Expert Systems With Applications 149 (2020) 113231

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

A Bayesian learning model for design-phase service mashup popularity prediction

Moayad Alshangiti*, Weishi Shi, Xumin Liu, Qi Yu

Golisano College of Computing and Information Science, Rochester Institute of Technology, Rochester, NY 14623, USA

ARTICLE INFO

Article history:
Received 12 September 2018
Revised 6 January 2020
Accepted 21 January 2020
Available online 21 January 2020

Keywords:
Popularity prediction
Bayesian learning
Service mashup

ABSTRACT

Using web services as building blocks to develop software applications, i.e., service mashups, not only reuses software development efforts to minimize development cost, but also leverages user groups and marketing efforts of those services to attract users and improve profits. This has significantly encouraged the development of a large number of service mashups in various domains. However, using existing services, even popular ones, does not guarantee the success of a mashup. In fact, a large portion of existing mashups fail to attract a good number of users, making the mashup development effort less effective. Design-phase popularity prediction can help avoid unpromising mashup developments by providing early-on insight into the potential popularity of a mashup. In this paper, we investigate the factors that can affect the popularity of a mashup through a comprehensive analysis on one of the largest mashup repository (i.e., ProgrammableWeb). We further propose a novel Bayesian approach that offers early-on insight to developers into the potential popularity of a mashup using design-phase features only. Besides identifying those relevant features, the Bayesian learning model can provide a confidence level for each prediction. This provides useful guidance to developers for successful mashup development. Experimental results demonstrate that the proposed approach achieves high prediction accuracy and outperforms competitive models.

© 2020 Elsevier Ltd. All rights reserved.

1. Introduction

Web service technologies have been widely acknowledged as the key enabler for boosting software reuse and loosely-coupled distributed application development (Yu, Liu, Bouguettaya, & Medjahed, 2008). Driven by their benefits, the number and variety of web services have continuously increased and in a rapid pace. Meanwhile, web services make significant contributions to mobile app development. Through the invocation of web services, the functionality of mobile apps can be outsourced to relevant web services hosted on remote servers. This can effectively address the hardware and software restrictions of mobile devices. Therefore, the recent enormous success of mobile apps has further fueled the development of web services. A recent study shows that there are more than 17,000 web services in hundreds of domains registered in ProgrammableWeb [1], one of the biggest public web service repository.

The significant number and variety of web services have greatly encouraged the usage of web services as the building blocks to develop software applications, especially web and mobile applications, i.e., generating *service mashups* (Benslimane, Dustdar, & Sheth, 2008). A service mashup can benefit from the services it is built upon in many aspects. First, reusing the development efforts of those web services can minimize the cost of developing the mashup. Second, a mashup can deliver its functionality by leveraging the service providers' resources including information (e.g., traffic data from a mapping service), CPU cycles for computation or data processing, and data storage.

Unfortunately, the success of a service, in terms of attracting users, cannot be directly translated to the success of the mashups that it participates in. Through a comprehensive analysis on the service mashup data collected from ProgrammableWeb, we found that around 75% of service mashups fail to attract a good number of users. *One key motivation behind our work is to help avoid unpromising service mashup developments by providing early-on insight into the potential popularity of a mashup.* To do so, it is important to predict the popularity of a service mashup before it's developed.

Current research efforts on web service popularity can be classified into two categories: leveraging popularity for service

* Corresponding author.
E-mail addresses: mma4247@rit.edu, mshangiti@uj.edu.sa (M. Alshangiti), ws7586@rit.edu (W. Shi), xmlics@rit.edu (X. Liu), qi.yu@rit.edu (Q. Yu).

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

https://doi.org/10.1016/j.eswa.2020.113231
0957-4174/© 2020 Elsevier Ltd. All rights reserved.

selection or recommendation (Hora & Valente, 2015; Hou & Pletcher, 2010; Jain, Liu, & Yu, 2015; Mileva, Dallmeier, & Zeller, 2010) and web service popularity prediction (Wan, Chen, Wu, & Yu, 2015). In (Wan et al., 2015), service popularity is measured as the number of mashups using the service. It is predicted based on the historical usage of the service, i.e., using the past popularity to predict the future one. Thus, it requires that a service is developed and released to the public for a given time before a prediction can be made. To our best knowledge, there is no systematic approach that can be used to predict the popularity of a mashup during the design time, i.e., even before it has been developed and used.

In this paper, we propose a novel approach that provides early-on insight into the potential popularity of a mashup before it is developed. We identify a set of features that play a major role in determining the popularity of a mashup, which are described as follows. (1) **Functionality:** We consider it as the most significant factor since it is the primary search criterion for a user to select a mashup. (2) **Novelty:** Regardless of the functionality and quality of services, a mashup may fail to attract its users if similar mashups are already available and have taken up the market. In another words, the timing of the release decides if a mashup's idea is novel or not. (3) **Use of tags:** As tags play an important role in mashup queries, using appropriate tags can improve the visibility of a mashup, which in turn help attract users. (4) **Selection of services:** A mashup can borrow the user groups and marketing effort of the services it uses. Therefore, the selection of services can make a positive impact, if popular services are selected, or a negative one, if less reputable services are selected, on the popularity of mashups. (5) **Combination of selected services/tags:** We take a further step when checking the services and tags a mashup selects by checking how their combination can affect the popularity of a mashup.

Our approach models and integrates the features related to the above factors and learn how much each factor affects the popularity of a mashup. We then propose a Bayesian learning model that can identify the important features and offer a confidence level for each prediction. Thus, when the model has a high confidence, it will provide a trustworthy insight into the potential popularity. In contrast, when the model has a low confidence, the developer's domain knowledge can be further leveraged as the model does not have enough data to make an accurate prediction.

We summarize our major contributions as follows:

- We present an in-depth investigation on the popularity of mashups using a ProgrammableWeb dataset with 7392 mashups covering a period of five years.
- We are the first to discuss the lack of novelty observation and to exploit the use of tag/API compositions for popularity prediction.
- We suggest a unique approach to build an optimized and self-explanatory feature space that can overcome the sparse nature of the data and quantify the popularity contribution of each feature.
- We propose a Bayesian learning model that can utilize our suggested feature space to make accurate predictions, identify important features, and offer confidence level with each prediction which can provide guidance to developers for successful mashup development.
- We conduct extensive experiments over real-world mashup data to demonstrate the effectiveness of the proposed approach.

The remainder of this paper is organized as follows: In Section 2, we give an overview of the related work. In Section 3, we present our in-depth mashup popularity investigation on the dataset from ProgrammableWeb. In Section 4, we discuss our suggested novel Bayesian approach for design-phase popularity prediction. In Section 5, we present a set of comprehensive experiments to demonstrate the effectiveness of our approach. In Section 6, we conclude and discuss the future work.

2. Related work

In this section, we describe several related work and differentiate them from ours. In general, current research efforts aim to predict the popularity of a web item (Bandari, Asur, & Huberman, 2012; Figueiredo, 2013; He, Gao, Kan, Liu, & Sugiyama, 2014; Keneshloo, Wang, Han, & Ramakrishnan, 2016; Kim, Kim, & Cho, 2012; Lee, Moon, & Salamatian, 2010; Szabó & Huberman, 2010; Tatar, Antoniadis, de Amorim, & Fdida, 2012; Yao, Fu, Liu, Liu, & Xiong, 2016; Yin, Luo, Wang, & Lee, 2012), or leverage the popularity for item filtering or recommendation (Hora & Valente, 2015; Hou & Pletcher, 2010; Jain et al., 2015; Mileva et al., 2010; Wan et al., 2015). In this work, we focus on the earlier, specifically popularity prediction in the service computing domain.

2.1. Popularity prediction in service computing

In (Hora & Valente, 2015; Hou & Pletcher, 2010; Jain et al., 2015; Mileva et al., 2010; Wan et al., 2015), the authors use popularity prediction as part of their model to recommend APIs for mashup developers. In (Hora & Valente, 2015; Mileva et al., 2010), they aim to help developers decide between multiple APIs that offer the same functionality. They both developed a tool that analyzes the usage information of APIs as a metric for popularity, and use such information to make recommendations. In (Jain et al., 2015), the authors suggested a recommender system that can discover and recommend relevant web APIs to developers based on their functionality, usage, and popularity. They used the number of times an API has been used in existing APIs as a way to rank their final list of recommendation. In (Hou & Pletcher, 2010), the authors developed a tool that utilizes the popularity of APIs and their elements to rank suggestions given by code completion systems, and they show that ranking suggestions based on their usage frequency (i.e. popularity) can result in better filtering than other approaches such as alphabetical ranking or relevance ranking. Thus, (Hora & Valente, 2015; Hou & Pletcher, 2010; Jain et al., 2015; Mileva et al., 2010) have used the popularity as a feature in their model/tool to filter/rank existing APIs which is different from our work where we aim to predict the popularity itself. The only exception is (Wan et al., 2015) which we have already addressed in the introduction of this paper. We differ in that we aim to predict the popularity *before* the service is released to the public.

2.2. Popularity prediction in other domains

Current work follows one of two directions (Tatar, de Amorim, Fdida, & Antoniadis, 2014). The first is predicting the popularity prior to the release of the web item (Bandari et al., 2012; Tsagkias, Weerkamp, & de Rijke, 2009), and the second is predicting the popularity after the release of the web item (Figueiredo, 2013; He et al., 2014; Hong, Dan, & Davison, 2011; Keneshloo et al., 2016; Kim et al., 2012; Lee et al., 2010; Lerman & Hogg, 2010; Pinto, Almeida, & Gonçalves, 2013; Rizos, Papadopoulos, & Kompatsiaris, 2016; Szabó & Huberman, 2010; Tatar et al., 2012; Wu, Mei, Cheng, & Zhang, 2016; Yin et al., 2012). The two directions are not competing with each other, but rather, they have a complimentary relationship as the pre-release prediction can address some of the post-release prediction's limitations. They key difference is that a post-release prediction exploits the time-series information for how the popularity changes over time to make a prediction. Such information is not available when the item has not been released yet, or is in its early stages. Furthermore, a pre-release prediction can have a significant value when the goal is to have

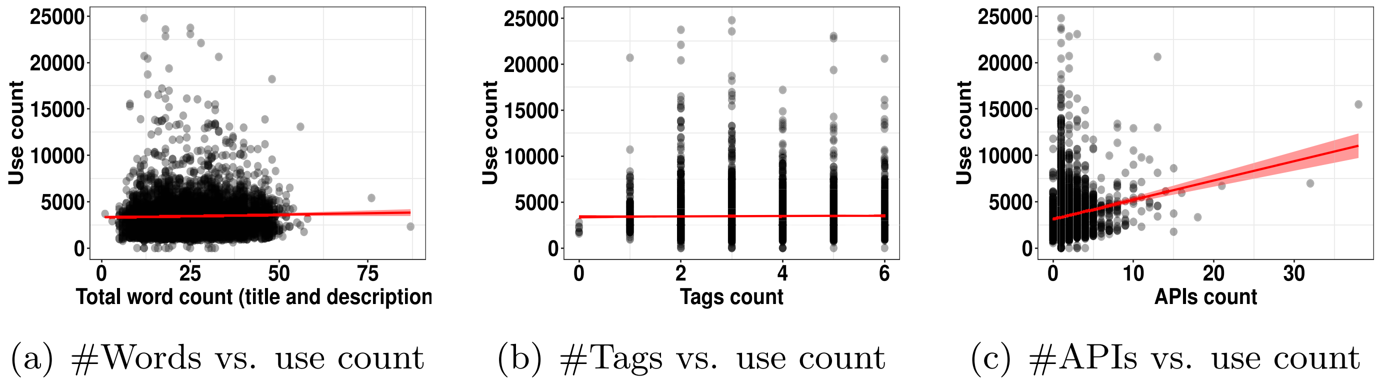


Fig. 1. Is there a strong correlation between the word count, tag count, API count, and the popularity? We compare the popularity of a mashup against the three potential factors (a) textual length (word count), (b) search exposure (tag count), and (c) integrated functionality (API count)

an early-on insight into the potential popularity of a web item to make critical budgeting or marketing decisions, which is what our work aims to provide. The literature on post-release popularity prediction suffers from the same limitation as (Wan et al., 2015) where we explained that an item has to be released to the public and used for a given period of time before a prediction can be made. This kind of setting does not apply to our problem as a pre-development prediction is required. As for the work on pre-release popularity prediction (Bandari et al., 2012; Tsagkias et al., 2009). The authors attempt to predict the popularity of news stories using its content. However, they were not successful as they did not have access to the full body of the news story, which limited their ability to utilize the content thoroughly. Moreover, they ignored other factors that may play a major role in the popularity of news stories such as the geographical factor where the topic might be a popularity magnet, but it is too local, i.e., popular in one source, but not the others. We align ourselves with this kind of work. However, we plan to have a more thorough analysis of the content, and to investigate other factors that may contribute to the popularity. Moreover, our proposed approach *is not simply about an accurate point prediction, but rather about providing a complete prediction framework that can offer an early-on insight into the estimated popularity of a web item, the prediction's confidence level, and the reasoning behind it.*

3. Dataset analysis

We used a dataset from ProgrammableWeb.com, one of the most comprehensive online directories for APIs and mashups (Jain et al., 2015). The website is considered a free and convenient way for developers to market their APIs and mashups. They first started in 2005, and their directory quickly grew to over 10,000 API by 2013[2] The dataset we used was provided by (Jain et al., 2015), and it consists of 4543 mashups. Moreover, we removed eight outlier mashups using the *Extreme Studentized Deviate* test which is a standard technique to identify outliers. The final mashup count after outliers removal became 4535 mashups. Table 1 shows the available information for each mashup in this dataset.

3.1. Exploring the candidate popularity factors

A summary statistics can been seen in Table 2 where we have found that 1) seventy-five percent of mashups use thirty-three words or less to describe their mashup, which means we have short textual information, 2) seventy percent of mashups are tagged with two to four keywords (i.e. tags count), 3) eighty percent of mashups use one or two APIs at most with their mashup (i.e., API count).

Table 1
Summary of the available information for each mashup.

Column	Example
Title	Haiku
Date	2009-07-02T21:35:07Z
Description	Parses #haiku on Twitter and matches . . .
Tags	art, haiku, microblogging, . . .
APIs	Flickr, Twitter
Use count	7097

Table 2
Summary statistics of the service mashups in the ProgrammableWeb dataset.

Column	Min	Mean	3rd Quartile	Max
Use Count	3	3474	4086	24780
log(Use Count)	1.099	8.004	8.315	10.120
Word count	1	25	33	76
Tag count	0	3	4	6
API count	0	1	2	38

Based on Fig. 1, we observed that there's no correlation between the number of words, the number of tags, and the mashup popularity (i.e., use count). This means that having a long description or a large number of tags will have very little effect on the popularity of a mashup. However, it is also observed that having no tags will affect the popularity, as all mashups with zero tags ended up being in the low popular range as seen in Fig. 1. We believe that not properly tagging a service mashup when listing it in online markets can limit the users ability to find it, which may explain this observation.

On the other hand, we can see a much stronger correlation, in Fig. 1, between the API count and the popularity. We observed that mashups in the *high* popular range mostly use one to three APIs; whereas, mashups that use more than three APIs immediately lower their chances of being in the *high* popular range. When taking a closer look, we found that mashups with a high number of APIs are mostly not targeting the general public, but a more specific audience. For example, *USPS Tracking* is a mashup in the upper half of high popularity range (e.g., 20,699 use count) which uses only two APIs (Google Maps and USPS Track & Confirm), and offers a service to track USPS shipments with Google Maps, and is considered relevant to a wide range of audience which explains its very high popularity. Whereas, *Congress SpaceBook* is a mashup in the lower half of medium popularity range (i.e., 4737 use count) which uses eleven APIs (e.g., Flickr, YouTube,

[2] https://www.programmableweb.com/api-research.

Table 3
Demonstrating the effect of the lack of novelty with an example of a cluster with a dominating mashup, and another with no dominating mashup.

Cluster (8) with a dominating mashup		
Title	Pub. Date	Use count
1001 Secret Fishing Holes	Nov. 2005	23,567
Fishingnotes.com	Mar. 2003	3125
Fish Mapper	Apr. 2006	3011
Fishing Stories	Oct. 2006	2842
Flyfishmap	Jun. 2009	1673
. . .	. . .	. . .
Cluster (658) with NO dominating mashup		
Title	Pub. Date	Use count
Earthquake Vulnerable Cities	Aug. 2008	2785
Earthquakes in Last 7 Days	Nov. 2005	3146
Earthquakes this Week	Nov. 2005	4082
World and Regional Earthquakes	Nov. 2006	2322
. . .	. . .	. . .

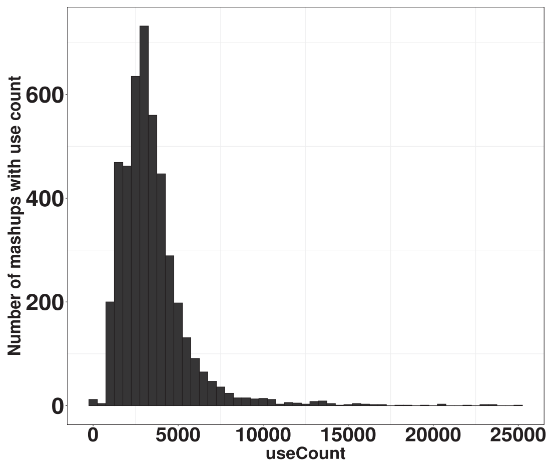
Google Social Graph,..etc), and basically offers a social networking platform for congress, is considered relevant to a significantly smaller audience which explains its low popularity. Thus, the general observation is that the more APIs consumed by a mashup, the higher the chances of it being in the medium popular range (i.e., 2000-7000 use count) as it will most likely be targeting a much smaller audience, so even if it was successful in reaching it's targeted audience, it will still overall be considered within the medium-low popular range (i.e., below 3rd quantile).

When taking a closer look into the functionality the mashups offer, we found that similar mashups have an interesting relationship between them. If we consider a group of similar mashups to be forming a cluster for a specific functionality (e.g., they all offer a hotel finding service), then we can observe that they fall under one of two states: They either have a dominant mashup (i.e., a mashup that has captured most of the attention for that functionality), in which case that dominating mashup would have a significantly higher popularity than its neighbors within the cluster, or they would all be closely related in popularity with no dominant mashup. Table 3 shows an example of a cluster with a dominating mashup, and an example of a cluster with no dominating mashup. We can see that mashups within the same cluster offer similar functionality. For the first case, we observe that once a dominating mashup appears, all the later mashups are likely to be in the low-range popularity of that cluster. As for the second case where we do not have a dominating mashup, we believe that if a cluster has an overall mid-range popularity, then the cluster's functionality can be considered a promising open area for developers to try and build the next mashup that will dominate it. However, in case the cluster had an overall low-range popularity average, then this may indicate that this cluster offers a useless or uninteresting functionality that developers should avoid in the future. In rare cases, a cluster of similar mashups can be dominated by more than a single mashup, however, we have found that in most cases, we only have a single dominating mashup.

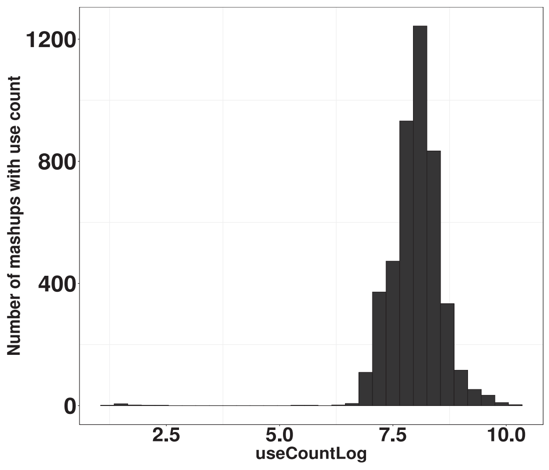
3.2. Measuring the popularity

We measure the popularity of a mashup using the use count metric provided by ProgrammableWeb, which is the only provided popularity metric. Table 2 shows a summary statistics of the use count. The use count metric measures only the raw popularity, i.e., the level of public exposure. It does not capture other aspects of the popularity, e.g., user satisfaction, in which another metric such as the ratio of thumbs up/down would be more appropriate. Nonetheless, it was shown in multiple studies that there's a strong correlation between all popularity metrics. For example, He et al. (2014) showed a strong correlation between the number of views (captures user exposure) and the number of votes up/down (captures user satisfaction) for news stories on the news website Digg. Moreover, Borges, Hora, and Valente (2016) found that there's a strong correlation between the two popularity metrics used in GitHub, the number of forks (captures user exposure) and the number of stars (captures user satisfaction). This means that, in most cases, the higher the public spread/exposure, the higher the user satisfaction. Thus, even though the use count metric does not capture user satisfaction, it should be highly correlated with it.

It was reported previously (Szabó & Huberman, 2010; Yu & Woodard, 2008) that a power-law distribution is expected for this kind of problem. We can observe in Fig. 2 a right skewed distribution of the use count, but without a clear shape of a power law distribution. We believe that because we suffer from not having any zero count mashups, the long-tail distribution is not as clear as we want it to be. Nonetheless, having a dataset with more recent data, we believe a clearer long-tail distribution will be observed. Also, in Fig. 2, we see the logarithmic transformation of the use count which gives us a condensed normal distribution



(a) Y distribution



(b) Log(Y) distribution

Fig. 2. The use count (Y) distribution in the ProgrammableWeb dataset. We show the distribution of the popularity in (a) and the log popularity distribution in (b).

shape with a mean of roughly eight. It's worth mentioning that it's a common statistical practice to log transform the response when we have such a distribution as it makes it less sensitive to outliers and easier to model.

4. Design-phase popularity prediction

In this section, we discuss our suggested approach which consists of our method to construct an optimized and self-explanatory feature space from raw sparse data, and our Bayesian learning model that can predict, select features, and offer confidence level with each prediction.

4.1. Constructing the feature space

The functionality: To derive the functionality of a mashup, we suggest leveraging its title and description as follows. First, we apply a standard natural language processing methods, such as stop-word removal and word stemming, on the textual content of the title and the description to generate a *term frequency-inverse document frequency* matrix or TF-IDF matrix (Manning, Raghavan, & Schütze, 2008). The TF-IDF matrix is a representation of the content where each row is a mashup, and each column is a term. The elements in this matrix represent how relevant a given term is to a specific mashup. This representation allows us to capture the most important terms that describe the content of a mashup. However, TF-IDF usually produces a large matrix that is highly sparse, i.e., a given mashup's vector would have many zero entries as it uses only a few terms out of the available dictionary.

To address this issue, we utilize the probabilistic topic modeling technique Latent Dirichlet Allocation (LDA) (Blei, Ng, & Jordan, 2003). The intuition behind using LDA is that given the TF-IDF matrix, LDA can leverage such representation by grouping together the frequently co-occurring terms into an approximation of a real-world concept, i.e., a topic. The set of topics discovered by LDA would represent a higher level summary of the terms discovered by the TF-IDF approach. As such, LDA is expected to provide a good and compact approximation of the TF-IDF matrix as the number of topics in the LDA matrix is significantly smaller than the number of terms in the TF-IDF matrix. LDA produces a topic proportion matrix D where each row in the matrix represents a mashup, and each column represents a discovered topic. The entries $D_{i,k}$ in the LDA matrix essentially denote the *probability* that topic k describes mashup i. As part of using LDA, we need to specify the number of topics k, and through cross-validation, as seen in Table 4, we found one-hundred to be a good candidate as it offers a balance between model's complexity and model's accuracy. We believe those topics represent the mashups functionalities that we aim to derive. To give a better insight into those discovered topics, Fig. 3 shows the content of two topics, the first (left side) is about traveling, while the second (right side) is about real-estate. We learn the contribution of each discovered topic as follows:

$$d_k = \frac{\sum_{i=1}^{m} D_{i,k} \times y_i}{\sum_{i=1}^{m} D_{i,k}}, \quad \forall k \in \{1, .., K\} \tag{1}$$

where m is the total number of mashups, K is the total number of topics, and y_i is the corresponding popularity (i.e., use count) for mashup i. Thus, each entry in the vector d_k is a score that indicates the topic's contribution towards the popularity. When splitting the dataset into training and testing, we learn the contribution of a new testing mashup with vector $\boldsymbol{\theta}_t \in \mathbb{R}^n$ as follows:

$$\boldsymbol{\theta}_t = \boldsymbol{\theta}_t \circ \mathbf{d} \tag{2}$$

simply, we do an element-wise multiplication between the new mashup's probability vector and the topic-contribution vector that represents the contribution of each discovered topic towards the popularity. We use the generated LDA matrix with the new topics score directly in our model as features.

The selection of tags and services: The standard way to capture the use of tags and the selection of services (i.e., APIs) is to create two binary frequency matrices. The rows in those matrices represent our mashups and the columns represent the used tags in the first matrix, and the selected services (i.e., APIs) in the second matrix, where each entry denotes if a given tag/API was used in a given mashup or not (i.e., binary score). However, since we have 1409 unique tag, and 788 distinct service (i.e., API), and that developers use on average 2-3 tags and 1-2 APIs per servie mashup, we have an extremely sparse matrix. Thus, we suggest a better two-step approach to replace those two sparse and large matrices with only two features: the tag score feature and the API score feature. These score features will denote the contribution of the used tags, for the tag score, and contribution of selected services (i.e., APIs) for a given mashup. We constructed those two features as follows: In step one, we learn the *averaged* contribution of each tag/API towards popularity. To learn the contribution of each tag, we divide the use count (i.e., popularity) of each mashup in the tag matrix by the number of tags it uses, and assign that as a new score for the used tags. At this point, for each column in both the tags matrix, we have a score that represents the contribution of that tag/API towards the popularity of the mashups. We take the average of each column which represents the *averaged* contribution towards popularity for a given tag, and assign it as a score for

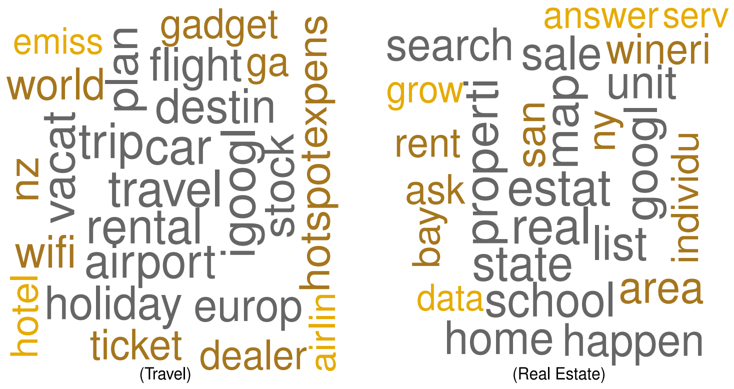


Fig. 3. An example of two discovered LDA topics, a travel related topic on the left, and a real state related topic on the right. The two example topics highlight LDA's ability to summarize the textual content into a set of real-world concepts.

Table 4
Finding the optimal number of topics (K) for LDA. The lowest RMSE can be observed when the number of topics is 100.

K	5	30	50	100	250	500	1000
RMSE	0.6302	0.6292	0.6242	**0.6181**	0.6270	0.6290	0.6355

Table 5
Examples of frequent tag/API compositions. The combination of such compositions lead to unique functionalities. For example, merging Flickr's capability with Google-maps allowed users to search for their images based on where the images were taken, i.e., location. This unique functionality, captured by the composition, can be a leading factor behind the popularity of the service mashup.

Tags	APIs
Photo, Map	Flickr, Google-maps
Video, Music	YouTube
Social, Microblog	Twitter
Video, Photo	Flickr, YouTube

the whole column. We do the same for the API matrix to learn the *averaged* contriubution of each API towards the popularity. In step two, given a service mashup, we add up the individual averaged contributions of the tags that it uses to create the tag score feature, and add up the individual contribuitons of APIs that it uses to create the API score feature. When splitting the data into training and testing, we use the average contribution for each tag/API that we learned from training as a score for the testing as well. We then add up the individual contributions in the same manner.

The combination of selected tags/services: To capture the role such compositions play in the popularity of a mashup, we suggest finding those compositions and building a binary frequency matrix that allows us to use them as features. To find those compositions, we suggest the use of Apriori algorithm (Tan, Steinbach, & Kumar, 2005) which is a standard technique to find frequently used compositions. The selected *support* level for Apriori should offer a balance between finding all possible compositions and maintaining a statistical meaning for the compositions. It is expected to have a large number of compositions, and that should not be a problem as our suggested Bayesian learning model can select the most relevant ones. In our dataset, we were able to find 178 frequently used compositions. Table 5 shows a few of the discovered compositions. For example, the first composition represents the use of (Photo and Map) as tags and (Flickr and Google-maps) as APIs. This combination created a mashup with an interesting functionality that allowed users to know the location of where their Flickr images were taken. We believe this interesting functionality, captured by the composition, is behind the popularity of the mashup. We used those frequent compositions to create the binary frequency matrix which we used directly as features in our model.

Novelty: As we have explained earlier, a mashup may fail to attract its users if similar mashups are already available and have taken up the market. We observed this through our analysis in which we found that when we cluster similar mashups together, it's common to see one of two states: A cluster with a dominant mashup, or a cluster with no dominant mashup. In the first case, we observed that once a dominant mashup appears, it would capture most of the attention for that cluster's functionality forcing all the other mashups, especially the later ones, in that cluster to settle-in for a lower popularity. In other words, we can say that the other *dominated* mashups within the cluster *lack the novelty* as the dominant mashup is presumed to be the first in the cluster to successfully capture all the user's needs for that functionality. Thus, we suggest to create a new feature vector called the *lack of novelty* where we penalize all the *dominated* mashups with a score of one, as we expect them to have a low/medium popularity (i.e., use count below 3rd quantile), and assign a score of zero to all other mashups including dominating mashups and mashups in clusters with no dominating mashup as we have no evidence that they lack the novelty. We then use that vector as a feature in our model.

However, to achieve the suggestion mentioned above, we need to determine the best approach to measure the similarity between the functionality of two mashups. We suggest combining the knowledge from both the content found in the title/description of the mashup, and from the list of used tags and APIs as follows:

$$Sim_{i,j} = \alpha \times \mathbf{C}_{i,j} + (1 - \alpha) \times \mathbf{J}_{i,j} \tag{3}$$

where α is a learned probability weight between zero and one. $\mathbf{C}_{i,j}$ is a cosine similarity matrix (Tan et al., 2005) that measures the similarity between the title and the description of two given mashups. $\mathbf{J}_{i,j}$ is a jaccard similarity matrix (Tan et al., 2005) that measures the similarity between the list of tags and APIs of two given mashups. The α weight measures how much trust you place on your content from the title/description. If the dataset lacks proper description, but is tagged properly, then less weight can be placed on the content from the title/description so that more weight is placed on the list of used tags and APIs, and vice versa. If there is no clear pattern in the dataset, then the recommended approach in such case would be to provide equal weights to both aspects. However, if there's a clear preference in the dataset (i.e., community), the proposed approach can provide better predictions if the preference is reflected in the provided weights. In the rare case where mashups are posted without any meta information (i.e., both the description and tags are empty), the model would not have enough data to make a confident predication. Nonetheless, it is expected that such an extreme case (i.e., no meta information) would be difficult even for human experts as they would find it impossible to make a judgement with no available information on the mashup. It is important to clarify that in this approach we assume that professional developers will put a great deal of effort in preparing the meta information (i.e., description and/or tags) of their mashup to ensure proper exposure of their work. This assumption is needed for the model to provide accurate and confident predictions. For our dataset, with cross-validation, we have found that an α value of 0.9 produced clusters that met our requirement in that mashups were clustered together based on functionality.

Next, we suggest using hierarchical clustering (Tan et al., 2005) to create the clusters using the averaged similarity matrix $Sim_{i,j}$ that we already constructed. As it's the case with most clustering algorithms, in hierarchical clustering, we need to specify the number of clusters as a parameter to the algorithm. we found 2197 to be a good number of clusters for our dataset. The number of clusters we chose is the total number of unique tags (1409) and APIs (788). Since each cluster should represent a unique possible functionality, the choice of the number of cluster should represent the number of unique possible functionality we assume to exist in the dataset. Thus, we are making the assumption that for each unique available tag and API, at least a single possible unique functionality exists.

Finally, to identify the clusters with a dominant mashup from the ones without, we looked for an outlier point in the cluster where we measured how many standard deviations each point is away from the mean using *z-score*. To determine if a point within a cluster is an outlier or not, we measure how many standard deviations it is from the mean of the cluster. If we found that it's *t* standard deviations away from the mean, then we declare it as a dominating mashup. The value for *t* has to be determined through cross validation. In our case, we have found three to be a good value for *t*. We can now create our lack of novelty feature vector as described above, and use it as a feature in our model.

4.2. The prediction model

We present a Bayesian learning model for popularity prediction. The proposed model offers three major advantages over other regression models. First, instead of just providing a point prediction, the Bayesian model outputs a predictive distribution for a given test mashup. The variance of the predictive distribution can be used to quantify the confidence level of the prediction. Second, we integrate the Bayesian learning model with the Auto Relevance Determination (ARD) mechanism (Bishop, 2006), which allows us to perform feature selection and identify the most important factors that affect mashup popularity. Third, by performing type 2 maximum likelihood, we can automatically optimize the hyperparameters of the model, which avoid the tedious process of cross-validation required by many other models.

4.2.1. Model inference

We start by assuming the response t is a random variable whose distribution conditioned on input x is Gaussian:

$$p(t|\mathbf{x},\mathbf{w},\beta)=\mathcal{N}(t|\mathbf{w}^T\boldsymbol{\phi}(\mathbf{x}),\beta^{-1}) \tag{4}$$

where β is the precision of the Gaussian and $\boldsymbol{\phi}(\mathbf{x})$ is the feature vector of mashup $\mathbf{x}$.

The likelihood of the training data $\mathbf{X}$ then is given as:

$$p(\mathbf{t}|\mathbf{X},\mathbf{w},\beta)=\prod_{n=1}^{N}p(\mathbf{t}_n|\mathbf{x}_n,\mathbf{w},\beta^{-1}) \tag{5}$$

The flexibility of the Bayesian inference framework allows us to incorporate different prior knowledge for different learning effects. Specifically in this work, we assume that not all features are equally important to the prediction problem. As a result we choose a conjugate Gaussian prior(A.K.A ARD prior) on the coefficient random variable $\mathbf{w}$ to conduct feature selection:

$$p(\mathbf{w}|\boldsymbol{\alpha})=\mathcal{N}(\mathbf{0},A^{-1}) \tag{6}$$

where A is a diagonal matrix governed by hyper-parameter $\boldsymbol{\alpha}$ where α_i denotes the i-th diagonal entry of A. Section 4.2.3 provides the detailed discussion of how feature selection can be achieved by adopting ARD prior.

According to the Bayesian rule, the posterior distribution of $\mathbf{w}$ is proportion to the product of the likelihood and prior, which is also Gaussian due to conjugacy:

$$p(\mathbf{w}|\mathbf{t},\boldsymbol{\alpha},\beta)=\mathcal{N}(\mathbf{w}|\mathbf{m},\Sigma) \tag{7}$$

where the posterior mean and the covariance are given as follows:

$$\mathbf{m}=\beta\Sigma\Phi^T,\quad \Sigma=(A+\beta\Phi\Phi^T)^{-1} \tag{8}$$

Φ is the design matrix. The i-th row of Φ is $\boldsymbol{\phi}(\mathbf{x}_i)$. Assume that the optimal values of the hyper-parameters, $\boldsymbol{\alpha^*}$ and β^* can be learned (see the next section for details). We can derive the predictive distribution over a test mashup $\mathbf{x}_t$ by integrating out $\mathbf{w}$, which is also a Gaussian:

$$\begin{aligned}p(t|\mathbf{X},\mathbf{x}_t,\boldsymbol{\alpha^*},\beta^*)&=\int p(t|\mathbf{x}_t,\mathbf{w},\beta^*)p(\mathbf{w}|\boldsymbol{\alpha}^*,\mathbf{X},\beta^*)d\mathbf{w}\\&=\mathcal{N}(\mathbf{m}^T\boldsymbol{\phi}(\mathbf{x}_t),\sigma^2(\mathbf{x}_t))\end{aligned} \tag{9}$$

where the predictive mean and the covariance are given as follows.

$$\sigma^2(\mathbf{x}_t)=(\beta^*)^{-1}+\boldsymbol{\phi}(\mathbf{x}_t)^T\Sigma\boldsymbol{\phi}(\mathbf{x}_t) \tag{10}$$

Besides using the mean of the predictive distribution (i.e., $\mathbf{m}^T\boldsymbol{\phi}(\mathbf{x}_t)$) to predict the future use count of $\mathbf{x}_t$, the variance $\sigma^2(\mathbf{x}_t)$ provides important information to quantify the confidence level of the prediction.

4.2.2. Learning process

Estimating hyper-parameters $\boldsymbol{\alpha}$, β yields a type-2 maximum likelihood problem. Specifically, we maximize the log of the model evidence given by:

$$\begin{aligned}\ln p(\mathbf{t}|X,\boldsymbol{\alpha},\beta)&=\ln\int p(\mathbf{t}|X,\mathbf{w},\beta)p(\mathbf{w}|\boldsymbol{\alpha})d\mathbf{w}\\&=\ln\mathcal{N}(0,C)\\&=-\frac{1}{2}(N\ln(2\pi)+\ln(C)+\mathbf{t}^TC^{-1}\mathbf{t})\end{aligned} \tag{11}$$

where C is given by

$$C=\beta^{-1}I+\Phi A^{-1}\Phi^T \tag{12}$$

By setting the partial derivative of (11) with respect to $\boldsymbol{\alpha}$ and β to zero, we derive the solutions for both hyper-parameters

$$\begin{aligned}\alpha_i^*&=\frac{\gamma_i}{m_i^2}\\(\beta^*)^{-1}&=\frac{||\mathbf{t}-\Phi\mathbf{m}||^2}{N-\sum_i\gamma_i}\end{aligned} \tag{13}$$

where γ_i is defined by

$$\gamma_i=1-\alpha_i\Sigma_{ii} \tag{14}$$

The learning proceeds by using (8) and (13) alternatively with randomly initialized $\boldsymbol{\alpha}$ and β until convergence.

4.2.3. Feature selection

The first updating rule from (13) implies an implicit solution as the right hand side is also a function of α_i. To determine the stationary point of the log likelihood function (11) explicitly, we can extract the contribution from α_i out of the covariance matrix C in (11):

$$\begin{aligned}C&=\beta^{-1}I+\sum_{j\neq i}\alpha_j^{-1}\boldsymbol{\varphi}_j\boldsymbol{\varphi}_j^T+\alpha_i^{-1}\boldsymbol{\varphi}_i\boldsymbol{\varphi}_i^T\\&=C_{-i}+\alpha_i^{-1}\boldsymbol{\varphi}_i\boldsymbol{\varphi}_i^T\end{aligned} \tag{15}$$

where $\boldsymbol{\varphi}_i$ denotes the i-th column of Φ and C_{-i} represents matrix C with the removal of $\boldsymbol{\varphi}_i$. Substituting (15) in (11), the log likelihood can be written as:

$$\ln\mathcal{N}(0,C)=L(\boldsymbol{\alpha}_{-i})+\lambda(\alpha_i) \tag{16}$$

where $L(\boldsymbol{\alpha}_{-i})$ denotes the log likelihood function with $\boldsymbol{\varphi}_i$ omitted and function $\lambda(\alpha_i)$ is defined as:

$$\lambda(\alpha_i)=\frac{1}{2}\left[\ln\alpha_i-\ln(\alpha_i+s_i)+\frac{q_i^2}{\alpha_i+s_i}\right] \tag{17}$$

where q_i and s_i are defined as:

$$\begin{aligned}s_i&=\boldsymbol{\varphi}_i^TC_{-i}^{-1}\boldsymbol{\varphi}_i\\q_i&=\boldsymbol{\varphi}_i^TC_{-i}^{-1}\mathbf{t}\end{aligned} \tag{18}$$

The partial derivative of (17) with respect to α_i is

$$\frac{\alpha_i^{-1}s_i^2-(q_i^2-s_i)}{2(\alpha_i+s_i)^2} \tag{19}$$

Setting (19) to zero gives two possible solutions for α_i:

$$\begin{cases}\alpha_i\to\infty, & q_i^2<s_i\\ \alpha_i=\frac{s_i^2}{q_i^2-s_i}, & \text{otherwise}\end{cases} \tag{20}$$

In the first case, as α_i (i.e., precision of coefficient w_i) approaches to infinity, w_i will be driven to its mean (i.e., 0). This will result in the removal of the corresponding feature from the model, which achieves feature selection.

5. Experiment

We first describe the experimental setup. We then compare the prediction accuracy of our proposed Bayesian learning model with other competitive models, and show how our learning model is overall superior. Moreover, we show examples of our model's ability to offer confidence level with each prediction. Finally, we evaluate our approach in building the feature space and ability to identify the important features.

5.1. Experimental Setup

The current use count (i.e., popularity) of a mashup is the total number of use count accumulated over the years since its publication date. Thus, it favors older mashups over newer ones as the newer mashups had less time to accumulate their use count. To have a more balanced and fair scale, we instead used the average yearly use count as our response in which we divided the original use count by the mashup's lifetime (i.e., number of years it's been available in the market).

To simulate a real world scenario, we used the information from the mashups listed in the first four years as the training data, to predict the popularity of the fifth's year mashups. In other words, we are training the model on service mashups that were created in the first four years, and testing the model on service mashups that were created on the fifth year. To measure the prediction accuracy, we used Root Mean Square Error (RMSE) which is a standard way to measure the difference between the true and predicted values. In general, the lower the RMSE, the more accurate the model.

5.2. Model performance

To evaluate the prediction performance of our proposed Bayesian learning model versus other models, we used a feature space of 287 features where for each mashup we have a tag score feature, an API score feature, a Lack of Novelty score feature, 100 LDA features (one feature per topic), and a 174 binary API/tag composition features.

Fig. 4 shows the prediction result of our proposed model versus Linear Regression with L1 norm regularization (i.e., Lasso) and Linear Regression with L2 norm regularization (i.e., Ridge Regression). Both ridge regression and lasso require parameter tuning (i.e., lambda) and their performance significantly rely on the selected parameter value. For ridge regression, we can see that a low or a high lambda value can drastically decrease the performance; whereas, with lasso regression, the higher the lambda value, the lower the performance as the model becomes more selective of what features to use. On the other hand, our proposed bayesian learning model does not require any parameter tuning as it can directly give the optimal or near-optimal solution. Moreover, it can identify the important features which ridge regression does not offer, and it provides a confidence level with each prediction which both ridge and lasso regression cannot do. Thus, overall, it offers the best prediction framework.

To show how our suggested learning model can offer confidence level with each prediction, we present a few examples in Table 6. The first one is a mashup we predicted with a relatively large error, and the second one is a mashup we predicted with a smaller error. The general observation is that if we have a small variance, then we are more confident about the prediction, and vice versa. For example, we predicted more accurately the second

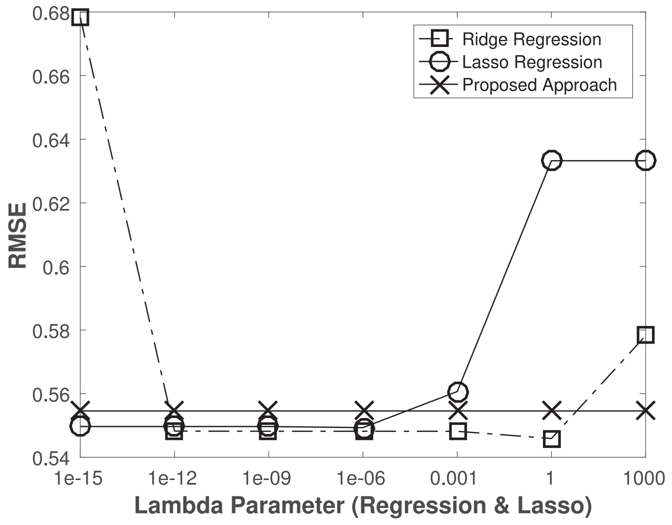


Fig. 4. Proposed model's performance vs. other regression models. We can observe that the performance of other regression models can vary greatly depending on the selected parameter value; whereas, the proposed approach provides a consistent performance as it requires no parameter tuning.

mashup, and the model confirms that fact by showing a small variance for the prediction (i.e., a high confidence). On the other hand, our prediction of the first mashup is more off as the model does not have enough historical data to make a more accurate prediction, so the model presents a much higher variance which means it has a low confidence in this prediction.

5.3. Feature analysis

To show the performance boost when using our unique approach to construct feature space versus simpler standard methods, we created an alternative feature space that uses word frequency matrix to capture the role of the title and the description of the mashup, and a binary frequency matrix to capture the role of the tags and APIs. The result can be seen on Table 7 where our approach in constructing the feature space is not only offering a significantly smaller feature space, but also a drastically better prediction accuracy compared to the frequency approach.

Furthermore, we show in Table 8 the added value of each suggested feature using different models as follows:

Table 6
Examples of the model's estimated popularity (on a logarithmic scale) and confidence level compared to the true mashup's popularity. We can observe that the predicted values are close to the true values, and that the behaviour of the model matches the intuition in that the lower variance (i.e., higher confidence) maps to a more accurate model.

Mashup	True Pop.	Predicted Pop.	Variance	SD
Adult Or Not	8.4740	7.4677	0.3763	0.6134
QuoteRelish	6.9697	7.1060	0.2655	0.5152

Table 7
Comparing our unique approach to construct the feature space versus other standard methods in terms of complexity and accuracy. We can observe that the proposed approach provides a superior accuracy while using a significantly less number of features.

Approach	#Features	RMSE
Standard methods	10455	1.1046
Suggested approach	277	0.5545

Table 8
Measuring the incremental performance boost with each added set of features starting with the LDA topics as a base, and then adding our features incrementally.

Model	#Features	RMSE
Base	100	0.6287
+ Compositions	274	0.5897
+ API Score	275	0.5742
+ Lack of Nov.	276	0.5688
+ Tag Score	277	0.5545

- **Base**: Using the 100 topics generated from LDA, where each topic's probability is replaced with the calculated score (100 features).
- **+ Compositions**: Using the previous model features and the binary matrix of compositions generated from the Apriori algorithm as features (274 features).
- **+ API Score**: Using the previous model features and the API score feature (275 features).
- **+ Lack of Nov.**: Using the previous model features and the lack of novelty feature (276 features).
- **+ Tag Score**: Using the previous model features and the tag score feature (277 features).

As we can see, the baseline is performing quite well as expected since the discovered LDA topics are able to capture the offered functionality of the mashup, and their current score represent their contribution towards the popularity. Nonetheless, each of our added features was still able to improve the model, and collectively they improved the baseline model's accuracy by roughly 12%. The compositions offered the biggest improvement, but it added a high complexity (174 new features). Whereas, the other three features were able to collectively add the same level of improvement to the model but with drastically less added complexity which shows their significance. Furthermore, we used random forest regression and lasso regression as well as our proposed bayesian model to evaluate their importance to the model as features. We found that all three models picked the *tag score feature* and the *API score feature* as the top two most important features which confirms that they play a major role in the accuracy of the model. The models did not all agree on the rank of the *lack of novelty feature* as our bayesian learning model suggested it was the 7th most important feature, random forest as the 16th, and lasso as the 25th. We believe this is the case because the lack of novelty feature is targeting a specific observation, and thus is used for only a small subset of the mashups (roughly 16% in our dataset) which may not show an overall significance, but should be critical for those relevant mashups.

Table 9
Demonstrating a test mashup as an example of the information a software developer would provide to the proposed model (input), and the design-phase insight he/she would recieve (output).

Mashup's information	
Title	Flyfishmap
Description	User generated fly fishing information using Google . . .
Tags Used	#fishing #flyfishing . . .
APIs used	Google-maps, YouTube . . .
Mashup's design-phase popularity insight	
True Popularity (Log)	7.4223 (i.e., 1673 use count)
Predicted Popularity (Log)	6.9475 (i.e., 1040 use count)
Popularity Range	Low
Prediction Variance	0.27243
Func.#1 (Maps & Social Sharing)	25.3 (above 90th Percentile)
Func.#2 (Fishing & Wildlife)	3.26 (below 30th Percentile)
Lack of Novelty	1
Tags Contribution Score	4.04 (below 10th Percentile)
APIs Contribution Score	15.41 (above 90th Percentile)
Popular Combinations	map (tag), YouTube (API), Google-maps (API)

5.4. Illustration ofoffered insight

One key benefit of using the design-phase popularity prediction model (i.e., a pre-release model) is the early-on insight it can provide into the potential popularity of a web item. Such insight plays a major role when critical budgeting or marketing decisions need to be made before an item is released to the public. In previous work, the offered insight was merely a popularity range prediction (i.e., whether the web item would recieve a low, medium, or high popularity). In this paper, one of our contributions is that we take this early-on insight a step further by offering a confidence level with each prediction and by exploiting the self-explanatory feature space we built to provide the reasoning behind the popularity.

In Table 9, we show a test mashup from our experiment where the developer created a web app that shows nearby fishing locations on Google Maps. Also, the web app allows users to post images and YouTube videos to share their experience. In most cases, all the information shown in Table 9 is known early-on to the developer, even before he/she starts to invest time and money

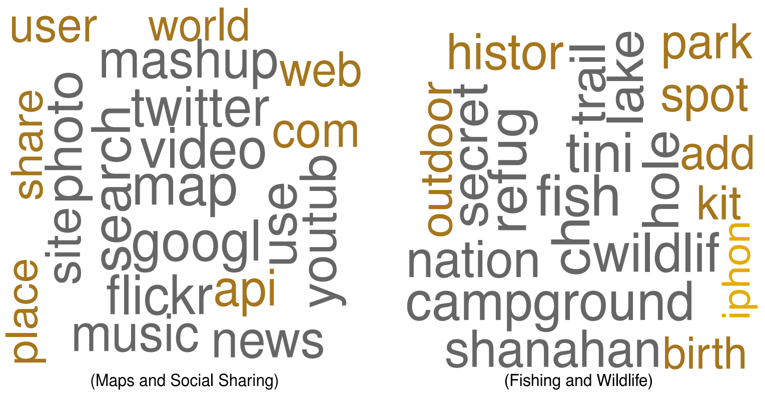


Fig. 5. The two discovered topics (i.e., functionalities). We can observe that the model was able to capture the two main concepts behind the shown test mashup, in which users share fishing information and locations. The first discovered topic shown on the left can be mapped to the general concept of *Maps and Social Sharing* based on the observed terms (e.g., *map, Twitter, share*, and *user*). The second discovered topic can be mapped to the general concept *Fishing and Wildlife* based on observed terms (e.g., *campground, park, outdoor*, and *fish*).

on developing the mashup. Given this information, here is the kind of insight the suggested model would offer to the developer:

- First, we provide an estimated popularity with a low variance of 0.27 which indicates a high confidence in the prediction. We can see that the model is quite accurate with only a 633 use count difference between the true and predicted popularity.
- Second, as seen in Fig. 5, two functionalities were discovered. The first (i.e., maps and social sharing of images and videos) attracts a large audience as the functionality contribution score is above the 90th percentile of all the discovered functionalities scores. Whereas, the second (i.e., fishing and wildlife) attracts only a small audience as its score is below the 30th percentile.
- Third, the idea lacks the novelty as this functionality is already offered by an existing mashup that has successfully captured the market.
- Fourth, the selected tags (all related to fishing) attract a small audience (i.e., users searching for such functionality represent a small club) as the tags contribution score is below the 10th percentile.
- Fifth, the selected APIs (i.e., Google-maps and YouTube) attract a large audience as the APIs contribution score is above the 90th percentile of the APIs scores.
- Finally, the used combination of tag (maps) and APIs (YouTube and Google-maps) is a popular combination.

Given such insight, the developer is well-informed *early-on* of the estimated popularity, the estimated audience for each component (i.e., strength/weakness), and the confidence in the estimation. We hope such insight provides useful guidance to developers for successful mashup development.

6. Conclusion and future work

In this work, we presented a comprehensive investigation on the popularity of mashups through the analysis of a five-year period ProgrammableWeb dataset. Based on our analysis, we identified five factors that contribute to the popularity of mashups: 1) Functionality, 2) novelty, 3) use of tags, 4) selection of services, and 5) combination of selected services/tags. Our suggested approach models those factors and construct the feature space in an optimized and self-explanatory way that solves the sparsity issue of real-world data and quantifies the contribution of each feature towards the popularity. More importantly, we suggested a Bayesian learning model that can estimate the popularity of a service mashup even before it is developed using design-phase features only. Furthermore, the model can identify the important features and offer a confidence level with each prediction which provides useful guidance to developers. When the model has enough data to make an accurate prediction (i.e., has high confidence), it can accurately estimate the popularity and offer an extended insight into the reasoning behind the popularity. When the model does not have enough data to make an accurate prediction, it informs the developer that his/her domain expertise should be leveraged more by presenting a low confidence in the prediction.

For future work, we would like to use Hierarchical Dirichlet Process (Teh, Jordan, Beal, & Blei, 2004) to determine the optimal number of topics for Latent Dirichlet Allocation (LDA). We also plan to integrate real-world features with our model such as the popularity of a specific LDA topic on social media. Moreover, we plan to capture the temporal nature of popularity as we do not consider the time aspect in our current work.

Acknowledgment

This research was supported in part by an NSF IIS award IIS-1814450 and an ONR award N00014-18-1-2875. The views and conclusions in the paper are those of the authors and should not be interpreted as representing any funding agency.

Credit authorship contribution statement

Moayad Alshangiti: Conceptualization, Formal analysis, Writing - original draft. **Weishi Shi:** Formal analysis, Writing - original draft. **Xumin Liu:** Conceptualization, Data curation, Writing - original draft. **Qi Yu:** Conceptualization, Writing - original draft.

References

Bandari, R., Asur, S., & Huberman, B. A. (2012). The pulse of news in social media: Forecasting popularity. *AAAI international conference on weblogs and social media ICWSM*.

Benslimane, D., Dustdar, S., & Sheth, A. (2008). Services mashups: The new generation of web applications. *IEEE Internet Computing, 12*(5), 13–15.

Bishop, C. M. (2006). *Pattern recognition and machine learning (information science and statistics)*. Springer-Verlag New York, Inc..

Blei, D. M., Ng, A. Y., & Jordan, M. I. (2003). Latent dirichlet allocation. *Journal of Machine Learning Research, 3*, 993–1022.

Borges, H., Hora, A. C., & Valente, M. T. (2016). Understanding the factors that impact the popularity of github repositories. In *IEEE international conference on software maintenance and evolution* (pp. 334–344).

Figueiredo, F. (2013). On the prediction of popularity of trends and hits for user generated videos. In *ACM international conference on web search and data mining* (pp. 745–754).

He, X., Gao, M., Kan, M., Liu, Y., & Sugiyama, K. (2014). Predicting the popularity of web 2.0 items based on user comments. In *ACM international conference on research and development in information retrieval* (pp. 233–242).

Hong, L., Dan, O., & Davison, B. D. (2011). Predicting popular messages in twitter. In *Proceedings of the ACM 20th international conference on world wide web, WWW* (pp. 57–58).

Hora, A. C., & Valente, M. T. (2015). Apiwave: Keeping track of API popularity and migration. In *IEEE international conference on software maintenance and evolution* (pp. 321–323).

Hou, D., & Pletcher, D. M. (2010). Towards a better code completion system by API grouping, filtering, and popularity-based ranking. In *ACM international workshop on recommendation systems for software engineering* (pp. 26–30).

Jain, A., Liu, X., & Yu, Q. (2015). Aggregating functionality, use history, and popularity of apis to recommend mashup creation. In *International conference on service-oriented computing: 9435* (pp. 188–202). Springer.

Keneshloo, Y., Wang, S., Han, E. S., & Ramakrishnan, N. (2016). Predicting the popularity of news articles. In *Proceedings of the 2016 SIAM international conference on data mining* (pp. 441–449).

Kim, S., Kim, S., & Cho, H. (2012). A model for popularity dynamics to predict hot articles in discussion blog. In *International conference on ubiquitous information management and communication* (pp. 10:1–10:8).

Lee, J. G., Moon, S. B., & Salamatian, K. (2010). An approach to model and predict the popularity of online contents with explanatory factors. In *IEEE/WIC/ACM international conference on web intelligence* (pp. 623–630).

Lerman, K., & Hogg, T. (2010). Using a model of social dynamics to predict popularity of news. In *ACM international world wide web conference* (pp. 621–630).

Manning, C. D., Raghavan, P., & Schütze, H. (2008). Introduction to information retrieval.

Mileva, Y. M., Dallmeier, V., & Zeller, A. (2010). Mining API popularity. In *International conference on testing, practice and research techniques: 6303* (pp. 173–180). Springer.

Pinto, H., Almeida, J. M., & Gonçalves, M. A. (2013). Using early view patterns to predict the popularity of youtube videos. In *Proceedings of the ACM 6th international conference on web search and data mining, WSDM* (pp. 365–374).

Rizos, G., Papadopoulos, S., & Kompatsiaris, Y. (2016). Predicting news popularity by mining online discussions. In *ACM 25th international conference on world wide web, WWW* (pp. 737–742).

Szabó, G., & Huberman, B. A. (2010). Predicting the popularity of online content. *Commun. ACM, 53*(8), 80–88.

Tan, P., Steinbach, M., & Kumar, V. (2005). *Introduction to Data Mining*. Addison-Wesley.

Tatar, A., de Amorim, M. D., Fdida, S., & Antoniadis, P. (2014). A survey on predicting the popularity of web content. *J. Internet Services and Applications, 5*(1), 8:1–8:20.

Tatar, A., Antoniadis, P., de Amorim, M. D., & Fdida, S. (2012). Ranking news articles based on popularity prediction. In *IEEE international conference on advances in social networks analysis and mining* (pp. 106–110).

Teh, Y. W., Jordan, M. I., Beal, M. J., & Blei, D. M. (2004). Sharing clusters among related groups: Hierarchical dirichlet processes. In *Proceedings of the 17th conference on advances in neural information processing systems NIPS 2004* (pp. 1385–1392).

Tsagkias, M., Weerkamp, W., & de Rijke, M. (2009). Predicting the volume of comments on online news stories. In *Proceedings of the ACM 18th conference on information and knowledge management, CIKM* (pp. 1765–1768).

Wan, Y., Chen, L., Wu, J., & Yu, Q. (2015). Time-aware API popularity prediction via heterogeneous features. In *IEEE international conference on web services* (pp. 424–431).

Wu, B., Mei, T., Cheng, W., & Zhang, Y. (2016). Unfolding temporal dynamics: Predicting social media popularity using multi-scale temporal decomposition. In *Proceedings of the AAAI 30th conference on artificial intelligence, AAAI* (pp. 272–278).

Yao, Z., Fu, Y., Liu, B., Liu, Y., & Xiong, H. (2016). POI recommendation: A temporal matching between POI popularity and user regularity. In *Proceedings of the IEEE 16th international conference on data mining, ICDM* (pp. 549–558).

Yin, P., Luo, P., Wang, M., & Lee, W. (2012). A straw shows which way the wind blows: ranking potentially popular items from early votes. In *ACM international conference on web search and web data mining* (pp. 623–632).

Yu, Q., Liu, X., Bouguettaya, A., & Medjahed, B. (2008). Deploying and managing web services: issues, solutions, and directions. *VLDB J., 17*(3), 537–572.

Yu, S., & Woodard, C. J. (2008). Innovation in the programmable web: Characterizing the mashup ecosystem. In *International conference on service-oriented computing: 5472* (pp. 136–147). Springer.