Time-aware API Popularity Prediction via Heterogeneous Features

Yao Wan, Liang Chen, Jian Wu
College of Computer Science & Technology
Zhejiang University
Hangzhou, China
{wanyao, cliang, wujian2000}@zju.edu.cn

Qi Yu
College of Computing & Information Sciences
Rochester Institute of Technology
Rochester, USA
qi.yu@rit.edu

***Abstract*—Application Programming Interfaces (APIs)[1], which are emerging web services in general, are increasing with a rapid speed in recent years. With so many APIs, many management platforms have been developed and deployed, leading to the boom of API markets, that are similar to the mobile App markets. Meanwhile, it has become more and more difficult to select and manage APIs. In reality, most existing management platforms typically recommend currently popular APIs to developers[2]. However, the fact that popularity of API varies over time is ignored in those platforms, leading to the difficulty of recommending APIs that are just released but may be popular in the near future. To tackle this challenge, an approach of predicting the popularity of APIs is proposed in this paper. Predicting the popularity of API can not only be used for API ranking, recommendation and selection, but also make it more convenient for API providers and consumers to manage or select API respectively. In this paper, we propose a time-aware linear model to predict the API popularity, using time series feature of APIs and API's self-features such as its' provider ranking and description features, which are called heterogeneous features in our paper. Comprehensive experiments have been conducted on a real-world ProgrammableWeb dataset with 613 real APIs. The experimental results show that our model has a better performance, when compared with some other state-of-the-art prediction models.**

I. Introduction

Web services are programmable modules with standard interface descriptions that provide universal accessibility through standard communication protocols [1]. While many prominent Web service providers such as Google, Youtube and Facebook, are also opted to externalize their business interaction throuth APIs. To some extent, APIs are a kind of RESTful Web services. Besides, APIs can be invoked over the Internet using standard protocols, which will facilitate and accelerate the Web to be programmable. On this programmable Web, developers can easily create their mashups by combining various APIs to solve all types of problems. Currently, APIs and mashups are the primary manifestation of the programmable Web.

Compared with conversional Web services, APIs are lightweight and easier to use. Besides, an API architecture attempts to create one-to-many, reusable interfaces. This is different with SOA which focuses primarily on limited number of tight integrations for specific use-cases [2]. The number of APIs and mashups is increasing since 2005 and this trend has been dramatically accelerated over the past few years. According to ProgrammableWeb.com[3], until November 2014, the number of APIs and mashups have reached to 10634 and 6049, respectively.

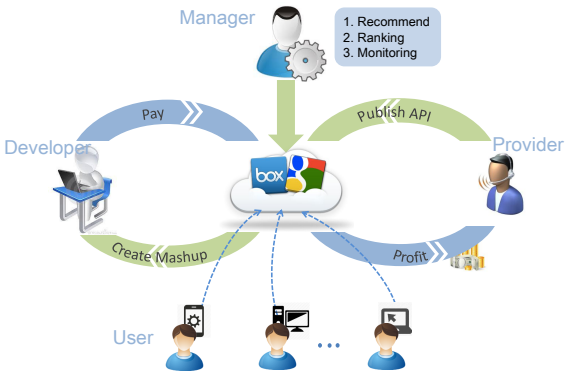


Fig. 1: Framework of an API market

Besides making the Web programmable, APIs also have great commercial values. They can provide uniform data and transaction interfaces to internal and external developers or customers, for the sake of improving data access and transactions. And mobile computing in particular are pushing increasing amounts of economic transactions from web browsers to API-driven interactions, although both of which continue to grow. Now, more and more big companies are joining the API Economy. Take eBay, an online auction and shopping website in which people can buy a broad variety of goods and services worldwide, as an example. It provides mobile, web and other client interfaces as an API, enabling transactions to be conducted from anywhere. Consumers can buy and pay their products not only on browers but also on mobiles. It is said that eBay has a vast ecosystem of power-seller and re-seller APIs that drives over 60% of its listings.

[1]In general, API includes local API and web API. In our paper, only web API is considered. If not specially declared, web API is called API in short.

[2]In our paper, the popularity of API is defined as the number of mashups that consist of the API (or are built on the API).

[3]http://www.programmableweb.com/

Another example is that Twitter has 13 billion calls through its APIs per day in 2012.

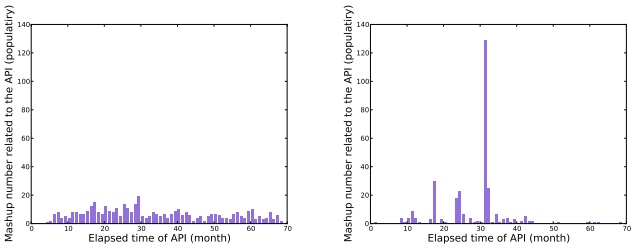
Recently, more and more Web sites dedicated to API market are emerging, such as Mashape[4]. Figure 1 describes the framework of such platform. There are mainly four roles in the API market. The API market is a repositoriy of APIs. This platform can be deployed on the cloud. API providers publish their own APIs on the cloud and they can get some profits if they want. Application developers which are also called consumers can invoke the APIs from the cloud, and combine them to build their applications or mashups. The developers also need to pay for using an API. Client users then can invoke the API conveniently from all kinds of devices, such as smart phones and personal computers. The manager is the administrator that manages the whole platform. They can recommend the most popular APIs to a special client, set a rule to rank all the APIs, and monitor the trend of APIs, and so on.

With the growing number of APIs, the need to search and manage APIs becomes more and more urgent. From our observation, those current API platforms such as ProgrammableWeb and Mashape manage APIs just through comparing the number of mashups which use the API at a time, or the number of followers. In essence, APIs used by more mashups or followered by more users are regarded as more popular. In our paper, the number of mashups that use a specific API is called the popularity of that API. One major drawback of those methods is that the time series factor is not considered. For traditional service-oriented computing, most existing service management approaches adopt some ineffective features of service, such as failure rate and service popularity to measure the quality of service, and recommend service according to the QoS. Most works predict the QoS purely based on the historical records, but few take into account the time. However, the QoS of web services or the popularity of API varies in time.

Predicting the popularity of APIs will benefit API ranking, recommendation and selection. It can also help API providers more effectively manage APIs. For example, providers can give a higher value to an API which is recently released, but have a tendency to be of high popularity in the near future. This will enable users to select an appropriate API which is newly released, although the popularity is still zero.

There are have been recent efforts towards building models to predict the popularity using various techniques. Szabo and Huberman(S-H) [3] observed that the feature popularity of a given content is strongly correlated with its early popularity on a reference date. But this model completely ignores the influence of specific information of the content itself. For example, two different pieces of content may have very similar popularity at the same reference date and yet exhibit distinct popularity behaviors thereafter. In our context, two real APIs' popularity trend are shown in Figure 2. After 40 months since uploaded, the Youtube API was used by 288 mashups, while

[4]https://www.mashape.com/



(a) Popularity of Youtube API (b) Popularity of Twilio API

Fig. 2: Popularity of Youtube and Twilio in the lifecycle

the Twillo API was used by 292 mashups, the popularity of these two API are similar at the refence date. Thus the S-H model would predict that both APIs would have similar popularity on the 69-th month since upload. Yet they end up with very different total popularity: the Youtube API is used by 442 mashups, whereas, the popularity of Twilio API is 307 in 69-th month. This is because the patterns of these two APIs are quite distinct.

In this paper, we take the time factor into account and propose a time-aware model to predict the popularity of API, as the popularity is varied from time to time. As soon as an API is released, the popularity of this API is zero. But with time goes on, the popularity will usually increase. Besides, we think specific information of API itself has a great effect on the pattern of its popularity. So we analyze the effect of specific information of API, especially the text features of API. In our paper, the time features, API's numerical features and textual features are called heterogeneous features. We propose a linear regression model to integrate the heterogeneous features together.

We evaluate our model by comparing it against the S-H model, on three datasets from ProgrammableWeb. The main contributions of this paper are summarized as follows:

- To our best knowledge, this is the first paper to model and tackle the problem of API popularity prediction.
- A linear prediction model integrating heterogeneous features is proposed in our paper to predict the popularity of API. Specifically, we analyze the information of API and find out many factors that have an effect on the popularity of API.
- Comprehensive experiments show that our model has a better performance compared with other state-of-the-art methods.

The remainder of this paper is organized as follows. Section II introduces the related work of this paper. Section III describes the datasets we will use in our experiment. Two classical models and our proposed model are presented in Section IV. Section V shows the experimental results and analysis. Finally, we conclude this paper and give some future directions in Section VI.

II. Related Work

In the realm of service computing, a number of literatures are focusing on issuses such as service discovery, service selection, service recommendation and service composition, etc. [4], [5], [6], [7], [8], [9], [10]. One work which is related to our's is [11], in which the authors proposed a time-aware personalized QoS prediction framework for Web services using tensor factorization. And in [12], time information is integrated into the similarity measurement and the QoS prediction, and a hybrid personalized random walk algorithm is employed to handle data sparsity.

With the emmergence of APIs, many focuses are transfering from WSDL-based Web services to APIs. [13] and [14] proposed a framework (or model) for the discovery of APIs. [15] provided a linked data perspective of Web APIs to enhance effective multi-perspective Web APIs search for fast development of web mashups. [16] and [17] proposed an approach for ranking the APIs. Services that are composed by APIs are called Mashup, and more and more literatures are focusing on Mashup. Predicting the popularity of API can also help developers select appropriate APIs to compose their Mashup. Cao et al. [18] proposed an approach to recommend Mashup services to users based on user interest and social network of services. Huang et al. [19] have had an empirical study of ProgrammableWeb, and they are building a service ecosystem based on Web APIs and mashups. In [20] the authors presented a method to extract service evolution patterns for Mashup creation by exploiting Latent Dirchlet Allocation (LDA) and time series prediction. And they proposed an innovative three-phase network prediction approach (NPA) for service recommendation based on network in [21].

To the best of our knowldge, few works take the time series into account to predict the popularity of API currently. Most service selection and recommendation are based on QoS. However, large-scale real-world Web service QoS datasets available for studying the prediction accuracy are difficult to collect. Besides, in reality the QoS of service varies from time to time according the invocation time, network condition of users, etc. In this paper we focus on predicting the popularity of API, for service (API) ranking and selection.

There are have been several efforts towards predicting the popularity of online content. For example, Szabo and Huberman [3] observed a linear correlation between the early and future popularity of online content, they use the popularity at a reference time to predict the future popularity. [22] proposed a model based on reservoir computing to predict the near future popularity of videos based on popularity data from the previous days. However their approach can be affected by randomization effects. [23] proposed a linear model to predict the popularity of YouTube Videos, but the information of YouTube itself is not considered. [24] used Cox proportional hazard regression model to predict the popularity of online content. This paper is rooted in survival analysis and inferring the likelihook with which the content will be popular.

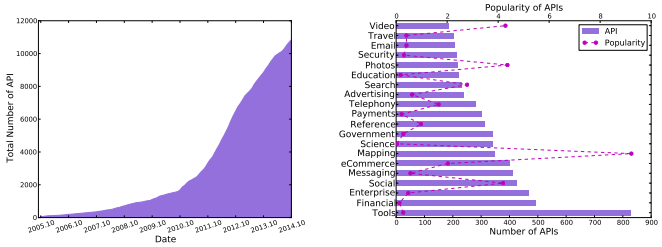
III. Dataset Characteristics

A. Dataset

ProgrammableWeb is one of the most popular platforms that has collected lots of APIs and mashups used in Web and mobile applications. It records the daily evolution of the global API economy. We crawled all the APIs and mashups until Nov. 2014 from ProgrammableWeb. We first desscribe the dataset and provide some statistical analysis.

TABLE I: An Overview of ProgrammableWeb Data

Number of APIs	10634
Number of mashups	6049
Number of users	52511



(a) Growth trend of API (b) Top-20 API categories

Fig. 3: (a) The growth trend of API since Nov. 2005 (b) The number and average popularity of Top-20 API categories

Table I is an overview of the dataset. Althouth the number of APIs and mashups are both large, our analysis reveals that most of the APIs in ProgrammableWeb are not used in any mashups. So the popularity of these APIs are always zero. In our experiment, we will filter out these APIs.

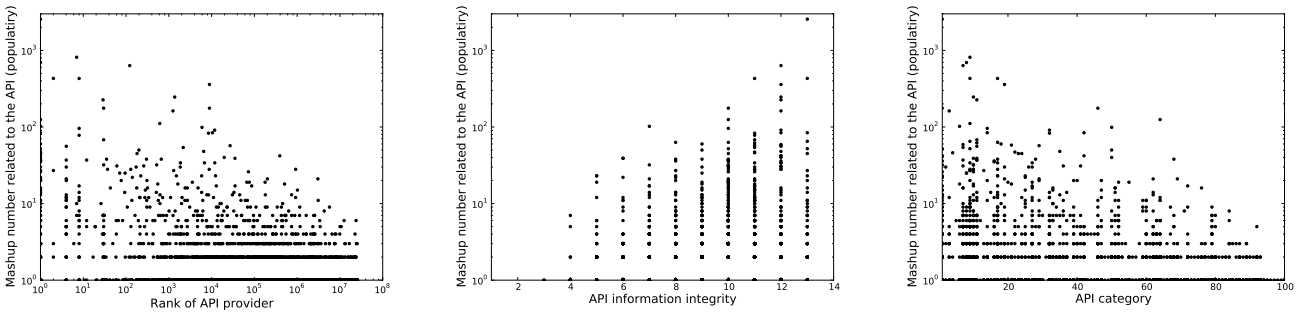
Figure 3(a) describes the trend of API growth, we can see that most APIs are created in the last 4 years. In 2005 there was just one API (Google Maps) in ProgrammableWeb, but until 2014, the number has reached to more than 10000, and it is increasing dramatically.

Figure 3(b) shows the number and average popularity of API in different categories. From this figure, we can observe that API in categories “Mapping”, “Photos” and “Video” have a higher popularity than others.

B. Statistical Analysis

In this section, we will analyze the features of API that may have an effect on its popularity. From our analysis, we find at least four factors are related to the popularity of API. These are provider rank, the information integrity, the category and the followers number of API. The scatter plots of those factors are shown in Figure 4.

From Figure 4(a), we can find that the lower the rank of provider, the lower of its popularity. In our paper, the provider rank is attained from the website *Alexa.com*, which provides commercial web traffic data and the rank of many websites. The rank of the API provider's website can represent the quality of provider to some extent. Before an API is released,



(a) The effect of API's provider rank (b) The effect of API's information integrity (c) The effect of API category

Fig. 4: The scatter plot of some factors against the popularity of APIs. (a) Relationship between API's provider rank and the popularity of API. (b) Popularity of API plotted against API's information integrity. (c) The effect of API category to API's popularity, the popularity is sorted by the average popularity of each category.

the provider is required to fill in some relevant information of that API such as it's category, description, homepage, and so on. In our paper, API information integrity means the degree of details providers describe the API. For example, for Twilio API, the provider fills in 14 relevant informations in ProgrammableWeb, so the information integrity of Twilio API is 14. Figure 4(b) shows that the API with a higher information integrity may have a higher popularity. That is, the provider of API that with a higher information integrity is more prudent to release the API, the provider and the API itself may be of higher quality. Figure 4(c) gives a hint that the category of API may also has an effect on it's popularity.

IV. Popularity Prediction Models

A. Evaluation Metric

As suggested in [23], we use relative squared error (RSE) to evaluate the models in our paper. Here we not use other evaluation metrics for the reason that, the popularity of API varies greaty from one to other. The relative error are more relevant than the absolute error. Besides, square always gives a positive value, and emphasizes on larger differences. This is also suitable for our popularity prediction context.

Let $N(a)$ be the total number of mashups that have used API a, namely the popularity of API a. Let $N(a|t)$ be the total number of mashups that has used API a from the first t months. As shown in Figure 2(a), after 40 months sinces uploaded, the Youtube API was used by 288 mashups. It can be represented as $N(Youtube|t = 40) = 288$. Then the RSE of API a's prediction can be defined as this form:

$$RSE = \left(\frac{\hat{N}(a, t_t|t_r)}{N(a, t_t)} - 1 \right)^2 \tag{1}$$

If we denote the collection of APIs by C, the mean relative squared error(mRSE) can be defined as this formular:

$$mRSE = \frac{1}{|C|} \sum_{a \in C} \left(\frac{\hat{N}(a, t_t|t_r)}{N(a, t_t)} - 1 \right)^2 \tag{2}$$

We use the mRSE to evaluate our model, comparing with the S-H and LR model. The less the mRSE is, the better performance the model will achieve.

B. Szabo-Huberman (S-H) Model

After performing a logarithmic trasformation on the popularity of instances, Szabo and Huberman [3] found a strong linear correlations between the early and future popularity, with a normally distributed noise. This means that the more pupular instances are at the beginning, the more popular they will also be in the future. In our context, the connection between early and future popularity can be described as:

$$ln\hat{N}(a, t_t|t_r) = ln\left[r(t_r, t_t)N(a, t_r)\right] + \xi(t_r, t_t) \tag{3}$$

where $\hat{N}(a, t_t|t_r)$ is the predicted popularity of API at target time t_t based on the popularity at reference time t_r, $r(t_r, t_t)$ accounts for the linear relationship between the log-transformed popularity at different times, and ξ is a noise term that accounts for the natural variances in individual content dynamics beyond the expected trend in the model and is drawn from a fixed distribution with mean zero.

Briefly, the future popularity of an API can be expressed as:

$$\hat{N}(a, t_t|t_r) = \alpha_{t_r,t_t} \cdot N(a, t_r) \tag{4}$$

where, α_{t_r,t_t} is independent of the API a. And $\alpha_{t_r,t_t} = r(t_r, t_t)e^{\xi(r_f,r_t)}$. This means that the future popularity of API a is just related to its early one by a constant factor.

When given the reference date t and the training set C of videos, combining Eq. (2) and Eq. (4), we can get the value of α_{t_r,t_t}:

$$\alpha_{t_r,t_t} = \frac{\sum_{a \in C} \frac{N(a,t_r))}{N(a,t_t)}}{\sum_{a \in C} \left(\frac{N(a,t_r))}{N(a,t_t)} \right)^2} \tag{5}$$

When the α_{t_r,t_t} is known, according to Equation 4, we can easily get the popularity of API a in future, just multiplying the popularity in the reference date $N(a, t_r)$ by α_{t_r,t_t}.

The computaional complexity of this model is $O(n)$ with n training data. So this model is simple and has a perfect scalability.

C. Linear Regression (LR) Model

The S-H model just uses the popularity of API in a specific date (t_r) to predict the future popularity. In other words, it just utilizes only one piece of time series of API. In our dataset, we can get the delta of API popularity at each time series. So, here we try to predict the popularity of API at t_t as a linear function of these popularity deltas at each month.

Specifically, let $x_i(a)$ be the number of mashups that use API a on the i-th month since its released ($x_i(a) = N(a,i) - N(a,i-1)$). The feature vector of API is defined as:

$$X_{t_r}(a) = (x_1(a), x_2(a), \cdots, x_{t_r}(a))^T \tag{6}$$

and the prediction value of popularity of API a at t_t can be expressed as:

$$\hat{N}(v, t_t | t_t) = \Theta_{(t_r,t_t)} \cdot X_{t_r}(v) \tag{7}$$

where $\Theta_{(t_r,t_t)} = (\theta_1, \theta_2, \cdot, \theta_{t_r})$ presents the vector of model parameters and depends only on t_r and t_t. Given a traing set C, t_r, t_t, we can define our oprimization problem as:

$$\underset{\Theta_{(t_r,t_t)}}{argmin} \frac{1}{|C|} \sum_{a \in C} \left(\frac{\Theta_{(t_r,t_t)} \cdot X_{t_r}(a)}{N(a,t_t)} - 1 \right)^2 \tag{8}$$

Let $X_v^* = \frac{X_{tr}(a)}{N(a,t_t)}$, we can express the optimization problem as:

$$\underset{\Theta_{(t_r,t_t)}}{argmin} \frac{1}{|C|} \sum_{a \in C} \left(\Theta_{(t_r,t_t)} \cdot X_v^* - 1 \right)^2 \tag{9}$$

which is an ordinary least squares (OLS) problem that can be solved via a singular value decomposition. The computional complexity of this optimization problem is $O(np^2)$, where n is the number of training examples and p the number of model parameters.

From another aspect, S-H model uses the popularity at t_r, represented by $N(t_r)$, while LR model uses a series of API's delta popularity at each month since released date to the reference date, represented by $(x_1(a), x_2(a), \cdots, x_{t_r}(a))$. Here $x_1(a) + x_2(a) + \cdots + x_{t_r}(a) = N(t_r)$. So the S-H model is a special case of this LR model, with the added restriction that $\theta_1 = \theta_2 = \cdots = \theta_{t_r}$.

D. LR-HF Model

The S-H model utilizes one piece of time series of API, while the LR model utilizes many time series of API until the reference date t_r. Both of these two models just use the time features of API, but ignore other features. Form the analysis of Section III, we find other features of API besides time features have an effect on the popularity.

Our LR-HF model utilizes all the features of API that may affect it's popularity. Since the features are from different aspects and have different data structures, we call them heterogeneous features. The heterogeneous features can be mainly devided into four types:

- **Time features:** These features are the same as features used in LR model. We devided the time from API's released time to the reference date t_r into many interval by month. The delta popularity in each interval can be a feature.
- **Numerical features:** The provider rank and the information integrity of API are both numerical features.
- **Categorical features:** Each API in our dataset belongs to a specific category, such as mapping, tools, or eCommerce.
- **Textual features:** Each API has a detailed description, which belongs to textual features.

After inducing those heterogeneous features, and refering to Eq.(8), the optimization problem should be transformed into this form:

$$\underset{\Theta_{(t_r,t_t)}}{argmin} \frac{1}{|C|} \sum_{a \in C} \left(\Theta_{(t_r,t_t)} \cdot X_v^* - 1 \right)^2 + \lambda \left\| \Theta_{(t_r,t_t)} \right\|_2^2 \tag{10}$$

where λ is the penalty coefficient, which controls the degree of penalty, $X_v^* = \frac{X_{tr}(a)}{N(a,t_t)}$ and $\left\| \Theta_{(t_r,t_t)} \right\|_2^2$ is the regularization term to prevent overfitting, the regression is also called ridge regression.

V. Experiment

A. Setup

In our experiment, the target date t_t varies from 24 months to 48 months. So we extract the APIs whose releasing date is between 2005 and Nov. 2010, to ensure that each API has a span of 4 years' time series. Besides, we find the popularity of some APIs in our dataset is always zero, such APIs are beyond our consideration. Lastly, we get a dataset of 613 APIs.

To make our experiment more convincing, we split our dataset into two subsets, the popular dataset and the junk dataset.

- **Popular:** We define that the APIs whose popularity are greater than 5 until Nov. 2014 belong to popular dataset.
- **Junk:** On the other hand, those APIs whose popularity are less than 5 until Nov. 2014 belong to the junk dataset. We assume that the qualities of those APIs used by only few mashups are low, and the trend of those APIs' popularity are nearly smooth.

Our experiment is conducted on the basis of the full dataset, as well as the popular and junk dataset, then we have a comparison on those three datasets.

B. Feature Extraction and Selection

We can denote the feature vector as $\mathbf{x}^T = (\mathbf{x}_n^T, \mathbf{x}_c^T, \mathbf{x}_t^T)$, where $\mathbf{x}_n^T$ represents the numerical features, $\mathbf{x}_c^T$ the categorical features, and $\mathbf{x}_t^T$ the textual features. The numerical features includes information integrity, provider rank, and the popularity at each time interval. It is easy to introduce the numerical features in our model without any specific transformation, except for a normalization. But the categorical and textual features need to be transformed to numerical features. In our dataset, the categorical features consist of one or two words, we represent the categorical features by binary code.

TABLE II: Model prediction errors (mRSE) with various t_r and t_t on the full dataset.

mRSE	Methods	$t_r = 1$	$t_r = 3$	$t_r = 5$	$t_r = 7$	$t_r = 9$	$t_r = 11$	$t_r = 13$
$t_t = 24$	S-H	0.3987	0.3236	0.2750	0.2283	0.1867	0.1525	0.1183
	LR	0.3944	0.3240	0.2783	0.2320	0.1894	0.1540	0.1195
	LR-HF1	0.3121	0.2684	0.2364	0.2049	0.1698	0.1448	0.1166
	LR-HF2	**0.2151**	**0.2010**	**0.1886**	**0.1732**	**0.1523**	**0.1326**	**0.1104**
$t_t = 36$	S-H	0.4483	0.3824	0.3404	0.2933	0.2537	0.2214	0.1919
	LR	0.4472	0.3838	0.3434	0.2976	0.2569	0.2217	0.1931
	LR-HF1	0.3792	0.3347	0.3017	0.2679	0.2313	0.2081	0.1875
	LR-HF2	**0.2470**	**0.2366**	**0.2228**	**0.2104**	**0.1933**	**0.1756**	**0.1605**
$t_t = 48$	S-H	0.4577	0.3990	0.3556	0.3153	0.2785	0.2494	0.2218
	LR	0.4584	0.4005	0.3564	0.3186	0.2835	0.2527	0.2265
	LR-HF1	0.3912	0.3553	0.3195	0.2905	0.2588	0.2377	0.2179
	LR-HF2	**0.2607**	**0.2507**	**0.2350**	**0.2241**	**0.2123**	**0.1973**	**0.1858**

The description of API in our dataset consits of a bag of words that describe the functionality. Before we apply it to our model, several nature language preprocessing tasks should be done.

1) **Case-folding and tokenization.** Firstly, we reduce all letters of API's description to lower case, and then split the description into a bag-of-words using the tokenizer, for instance by using white-spaces.
2) **Pruning.** Then we filter the stopwords that are meaningless for representing the service, such as *is, very, should, etc*. Besides, we just keep the nouns and adjectives using a part-of-speech tagger.
3) **Stemming.** In the third step, we strip the word to obtain the stem word. For example, *map, mapping, maps,*and *mapping* have the same stem word *map*.
4) **Spell Correcting.** Lastly, we find some words in descriptions that may be misspelled. So we correct these words such as *communicte, communicata* to be *communicate* based on edit distance.

After conducting those processes, we get a corpus of all the APIs' description. Before the corpus can be used in the LR-HF model, it need to be transformed into numerical vectors. In our paper, the tf-idf approach is adopted.

In our processing we find there are nearly 68 different categories and 2653 distinct words. We need to conduct a feature selection on those features. We select $K(K = 20)$ best features based on variance. Features that with small variance will be removed.

C. Performance Evaluation

In this section, we will show the experimental results of S-H, LR and LR-HF models. S-H is the baseline model and LR is a model based on S-H model, utilizing more time features. The experiment only adding the textual features is called LR-HF1, while the experiment with all the heterogeneous features is called LR-HF2. All the experiments are conducted on three datasets, and the target date t_t varies from 24, 36 to 48 months.

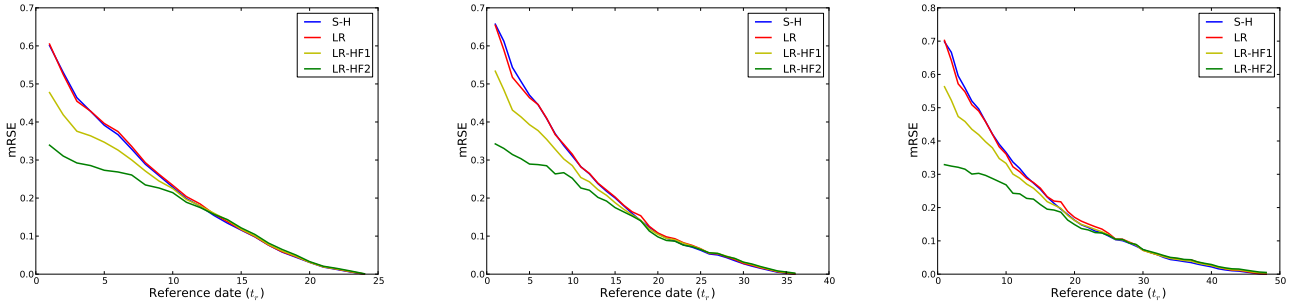
Table II shows the mRSE results of different prediction methods on prediction value. We set reference date t_r as 1, 5, 8 and 10. The target date t_t is setted as 24, 36 and 48 months. We set λ as 0.2. To prevent overfitting, each experiment was conducted using random permutations cross validation, which is a good alternative to KFold cross validation that allows a finer control on the number of iterations and the proportion of samples on each side of the train/test split. We first shuffle the dataset and then split it one by one, 50% of the dataset are used to training the parameter, and 25% of the dataset are used for cross validation, the rest are for testing.

The experimental results of Table II show that:

- Under all experimental settings, our LR-HF method always obtains a smaller mRSE values, which indicates better prediction accuracy. Especially, LR-HF2 has a smaller mRSE than LR-HF1. That means when introducing all heterogeneous features, we can get a better performance than just introducing the textual features.
- Our LR-HF model performs particularly well on the condition that t_r is small, the mRSE of LR-HF2 has reduced to nearly 0.2151. With the increase of t_r, the performance gap between our model and state-of-the-art models becomes smaller. While our model still has a better performance.
- For a specific model, with the increase of t_r, the prediction accuracy also achieves significant enhancement, since larger t_r provides more time-serial features for prediction.
- Fixing the value of t_r, we can find that for each method, the mRSE increases with the t_t becoming larger, since larger t_t means greater uncertainty of popularity.

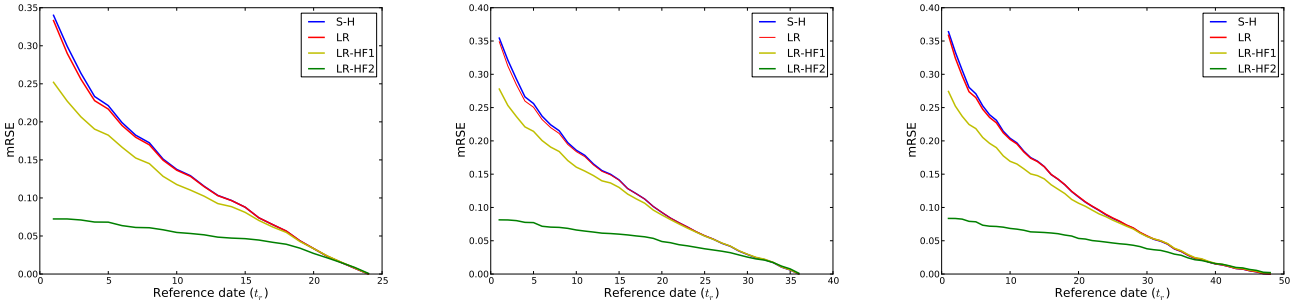
Figure 5 depicts the experimental results of four methods on popular dataset. Figure 5(a) shows the mRSE with $t_t = 24$, Figure 5(b) shows the mRSE with $t_t = 36$ and Figure 5(c) shows the mRSE with $t_t = 48$.

Observing from Figure 5, we can clearly find that our model have an obviously better performance over S-H and LR model, especially when the gap between reference time and target time is large. We can also find that the S-H model and LR model have a so similar performance that two lines even overlaps. Taking a closer look at it, we can find the LR model



(a) t_t=24 months (b) t_t=36 months (c) t_t=48 months

Fig. 5: Model prediction errors (mRSE) as functions of reference date t_r for various target dates t_t on popular dataset.



(a) t_t=24 months (b) t_t=36 months (c) t_t=48 months

Fig. 6: Model prediction errors (mRSE) as functions of reference date t_r for various target dates t_t on junk dataset.

may still have a better performance, that may be caused by our dataset, because the size of data that has a complete time-series feature is really small. Besides, we can find that with the reference time becoming closer to the target time, the mRSE descreases, for all the four models. In other words, it's easier to predict the future popularity if the future is not far away. What is more, with the reference time becoming closer to the target time, the advantages of our model over other models will shrink. For the reason that, it's enough for S-H or LR model to use the time features to predict the popularity.

Take Figure 5(a) for an example, the S-H model and LR model have the same mRSE on the condition that the reference date is one month. Because, if we only use the first month's popularity to predict the future popularity, the S-H and LR are same model, both model use only one piece of time series feature. When compared with the experimental results on full dataset, we can find the performance on popular dataset is a little poorer than that on the full dataset. For example, in Figure 5(a), given reference time is one month, the mRSE of our LR-HF2 model is nearly 0.38, while in Table II, the mRSE is only 0.22.

Figure 6 depicts the experimental results of four methods on junk dataset. Figure 6(a) shows the mRSE with $t_t = 24$, Figure 6(b) shows the mRSE with $t_t = 36$ and Figure 6(c) shows the mRSE with $t_t = 48$. From Figure 6, we can draw a conclusion that all the four methods have a conspicious improvement on

TABLE III: Impact of λ

mRSE	$\lambda = 0.0$	$\lambda = 0.2$	$\lambda = 0.4$	$\lambda = 0.6$	$\lambda = 0.8$	$\lambda = 1.0$
$t_r = 1$	0.25131	0.24748	**0.24746**	0.24776	0.24812	0.24846
$t_r = 5$	0.23292	**0.22910**	0.22921	0.22970	0.23026	0.23084
$t_r = 10$	0.19343	**0.18926**	0.18962	0.19081	0.19226	0.19379

different experiments and our LR-HF model still have an better performance, no matter we only introduce textual features or introduce all the heterogenous features. We can see that if we just introduce API's textual features, the improvement is not too big, when compared with that on full dataset and popular dataset. However, when all the heterogenous features are introduced, the improvement is very obvious. The reason that may cause this state is that the growth rate of junk APIs are quite slow. That means that the popularity on reference and one on the target date is almost the same, so prediction on this dataset is relatively easier. The experimental results on junk dataset proves that our model is particularly suitable for predicting the popularity of APIs that not changes frequently.

As shown in Figure 6(a), with the target date t_t is 24 months, the mRSE of our LR-HF2 model can be decreased to nearly 0.08 althouth only the first month's popularity is used. And other two model can also have a performance with mRSE 0.36. On this condition, our LR-HF2 model has a nearly 6 times of performance imporvement.

D. Impact of λ

With the parameters increase, it is more likely to overfit the linear model during the training stage. To prevent overfiting, we use a tuning parameter λ to control the penalty in our LR-HF model. If $\lambda = 0$, our model is a general linear model which is easy to be overfiting. If $\lambda \neq 0$, the linear model is called ridge regression.

To study the impact of parameter λ to our LR-HF2 method, we set the number of description features is 20, the target date t_t is 36, and we do the experiment on full dataset. We varies λ from 0 to 1.0 to analyze the impact of λ. Table III shows the impact of λ. From the table, we can find that the value of λ really has an impact on the prediction accuracy, and a suitable λ value will provide better prediction accuracy. On this condition, we can find that the most appropriate value of λ is between 0.2 and 0.4. The λ should not be too large or too small.

VI. Conclusion and Future Work

Predicting the popularity is of great importance in the management of API market. The more accurate the prediction is, the greater profits the provider and consumers will get. In this paper, we firstly analyze some factors that may have an effect on the popularity of APIs. We find that some features, including numerical features, categorical features and textural features can affect the popularity of API. By combining all the features, we propose an approach for predicting future popularity of APIs by linear regression model. We split our data into three dataset and do experiment on each of them. Comprehensive experiments on three datasets of real-world APIs in ProgrammableWeb show the effectiveness and feasibility of our method.

For future work, we plan to collect more time series records of APIs, and try to find more features that may have an effect on the popularity of API. We plan to discover the popularity trends of these APIs. Besides, we are going to conduct more research on inferring the impacts of other data sources, such as social media.

The dataset that our experiment based on will be released to the homepage of one of our authors[5].

Acknowledgment

This research was partially supported by the National Natural Science Foundation of China under grant of 61173176, Science and Technology Program of Zhejiang Province under grant of 2013C01073, National High-Tech Research and Development Plan of China under Grant No. 2013AA01A604.

References

[1] L.-J. Zhang, H. Cai, and J. Zhang, *Services computing*. Springer, 2007.

[2] 3scale, "Winning in the api economy," http://www.3scale.net/api-economy/ebooks/winning-in-the-api-economy/.

[3] G. Szabo and B. A. Huberman, "Predicting the popularity of online content," *Communications of the ACM*, vol. 53, no. 8, pp. 80–88, 2010.

[4] J. Wu, L. Chen, Z. Zheng, M. R. Lyu, and Z. Wu, "Clustering web services to facilitate service discovery," *Knowledge and information systems*, vol. 38, no. 1, pp. 207–229, 2014.

[5] G. Kang, J. Liu, M. Tang, X. Liu, and K. K. Fletcher, "Web service selection for resolving conflicting service requests," in *Web Services (ICWS), 2011 IEEE International Conference on*. IEEE, 2011, pp. 387–394.

[6] Z. Zheng, H. Ma, M. R. Lyu, and I. King, "Qos-aware web service recommendation by collaborative filtering," *Services Computing, IEEE Transactions on*, vol. 4, no. 2, pp. 140–152, 2011.

[7] F. Tao, L. Zhang, K. Lu, and D. Zhao, "Research on manufacturing grid resource service optimal-selection and composition framework," *Enterprise Information Systems*, vol. 6, no. 2, pp. 237–264, 2012.

[8] M. Li, J. Huai, and H. Guo, "An adaptive web services selection method based on the qos prediction mechanism," in *Web Intelligence and Intelligent Agent Technologies, 2009. WI-IAT'09. IEEE/WIC/ACM International Joint Conferences on*, vol. 1. IET, 2009, pp. 395–402.

[9] Y. Shen, J. Zhu, X. Wang, L. Cai, X. Yang, and B. Zhou, "Geographic location-based network-aware qos prediction for service composition," in *Web Services (ICWS), 2013 IEEE 20th International Conference on*. IEEE, 2013, pp. 66–74.

[10] X. Chen, Z. Zheng, Q. Yu, and M. R. Lyu, "Web service recommendation via exploiting location and qos information," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 25, no. 7, pp. 1913–1924, 2014.

[11] Y. Zhang, Z. Zheng, and M. R. Lyu, "Wspred: A time-aware personalized qos prediction framework for web services," in *Software Reliability Engineering (ISSRE), 2011 IEEE 22nd International Symposium on*. IEEE, 2011, pp. 210–219.

[12] Y. Hu, Q. Peng, and X. Hu, "A time-aware and data sparsity tolerant approach for web service recommendation," in *Web Services (ICWS), 2014 IEEE International Conference on*. IEEE, 2014, pp. 33–40.

[13] D. Bianchini, V. De Antonellis, and M. Melchiori, "A multi-perspective framework for web api search in enterprise mashup design," in *Advanced Information Systems Engineering*. Springer, 2013, pp. 353–368.

[14] R. Torres, B. Tapia, and H. Astudillo, "Improving web api discovery by leveraging social information," in *Web Services (ICWS), 2011 IEEE International Conference on*. IEEE, 2011, pp. 744–745.

[15] D. Bianchini, V. De Antonellis, and M. Melchiori, "A linked data perspective for effective exploration of web apis repositories," in *Web Engineering*. Springer, 2013, pp. 506–509.

[16] K. Gomadam, A. Ranabahu, M. Nagarajan, A. P. Sheth, and K. Verma, "A faceted classification based approach to search and rank web apis," in *Web Services, 2008. ICWS'08. IEEE International Conference on*. IEEE, 2008, pp. 177–184.

[17] D. Bianchini, V. De Antonellis, and M. Melchiori, "Model-based search and ranking of web apis across multiple repositories," in *Web Information Systems Engineering–WISE 2014*. Springer, 2014, pp. 218–233.

[18] B. Cao, J. Liu, M. Tang, Z. Zheng, and G. Wang, "Mashup service recommendation based on user interest and social network," in *Web Services (ICWS), 2013 IEEE 20th International Conference on*. IEEE, 2013, pp. 99–106.

[19] K. Huang, Y. Fan, and W. Tan, "An empirical study of programmable web: a network analysis on a service-mashup system," in *Web Services (ICWS), 2012 IEEE 19th International Conference on*. IEEE, 2012, pp. 552–559.

[20] Y. Zhong, Y. Fan, K. Huang, W. Tan, and J. Zhang, "Time-aware service recommendation for mashup creation in an evolving service ecosystem," in *Web Services (ICWS), 2014 IEEE International Conference on*. IEEE, 2014, pp. 25–32.

[21] K. Huang, Y. Fan, and W. Tan, "Recommendation in an evolving service ecosystem based on network prediction."

[22] T. Wu, M. Timmers, D. D. Vleeschauwer, and W. V. Leekwijck, "On the use of reservoir computing in popularity prediction," in *Evolving Internet (INTERNET), 2010 Second International Conference on*. IEEE, 2010, pp. 19–24.

[23] H. Pinto, J. M. Almeida, and M. A. Gonçalves, "Using early view patterns to predict the popularity of youtube videos," in *Proceedings of the sixth ACM international conference on Web search and data mining*. ACM, 2013, pp. 365–374.

[24] J. G. Lee, S. Moon, and K. Salamatian, "Modeling and predicting the popularity of online contents with cox proportional hazard regression model," *Neurocomputing*, vol. 76, no. 1, pp. 134–145, 2012.

[5] http://www.zjujason.com