “stars”)
  Output: $Y$ - popularity estimation
1  $T \leftarrow getPopularity(\mathcal{I}_R, u)$;                         // extract imagers popularity
2  $pt \leftarrow estimatePopularityThreshold(T)$;                      // estimate popularity threshold
3  $L \leftarrow \{l \mid l = 0 \text{ if } t < pt \text{ else } 1,\ \forall\, t \in T\}$;   // label a target value as popular or not
4  $\Psi \leftarrow trainDecisionTree(\mathcal{I}_R, L)$;                      // train decision tree
5  $\mathcal{I}_R^- \leftarrow \{I_R \mid \text{if } isPopular(\Psi, I_R) = 0\ \forall\, I_R \in \mathcal{I}_R\}$;     // classify uncommon imagers
6  $\mathcal{I}_R^+ \leftarrow \{I_R \mid \text{if } isPopular(\Psi, I_R) = 1\ \forall\, I_R \in \mathcal{I}_R\}$;     // classify popular imagers
7  $\mathcal{X}^- \leftarrow \{I_R \mid \text{if } isPopular(\Psi, I_R) = 0\ \forall\, I_R \in \mathcal{X}\}$;     // classify uncommon imagers
8  $\mathcal{X}^+ \leftarrow \{I_R \mid \text{if } isPopular(\Psi, I_R) = 1\ \forall\, I_R \in \mathcal{X}\}$;     // classify popular imagers
9  $T^- \leftarrow getPopularity(\mathcal{I}_R^-, u)$; $T^+ \leftarrow getPopularity(\mathcal{I}_R^+, u)$;   // extract popularity
10 $\Lambda^- \leftarrow trainRegressionTree(\mathcal{I}_R^-, T^-)$;              // train decision tree uncommon
11 $\Lambda^+ \leftarrow trainRegressionTree(\mathcal{I}_R^+, T^+)$;              // train decision tree popular
12 $Y^- \leftarrow estimatePopularity(\Lambda^-, \mathcal{X}^-)$;              // estimate popularity for uncommon
13 $Y^+ \leftarrow estimatePopularity(\Lambda^+, \mathcal{X}^+)$;              // estimate popularity for popular
14 $Y \leftarrow buildResult(\mathcal{X}, \mathcal{X}^-, \mathcal{X}^+, Y^-, Y^+)$;  // build final result w.r.t. original order
15 return $Y$;
```

In the second phase (lines 5-9), DARTER exploits the decision tree to classify both the imagers $\mathcal{I}_R$ and the images to estimate $\mathcal{X}$ as *popular* ($^+$) or *uncommon* ($^-$). In this task, DARTER exploits the function *isPopular*(Ψ, I_R), which follows a path along the decision tree Ψ according to the imager I_R to estimate whether I_R will be popular or uncommon (see Figure 1 *(left)*).

In the third phase (lines 10-14), DARTER trains two regression trees Λ^-, Λ^+ for uncommon and popular images, respectively. These regression trees are specialized to deal with very different types of images, which may have very different estimations on the leaves: High values for the popular regression tree, low values for the uncommon regression tree (see Figure 1 *(center) & (right)*). Finally, DARTER exploits the two regression trees to estimate the popularity of the images in X and returns the final estimation Y.

In summary, DARTER builds a hierarchical estimation model. At the top of the model, there is a decision tree Ψ that permits discriminating between popular and uncommon imagers. The leaves of Ψ are associated with two regression trees Λ^+ or Λ^-. Λ^+ permits estimating the level of popularity of an imager, while Λ^- permits estimating the level of uncommonness of an imager. It is worth noting that specialized regression trees for each leaf of Ψ could be trained. However, since in each leaf there are potentially few nodes, this could lead to model overfitting [30] decreasing the overall performance. On the other hand, the two regression trees Λ^+, Λ^- result to be much more general since they are trained on all the popular/uncommon imagers.

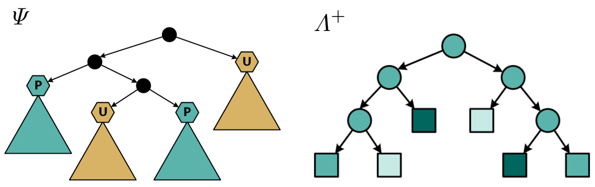

Fig. 1: Decision tree discriminating between popular and uncommon imagers (left), regression trees evaluating the popularity (center), and the degree of uncommonness (right). The regression trees are linked to corresponding leaves of Ψ: Regression trees like Λ^+ are linked to leaves marked with P (*popular*), while those like Λ^- to leaves marked with U (*uncommon*). (Best view in color.)

As an example we can consider the imager $I_R = \langle$`?`, `?`, $\langle$`Ubuntu 16.04.1 LTS`, `6`, `0.78`, `1.23`, $\{$`python, perl, curl, wget, tar`$\}\rangle\rangle$. Given $\Psi, \Lambda^+, \Lambda^-$ we can have that $isPopular(\Psi, I_R) = 0$. The latter means that I_R is uncommon, hence requiring to estimate its popularity with Λ^-, viz., the popularity of I_R is estimated as $y = estimatePopularity(\Lambda^-, I_R) = 0.56$ millions of pulls.

More in detail, we realized Algorithm 1 as follows. Function *estimatePopularityThreshold*$(\cdot)$ is implemented by the so-called "knee method" [30]. The knee method sorts the target popularity T and then, it selects the point threshold pt on the curve which has the maximum distance with the closest point on the straight line passing through the minimum and the maximum of the curve described by the sorted T (examples in Figure 4 *(bottom)*). As models for decision and regression trees we adopted an optimized version of *CART* [5]. We used the *Gini* criteria for the decision tree and the *Mean Absolute Error* for the regression trees [30]. We used a cost matrix for training the decision tree in order to improve the tree recall and precision in identifying popular images.

Finally, besides the good performance reported in the experimental section, the choice of decision and regression tree as estimation models lies in the fact that these models are easily interpretable [10]. Indeed, as shown in the following, we can extract from these trees an explanation of the estimation, and this explanation can be exploited to understand which are the changes that can lead to an improvement of the popularity of an image.

4.2 Recommending Improvements

To solve the recommendation for improvement problem we propose DIM (*Docker Image Meliorator*, Algorithm 2). First, DIM estimates the popularity y of the imager under analysis I_R in the context given by the decision tree Ψ (lines 1-2). The path along the tree leading to y constitutes the explanation for such estimation. In such terms, the proposed model is a *transparent box* which is both local and global explainable by design [10]. Then, it extracts the paths Q with a popularity higher than the one estimated for I_R (line 3), and it selects the shortest path sp among them (line 4). Finally, it returns an updated imager

```
Algorithm 2: $DIM(I_R, \Psi, \Lambda^-, \Lambda^+)$
  Input  : $I_R$ - imager to improve, $\Psi$ - decision tree, $\Lambda^-$, $\Lambda^+$ - regression trees.
  Output: $I_R^*$ - updated imager.
1 if $isPopular(\Psi, I_R) = 0$ then $y \leftarrow estimatePopularity(\Lambda^-, I_R)$ ;
2 else $y \leftarrow estimatePopularity(\Lambda^+, I_R)$ ;
3 $Q \leftarrow getPathsWithGreaterPopularity(I_R, y, \Psi, \Lambda^-, \Lambda^+)$;    // get improving paths
4 $sp \leftarrow getShortestPath(Q, I_R, y, \Psi, \Lambda^-, \Lambda^+)$;    // get shortest path
5 $I_R^* \leftarrow updateImager(I_R, sp)$;    // update docker image
6 return $I_R^*$;
```

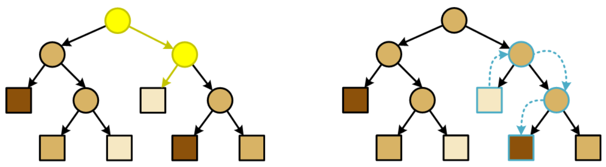
Fig. 2: The tree on the left shows the explanation of an estimation (viz., the yellow path). The tree on the right shows an example of recommendations for improvement, which is given by the shortest path leading to a leaf with a higher popularity (highlighted in blue). The latter indicates the minimum number of attribute changes that can lead to a popularity improvement. (Best view in color.)

I_R^* built from the input imager I_R by applying it the changes composing the improvement shortest path *sp* (lines 5-6).

Functions *getPathsWithGreaterPopularity* and *getShortestPath* respectively collects all the paths in the tree ending in a leaf with a popularity higher than y, and selects the shortest path among them (see Figure 2). When more than a shortest path with the same length is available, DIM selects the path with the highest overlap with the current path and with the highest popularity estimation.

Getting back to our example we have an estimation of $y = 0.56$ millions of pulls, viz., $I_R = \langle$`0.56`, ., $\langle$`Ubuntu 16.04.1 LTS`, `6`, `0.78`, `1.23`, {`python`, `perl`, `curl`, `wget`, `tar`}$\rangle\rangle$. By applying DIM on I a possible output is $I_R = \langle$`0.64`, ., $I_R^* = \langle$`Ubuntu 16.04.1 LTS`, `7`, `0.78`, `1.23`, {`python`, `perl`, `curl`, `java`}$\rangle\rangle$. That is, DIM recommends to update I_R by adding a new layer, which removes `wget` and `tar`, and which adds the support for `java`.

5 Experiments

5.1 Dataset

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 by DockerFinder[7] ranges from January 2017 to March 2018 at irregular intervals.

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

	$size_d$	$size_a$	*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

Fig. 3: Statistics of imagers, viz., median $\tilde{x}$, mean μ and standard deviation σ. The most frequent OSs and softwares are `Debian GNU/Linux 8 (jessie)`, `Ubuntu 14.04.5 LTS`, `Alpine Linux v3.4`, and `erl`, `tar`, `bash`, respectively.

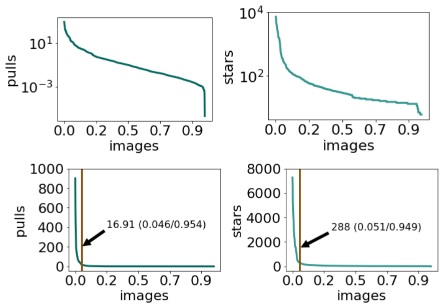

Fig. 4: Semilog pulls and stars distributions *(top)*. Knee method *(bottom)*.

If not differently specified in this work we refer to the most recent backup where 132,724 images are available. Since the popularity estimation problem require 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 (i.e., the most recent image of each repository). 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.

Details of the imagers extracted from the principal dataset analyzed can be found in Figure 3. $size_d$, $size_a$, p and s follow a long tailed distribution highlighted by the large difference between the median $\tilde{x}$ and the mean μ in Figure 3. The power-law effect is stronger for *pulls* and *stars* (see Figure 4). 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 could be a tall statement. For this reason we report experiments for both target measures. There are no other relevant correlations. 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 most common softwares among the 28 available (without considering the version) are `erl`, `tar` and `bash`.

5.2 Experimental Settings

The experiments reported in the following sections are the results of a 5-fold cross validation [30] repeated ten times. We estimate the goodness of the proposed approach by using the following indicators to measure regression performance [15,27,35]: Median absolute error (MAE), and mean squared logarithmic error ($MSLE$). These indicators are more relevant than mean absolute error, mean squared error or explained variance because we are in the case when target values have an exponential growth. MSLE penalizes an under-predicted estimate greater than an over-predicted estimate, which is precisely what we are interested in, as there are only few popular images. Besides aggregated statistics on these measures on the ten runs, we report (a sort of) area under the curve plot [30], which better enhances the overall quality of the estimation in terms

Table 1: Mean and standard deviation of MAE and MSLE for pulls and stars.

	pulls		*stars*	
Model	*MAE*	*MSLE*	*MAE*	*MSLE*
Darter	**0.222 ± 0.066**	**1.606 ± 0.268**	**19.925 ± 1.904**	**2.142 ± 0.171**
RegTree	0.355 ± 0.092	1.857 ± 0.430	22.650 ± 3.223	2.416 ± 0.233
RegKnn	0.748 ± 0.084	2.251 ± 0.195	30.020 ± 1.679	3.419 ± 0.445
Lasso	7.051 ± 1.207	4.978 ± 0.813	95.423 ± 13.445	4.767 ± 0.631
LinReg	7.998 ± 1.874	84.611 ± 123.256	112.794 ± 17.435	48.180 ± 69.352
Ridge	7.575 ± 1.736	8.236 ± 1.283	107.305 ± 15.207	5.169 ± 0.599
Null	3.471 ± 0.367	6.814 ± 1.023	3.122 ± 0.236	117.969 ± 13.459

of quantification [19], i.e., how good is the method in estimating the popularity of a set of images. We do not report learning and prediction times as they are negligible (less than a second for all the methods analyzed), and also because the experiments are more focused in highlighting the quality of the results.

For the popularity estimation problem we compare DARTER against the following baselines: Regression tree (*RegTree*) [5], k-nearest-neighbor (*RegKnn*) [1], linear regression model (*LinReg*) [36], *Lasso* model [31], and *Ridge* model [32], besides the *Null* model estimating the popularity using the mean value. We selected these approaches among the existing one because *(i)* they are adopted in some of the works reported in Section 6, *(ii)* they are interpretable [10] differently from more recent machine learning methods. On the other hand, as (to the best of our knowledge) no method is currently available for solving the recommendation for improvement problem, we compare DIM against a random null model[8].

5.3 Estimating Image Popularity

We hereby show how DARTER outperforms state-of-the-art methods in solving the popularity estimation problem. Table 1 reports the mean and standard deviation of MAE and MSLE for pulls and stars. The Null model performs better than the linear models (Lasso, LinReg and Ridge). This is probably due to the fact that linear models fail in treating the vectorized sets of softwares, which are in the image descriptions used to train the model. DARTER has both a lower mean error than all the competitors and a lower error deviation in term of standard deviation, i.e., it is more stable when targeting *pulls* and/or *stars*. The results in Table 1 summarize the punctual estimation of each method for each image in the test sets. In Figure 5 we observe the overall quantification of the estimation for the best methods. It reports the cumulative distribution of the estimation against the real values. The more a predicted curve is adherent to the real one, the better is the popularity estimation. All the approaches are good in the initial phase when uncommon images are predicted. Thus, the image difficult to estimate are those that somehow lay in the middle, and DARTER is better than the others is assessing this challenging task.

[8] The python code is available here `https://goo.gl/XnJ7yD`

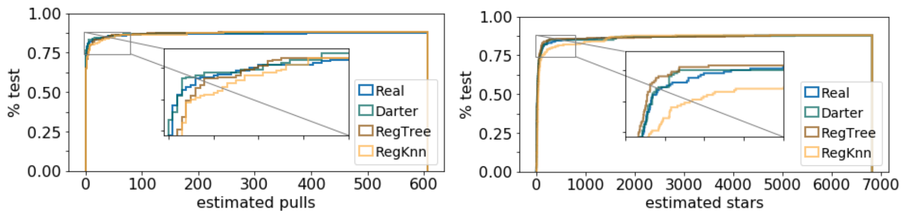

Fig. 5: Cumulative distribution of the estimations.

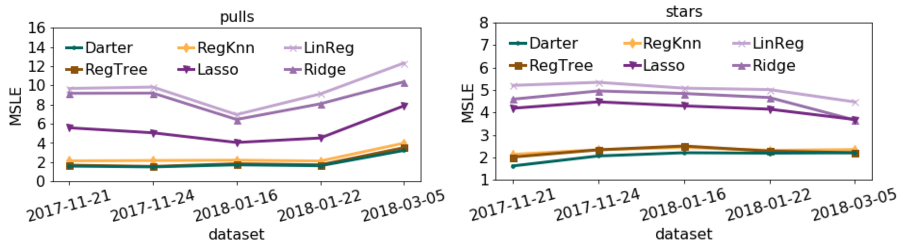

Fig. 6: Estimators stability for different datasets.

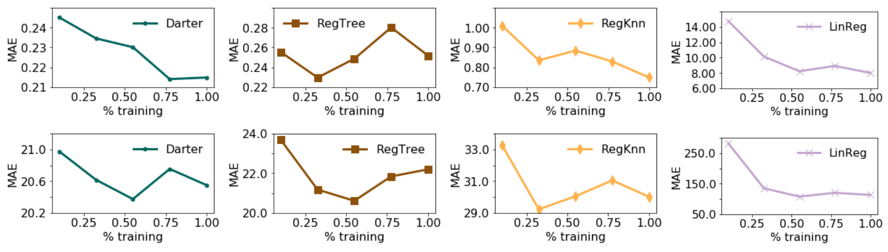

Fig. 7: MAE varying the portion of training set used: pulls first row, stars second row.

Further evidence on the stability of DARTER is provided by the fact that its MSLE keeps stable even when considering different datasets extracted by DockerFinder in different times, and steadily better than all other estimators. These results are highlighted in Figure 6 for both pulls and stars. Moreover, in Figure 7 we show the performance of the estimators in terms of MAE (pulls first row, stars second row) for increasing size of the training set in order to test the so called “cold start” problem [26]. Results show that DARTER suffer less than the other approaches when using less data for the training phase[9].

[9] The non-intuitive fact that with 50% training data MAE seems to be best for some algorithms can be explained with overfitting and partial vision of the observations.

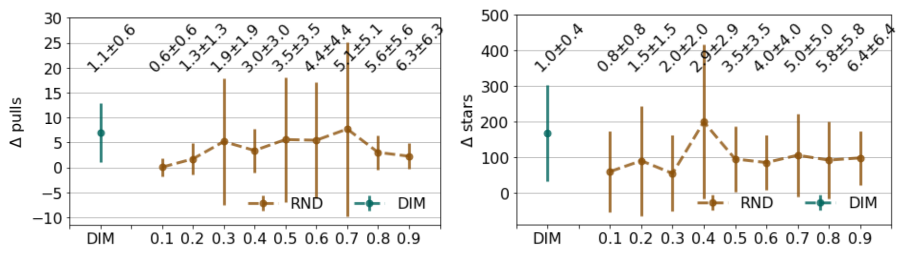

Fig. 8: Upper part: mean and standard deviation of the number of changes. Error plot: mean and standard deviation of the popularity improvement Δ.

5.4 Recommending Image Improvements

We now show that the recommendation for stars/pulls improvement problem cannot be solved by simply using a null model approach. We build a random null model (*RND*) that, given in input an imager I_R, changes a feature of I_R (e.g., *os*, *layers*, $size_d$) with probability π by selecting a new value according to the distribution of these values in the dataset, hence creating a new imager I_R^*. Then we apply RND and DIM on a test set of images. For each updated imager I_R^* we keep track of the number of changes performed to transform I_R into I_R^*, and of the variation Δ between the original popularity and the estimated popularity of I_R^*. We estimate the popularity of the improved images using DARTER since it is the best approach as we observed in the previous section.

Figure 8 reports the results of these experiment for pulls and stars. For the null model we vary $\pi \in [0.1, 0.9]$ with step 0.1. Every point in Figure 8 represents the mean popularity improvement Δ, while the vertical lines are one fourth of the standard deviation. The numbers in the upper part are the means and standard deviations of the number of changes. We notice that a random choice of the features to change can lead to an average improvement comparable to the one of DIM (π=0.7 for pulls, π=0.4 for stars). However, two aspects must not be neglected. The first one is that when RND has a higher Δ it also has a higher variability. The second one is that on average DIM uses just one or two changes to improve the image, while RND requires a consistently higher number of changes. This allows us to conclude that, given an imager I_R, DIM provides can effectively suggest how to build an imager I_R^* whose estimated popularity will be higher, and which can be obtained by applying very few changes to I_R.

5.5 Explaining Improvement Features

In this section we exploit the fact that DARTER and DIM are explainable to retrieve the most important features that should be changed to obtain the improved imager. We focus on the analysis of the most uncommon imagers by analyzing the tree Λ^- of uncommon imagers. In particular, among them we signal the subsequent: *(i)* required presence of one of the following *os*s: `Alpine Linux v3.7`, `Alpine Linux v3.2`, `Ubuntu 16.04.3 LTS`, *(ii)* having a size $size_d$ lower than 0.0241, *(iii)* having $size_a \leq 0.319$ or $size_a > 0.541$ (depending on the other features), *(iv)* having less than six software avoiding `tar` but including `ruby`.

Since the images realized by private developers can rarely reach the popularity of official imager repositories (e.g., *java*, *python*, *mongo*, etc.) we repeated the previous analysis by excluding official imagers. Results highlights that again `Alpine Linux v3.2`, `Ubuntu 16.04.3 LTS` are the required *os*s, but it is also generally recommended by DIM of having $size_a > 0.301$, $size_d \leq 0.238$ and to support the following software distributions: `gunicorn`, `go` and `ping`.

5.6 Portability of our approach

To show the portability of DARTER, we analyzed a musical Spotify-based dataset, where artists are associated with a popularity score and to the set of their tracks [25]. In this scenario, the artists play the role of "repositories", and tracks that of "images". Also in this context DARTER provids better estimations than state-of-the-art baselines (DARTER's MAE: 12.80±0.58, DARTER's MSLE: 4.36±0.175, RegTree's MAE: 13.91±0.57, RegTree's MSLE: 4.57±0.14).

6 Related Work

The problem of estimating and analysing popularity of Docker images resembles the discovery of success performed in various other domains.

A well-known domain is related to quantifying the changes in productivity throughout a research career in science. [34] defines a model for the citation dynamics of scientific papers. The results uncover the basic mechanisms that govern scientific impact. [24] 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. [28] 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 (viz., [34], [24] and [28]) model the success phenomena using the fitting of a mathematical formulation given from an assumption. 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 study of success is relevant is sport. The level of competitive balance of the roles within the four major North American professional sport leagues is investigated in [2]. The evidence 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: our complex model outperforms ordinary linear ones. Franck et al. [7] provide further evidence on contextual factors, by showing that the emergence of superstars in German soccer depends 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 [21]. The authors find that draws are difficult to predict, but they obtain good results in simulating the overall championships. Instead, the authors of [22] try to understand which are the features driving human evaluation with respect to performance in soccer. Like us, they use a complex model to mimic an artificial judge which accurately reproduces human evaluation, which permits showing how humans are biased towards contextual features.

Another field of research where the study of success and popularity is quite useful is that of online social networks, like Twitter, Instagram, Youtube, Facebook, etc. In [17], the authors 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 an explanation is not required, neither a way to produce a more popular hashtag. For understanding the ingredients of success of fashion models, the authors of [23] train machine learning methods on Instagram images to predict new popular models. Instead, Trzciński and Rokita [33] present a regression method to predict the popularity of an online video (from YouTube or Facebook) measured in terms of its number of views. 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.

Some forms of analytics have been recently applied to GitHub repositories. The authors of [14] present a first study on the main characteristics of GitHub repositories, and on how users take advantage of their main features, e.g., commits, pull requests, and issues. A deeper analysis is provided in [3], where the authors analyze various features of GitHub with respect to the impact they have on the popularity of a GitHub repository. A model for predicting such popularity is then described in [4], where multiple linear regressions are used to predict the number of stars assigned to a GitHub repository. The crucial difference between the approach in [4] and ours is that we exploit features that concretely describe a Docker image (such as the operating system and software distributions it supports, for instance), while in [4] the authors build models based only on the time series of the amounts of stars previously assigned to repositories.

Further domains where the analysis and prediction of success is a challenging task are music [25], movies [16] and school performances [11]. 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 estimate success and to provide recommendations for improvements based on an explainable model.

7 Conclusion

In this paper we have proposed DARTER and DIM, two methods specifically designed to analyze the popularity of Docker images. In particular, DARTER is a mixed hierarchical model formed by a decision tree and by two regression trees able to outperform state-of-the-art approaches in understanding the degree of popularity an image will get (in terms of pulls and stars). Moreover, DARTER predictions are explainable in terms of the characteristics of Docker images. This aspect is exploited by DIM to determine how to improve the popularity of a given image performing by applying it a minimal set of changes.

It is worth noting that DARTER and DIM are focused on the technical content of images, as their ultimate objective is to provide explainable models helping developers in analyzing and improving their Docker images. Hence, other factors that can orthogonally impact on the popularity of images (e.g., the previous reputation of a developer, or external endorsements by widely known experts in the field) are outside of the scope of this paper, as they could not lead to technical updates on images geared towards improving their popularity.

Besides testing the proposed method on other domains, we would like to strengthen the experimental section by means of a real validation. The idea is to release on Docker Hub a set of images and their improved versions and to observe how good are the prediction of DARTER and the recommendation of DIM 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. The extension of our approach to also consider time is in the scope of our future work. Finally, another interesting direction for future work is to extend DIM by allowing users to indicate the desired popularity for an image and constraints on acceptable image updates (e.g., software that cannot be removed from an image, or its minimum/maximum acceptable size).

Acknowledgments. Work partially 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. N. S. Altman. An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician*, 46(3):175–185, 1992.
2. 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.
3. H. Borges et al. Understanding the factors that impact the popularity of github repositories. In *ICSME*, pages 334–344. IEEE, 2016.
4. H. Borges, A. Hora, and M. T. Valente. Predicting the popularity of github repositories. In *PROMISE*, page 9. ACM, 2016.
5. L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen. *Classification and regression trees.* CRC press, 1984.
6. A. Brogi, D. Neri, and J. Soldani. DockerFinder: Multi-attribute search of docker images. In *IC2E*, pages 273–278. IEEE, 2017.
7. E. Franck and S. Nüesch. Mechanisms of superstar formation in german soccer: Empirical evidence. *European Sport Management Quarterly*, 8(2):145–164, 2008.
8. B. Goodman and S. Flaxman. Eu regulations on algorithmic decision-making and a right to explanation. In *ICML*, 2016.

9. R. Guidotti, S. J. N. Davide, and B. Antonio. Explaining successful docker images using pattern mining analysis. In *STAF*. Springer, 2018.
10. R. Guidotti, A. Monreale, F. Turini, D. Pedreschi, and F. Giannotti. A survey of methods for explaining black box models. *arXiv preprint arXiv:1802.01933*, 2018.
11. J. M. Harackiewicz et al. Predicting success in college. *JEP*, 94(3):562, 2002.
12. A. Hars and S. Ou. Working for free? - Motivations of participating in open source projects. *IJEC*, 6(3):25–39, 2002.
13. A. Joy. Performance comparison between linux containers and virtual machines. In *ICACEA*, pages 342–346, March 2015.
14. E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, and D. Damian. The promises and perils of mining github. In *MSR*, pages 92–101. ACM, 2014.
15. E. L. Lehmann and G. Casella. *Theory of point estimation*. Springer Science & Business Media, 2006.
16. B. R. Litman. Predicting success of theatrical movies: An empirical study. *The Journal of Popular Culture*, 16(4):159–175, 1983.
17. Z. Ma, A. Sun, and G. Cong. On predicting the popularity of newly emerging hashtags in twitter. *JASIST*, 64(7):1399–1410, 2013.
18. I. Miell and A. H. Sayers. *Docker in Practice*. Manning Publications Co., 2016.
19. L. Milli, A. Monreale, G. Rossetti, F. Giannotti, D. Pedreschi, and F. Sebastiani. Quantification trees. In *ICDM*, pages 528–536. IEEE, 2013.
20. 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]*.
21. L. Pappalardo and P. Cintia. Quantifying the relation between performance and success in soccer. *Advances in Complex Systems*, page 1750014, 2017.
22. L. Pappalardo, P. Cintia, D. Pedreschi, F. Giannotti, and A.-L. Barabasi. Human perception of performance. *arXiv preprint arXiv:1712.02224*, 2017.
23. 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.
24. 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.
25. L. Pollacci, R. Guidotti, et al. The fractal dimension of music: Geography, popularity and sentiment analysis. In *GOODTECHS*, pages 183–194. Springer, 2017.
26. P. Resnick and H. R. Varian. Recommender systems. *CACM*, 40(3):56–58, 1997.
27. M. V. Shcherbakov et al. A survey of forecast error measures. *World Applied Sciences Journal*, 24:171–176, 2013.
28. R. Sinatra, D. Wang, P. Deville, C. Song, and A.-L. Barabási. Quantifying the evolution of individual scientific impact. *Science*, 354(6312):aaf5239, 2016.
29. S. Soltesz et al. Container-based operating system virtualization: A scalable, high-performance alternative to hypervisors. *SIGOPS*, 41(3):275–287, 2007.
30. P.-N. Tan et al. *Introduction to data mining*. Pearson Education India, 2006.
31. R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288, 1996.
32. A. Tikhonov. Solution of incorrectly formulated problems and the regularization method. *Soviet Meth. Dokl.*, 4:1035–1038, 1963.
33. T. Trzciński and P. Rokita. Predicting popularity of online videos using support vector regression. *IEEE Transactions on Multimedia*, 19(11):2561–2570, 2017.
34. D. Wang, C. Song, and A.-L. Barabási. Quantifying long-term scientific impact. *Science*, 342(6154):127–132, 2013.
35. C. J. Willmott and K. Matsuura. Advantages of the mean absolute error (mae) over the root mean square error (rmse). *Climate research*, 30(1):79–82, 2005.
36. X. Yan and X. Su. *Linear regression analysis: theory and computing*. WS, 2009.