



FAME: An Influencer Model for Service-Oriented Environments

Faisal Binzagr¹, Hamza Labbaci², and Brahim Medjahed¹(✉)

¹ Department of Computer and Information Science,
University of Michigan - Dearborn, Dearborn, USA
{faisalb,brahim}@umich.edu

² Department of Computer Science, University of Tours, Tours, France
hemza.labbaci@univ-tours.fr

Abstract. We propose FAME (*inFLuencer Apis in developer coMMuni-tiEs*), a multi-dimensional influencer model for APIs in service-oriented environments. We define influence as the extent to which an API is likely to be adopted in mashups and service-oriented applications. The proposed model helps providers increase the visibility of their APIs and developers select the best-in-class APIs. We extract more than eighteen textual and non-textual API features from various programming communities such as `GitHub`, `StackOverflow`, `HackerNews`, and `ProgrammableWeb`. We perform sentiment analysis to quantify developers' opinions towards using APIs. We introduce a cumulative API influence score to measure the influence of APIs across communities and categorize APIs into tiers based on their influence. We introduce a linear regression technique to predict the evolution of influence scores and correlate API features to those scores. We conduct experiments on large and real-world data-sets extracted from the above mentioned programming communities to illustrate the viability of our approach.

Keywords: API · Service-orientation · Mashup · Social content · Influencer · Developer community

1 Introduction

Service-oriented computing allows companies to break down capabilities and business functionalities into individual, autonomous services [5]. The last decade has seen a surge of services in the form of Web APIs (simply APIs) in a variety of domains [15]. The API economy is growing rapidly and companies are making APIs an integral part of their software development strategies. For instance, the `ProgrammableWeb` directory lists more than 22,200 APIs (as of August 2019). APIs enable developers to access hardware and software resources via Internet and Web-specific protocols. Using APIs accelerates the development of value-added applications (e.g., mashups) by providing reusable functionalities out-of-the-box. However, integrating multiple APIs created by diverse third parties requires a wide array of technical skills such as Web (e.g., REST), data management (e.g., JSON), programming (e.g., SDKs), and security (e.g., authentication) [12, 15]. To overcome these challenges, developers often turn to

programming communities (e.g. `GitHub`) to share practices, knowledge, experience, and brainpower in solving intricate problems. For instance, `GitHub` reports (as of May 2019) having over 37 million users and 100 million repositories. Gathering and analyzing content about API usage and activities in existing communities (e.g., number of bugs, developer feedback, number of mashups) provides opportunities to better understand developers' interactions with APIs and detect relationships between APIs and mashups [5, 13].

Combining social network analysis with service-oriented computing could bring novel insights to service selection, recommendation, and composition [4, 5]. One particular research area that received significant attention in social computing is social *influence* [9]. Influencers have the power to impact the way others in the entire network behave or think [7]. Considerable work has been conducted to model influence or identify influencers in social media [6, 7, 11, 16]. The research proposed in this paper approaches the concept of influencers from an API perspective. We define an API as *influential* if it is widely adopted in mashup and service-oriented application development. The more influence an API has, the more interest that API sparks to developers. We perceive API developers (both consumers and providers) in programming communities as social actors. API consumers use existing APIs to build mashups and service-oriented applications. They share experiences, feedback, and opinions about APIs in various ways such as participating in discussion forums, reporting bugs, and following APIs. API Providers are the developers that created the APIs available in the community. They publish important information about their APIs such as tutorials, articles, SDKs, libraries, new releases, and source code.

Analyzing social content to identify *influencer APIs* has several advantages. First, API consumers will be able to integrate the best-in-class APIs. For instance, the `ProgrammableWeb` directory lists more than fifty mapping APIs. Selecting the right API is time-consuming and may have an impact on future mashup maintenance and development [5]. Second, consumers may have different views on what makes an API relevant. Some consumers may value APIs with the least number of reported bugs. Others may consider the opinions expressed by peers toward the API as significant. Measuring influence based on various API features assists consumers in selecting APIs that are most suitable to their development style, needs, and requirements. Third, identifying influencer APIs enables providers to increase the visibility of their APIs and set up a strategy to reach out a larger audience of developers. Providers will be able to compare their API's influence with the influence of a competitor's API and pinpoint a plan of actions to promote their APIs. Some providers may, for example, decide to enhance their involvement in discussion forums, while others may choose to increase the number of articles and tutorials published about their APIs.

Developing an influencer model for service-oriented environments poses several research challenges. First, social data are scattered across multiple independent platforms and cannot be accurately obtained from one single source. For instance, the number of applications that use a given API is determined by looking at mashups listed in `ProgrammableWeb` and repositories hosted in `GitHub`. Besides, current platforms return API-related data in heterogeneous formats.

For example, posts in **StackOverflow** and commit comments in **GitHub** are textual and require natural language processing techniques. News articles in **HackerNews** and bug reports in **Bugzilla** are presented in proprietary formats. Other data such as the number of issues in **GitHub**, number of posts in **StackOverflow**, and number of followers on **ProgrammableWeb** are returned as atomic values on different scales. Second, the social content collected from existing communities deals with various aspects of the APIs. It covers both technical (e.g., number of SDKs) and non-technical (e.g., number of API followers) information. This includes information about the API itself (e.g., number of change logs representing the API's evolution), API consumption (e.g., number of projects that use the API), and API social activities (e.g., number of posts and articles related the API). A multi-dimensional influencer model that captures various API features is needed. Once API features are gathered and cleaned, it is important to determine the extent to which an API is influential and the features that have or do not have an impact on the API influence and, if so, to what degree. Third, newly developed APIs lack the social content necessary to assess their overall influence. Therefore, recommender systems based on API influence scores may omit to return such APIs. This may lead to the starvation of newly deployed APIs as they lack the required social presence. The proposed influencer model should allow bootstrapping the influence score of newly created APIs, hence overcoming the traditional cold start problem.

The identification of influential nodes in social networks has been the subject of many research efforts [6, 7, 11, 16]. Existing research devoted to influencers in software ecosystems emphasizes on developers as influencers *not* APIs [3, 10]. In this paper, we propose FAME (*inFluencer Apis in developer coMmunitiEs*), an influencer model for APIs in service-oriented environments. To the best of our knowledge, FAME is the first approach to consider APIs (instead of developers) as influencers in building mashups and service-oriented applications. The main contributions of this paper are summarized below:

- We propose an influencer model that extracts more than eighteen API features from multiple programming communities. The extracted features capture non-technical and technical information about APIs in various formats such as text, atomic values, and other proprietary structures. We perform sentiment analysis to quantify developers' opinions towards using APIs. We introduce a cumulative *API Influence Score* (AIS) to assess the influence of APIs in mashups and service-oriented applications. We also categorize APIs into tiers based on their influence scores.
- We predict the evolution of the influence scores of newly deployed and existing APIs using Non Negative Least Square (NLS) linear regression technique. We conduct an analytical study to determine the degree to which each extracted API feature impacts the influence score.
- We conduct experiments on four real-world programming platforms: **GitHub**, **StackOverflow**, **HackerNews**, and **ProgrammableWeb**. We categorize the extracted social content in five data-sets depending on the deployment dates

of the corresponding APIs (between 2005 and 2019). We compute the recall and precision of each data-set. Experiments reveal that the proposed approach can predict up to 87% influencer APIs with 71% precision.

The rest of this paper is organized as follows. Section 2 gives an overview of the proposed approach. Section 3 presents the FAME model for identifying and predicting influencer APIs. We describe experiments to evaluate our approach in Sect. 4. In Sect. 5, we overview related work. We conclude in Sect. 6.

2 The FAME Approach: An Overview







In this section, we give an overview of the proposed approach. We first introduce two scenarios to motivate our approach. Then, we describe FAME architecture for identifying and predicting influencer APIs.

2.1 Motivation

We describe two scenarios that illustrate the benefits and challenges of API influencer identification for consumers and providers. In both scenarios, we refer to two weather APIs: *Aeris Weather* (API_{AW}) and *World Weather Online* (API_{WWO}).

Scenario 1 (API Consumers) - Let us consider a developer, *Mary*, looking for a weather API to use in a mashup. A search on *ProgrammableWeb* returns API_{AW} and API_{WWO} . Since *Mary* has no prior experience programming with those APIs, she turns to developers in various programming communities to help her select the right one. *Mary* first looks at the features of API_{AW} and API_{WWO} on *ProgrammableWeb* (Table 1). Below is a summary of her findings. API_{AW} has more SDKs than API_{WWO} . The two APIs have approximately the same number of articles published on the platform. API_{WWO} has much more followers than API_{AW} . Because *Mary* is interested in mashup development. She learns that API_{WWO} is used in more mashups than API_{AW} . However, she finds only 2 and 11 mashups for API_{AW} and API_{WWO} , respectively. *Mary* then looks at the number of projects on *GitHub* that are relevant to the APIs. She noticed a much larger number of repositories related to API_{WWO} than API_{AW} (39 vs. 6). Similarly, API_{WWO} outperforms API_{AW} in terms of number of Wikis (428 vs. 3). However, less issues are reported by developers about API_{AW} than API_{WWO} (29 vs. 310). *Mary* is overwhelmed by the number of API features published on each platform. She is confused about the features to consider in order to decide about the API to use. She becomes even more frustrated when she parses the long texts posted under the reported issues and commit comments on *GitHub* to get a better idea about her peers' opinions about the APIs.

Table 1. Motivating scenarios

Platforms	Features	AerisWeather API 	World Weather Online API 
	# of Articles	3	4
	# of SKDs	8	1
	# of Library	0	4
	# of Mashup	2	11
	# of Followers	126	271
	# of Changelog	21	19
	# of Repositories	6	39
	# of Commit	9	46
	# of Issues	29	310
	# of Wikis	3	428
	By API name	14,000	42,400,000
	By API EndPoint	1,750	569,000
	# of Posts	4	41

Scenario 2 (API Providers) - The provider of API_{AW} , *John*, performs a Google search on API_{AW} and a competitor's API, namely API_{WWO} . A search by API endpoints returns more than 500,000 additional results for API_{WWO} . A second search by API names returned more than 42 million additional hits for API_{WWO} . *John* is concerned about the significant lack of popularity of his API compared to API_{WWO} . To increase the visibility of API_{AW} and promote its adoption, he looks at some of the features of API_{AW} and API_{WWO} on ProgrammableWeb, GitHub, and StackOverflow (Table 1). The aim is to come up with an action plan to increase the adoption of API_{AW} by developers. The following questions need to be answered as part of *John's* action plan: how does he measure the influence of his API? which features are likely to have a higher impact on developers across programming communities to adopt API_{AW} ? how are the different API features related to each other? which features does he need to improve in order to enhance API_{AW} influence?

2.2 Architecture

The FAME architecture is composed of three modules (Fig. 1): *Unstructured Data Extractor* (UDE), *Structured Data Extractor* (SDE), and the *FAME Model*. UDE extracts and analyzes unstructured (i.e., textual) API features from developer communities. Such features include commit comments in GitHub, posts in StackOverflow, and articles in ProgrammableWeb. UDE conducts sentiment analysis to quantify textual features as positive, neutral, or negative sentiment scores. Since the extraction and analysis of textual features is time consuming, UDE tasks are executed periodically and offline. SDE collects quantitative API features (e.g., number of repositories) from ProgrammableWeb, GitHub, and HackerNews. These features are extracted online (i.e., on demand) during the

execution of an API influencer identification or prediction request. Since SDE-extracted features are measured on different scales, data normalization techniques are applied to adjust those features to a common scale. Once all features are extracted, cleaned, and normalized, the FAME model aggregates those features to determine the *API Influence Score* (AIS) of each API. The calculated scores are used to cluster APIs into tiers: nano (least influential), micro, mid-tier, mega, and celebrity (most influential). The FAME model also uses non-negative-least-square regression to figure out significant features and associate weights to those features. Such weights are used to predict the evolution of AIS scores and tweak API features in order to enhance API influence across communities.

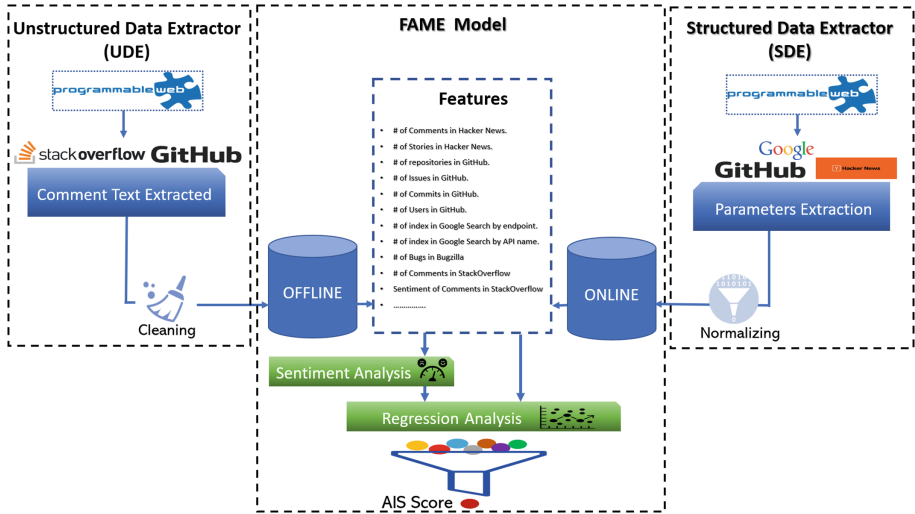


Fig. 1. FAME architecture

3 Influencer Identification and Prediction in FAME

In this section, we give details about the FAME approach for identifying and predicting influencer APIs. We first introduce the methods to extract both unstructured (i.e., textual) and structured features from programming community platforms (Sects. 3.1 and 3.2). Then, we describe the techniques for calculating *API Influence Scores* (AIS) and organizing APIs into influencer tiers based on AIS scores (Sect. 3.3). Finally, we present a linear regression-based model for predicting AIS scores (Sect. 3.4).

3.1 Unstructured Data Extractor (UDE)

UDE evaluates developers’ sentiments toward APIs. It analyzes developers’ feedback and computes scores of interest in APIs. UDE performs three major tasks:

data collection, data pre-processing, and sentiment analysis. The data collection task crawls and collects developers' textual data from `GitHub` (commit comments), `StackOverflow` (posts), and `ProgrammableWeb` (articles). As each one of these platforms exports large volumes of data, we define data extraction patterns based on API names, endpoints, and topics to sort out API-related content and speed-up data collection. The data pre-processing task cleans textual data from irrelevant information such as code snippets and hyperlinks.

Sentiment analysis is the main UDE task. It evaluates developers' opinions toward APIs. We use the Stanford NLP (Natural Language Processing) Parser¹. The parser adopts recursive neural nets, a deep learning technique, to figure out text polarity (positive, neutral, negative). It returns a sentiment score, $sscore \in [-1, 1]$, along with text polarity. An $sscore$ closer to -1 denotes a negative sentiment. The sentiment is considered as positive if $sscore$ is closer to 1. An $sscore$ closer to 0 represents a neutral sentiment.

Some textual features may have higher user views than others. For instance, the sentiment of a post on an API with a large number of views should be given more importance than the sentiment of another post (on the same API) with a smaller number of views. Therefore, we associate a weight w_i for each textual feature f_i . Each weight value w_i correlates to the number of views on f_i . We normalize weights according to the following formula, where $viewsnumber_i$ and $Max(viewsnumber)$ represent the number of f_i views and maximum number of views in all features:

$$w_i = \frac{viewsnumber_i}{Max(viewsnumber)}$$

Using the weight w_i and sentiment score $sscore_i$ of a feature f_i , we define the weighted sentiment as follows:

$$WeightedSentiment_i = w_i * sscore_i$$

Finally, we define the overall sentiment for a given API_j as the sum of the weighted sentiment of each extracted feature f_i divided by the total number of such features:

$$sentiment(API_j) = \frac{\sum WeightedSentiment_i}{TotalNumberOfExtractedFeatures}$$

3.2 Structured Data Extractor (SDE)






SDE collects structured API features from four programming community platforms: `ProgrammableWeb`, `GitHub`, `StackOverflow`, and `HackerNews`. By structured, we mean that the features are well formatted in the platforms and ready to extract/use. We rely on Selenium WebDriver², a framework that automates Web data extraction. The framework allows feature extraction using predefined parsing rules. A rule contains the URLs to load data from and keywords describing

¹ <https://nlp.stanford.edu/>.

² <https://www.seleniumhq.org/projects/webdriver/>.

APIs to filter data with. Data is parsed to extract features using DOM (Document Object Model)³. Table 2 summarizes all API features extracted by UDE and SDE.

Table 2. API features in developer communities

Platforms	Features	Description
	# of Articles	The number of articles related to this API.
	# of SDKs	The number of SDKs available tools for consuming existing APIs.
	# of Library	The number of tools that, when installed, results in the provisioning a specific API.
	# of Followers	The number of users who follow the APIs.
	# of Changelog	The number of change Log which represents the APIs evolution history.
	# of Users	The number of users who mentioned the API name in their profiles.
	# of Commit	Number of commit that have mentioned the API name.
	#of Issues	Number of issues that have mentioned the API name.
	# of Topic	Number of Topics wikis that have mentioned the API name.
	# of Wikis	Number of wikis that have mentioned the API name.
	By API name	The search index when we search by API name.
	By API EndPoint	The search index when we search by API endpoint.
	# of Stories	Number of stories that have mentioned the API name.
	# of Comments	Number of comments that have mentioned the API name.
	# of post	Number of post (question /Answer) that have mentioned the API name.
	Average post view	The average view for all post related to specific API.
	Max post view	The maximum post view related to specific API.
	Average Sentiment	The average sentiment of all post related to API.

Another important factor that helps assess the influence of an API is its spread over the internet. A well distributed/spread API is usually indexed on many search engines, which increases its visibility and eases its access. For example, **Twitter** API is accessible via multiple resources such as tutorials, documentations, and videos. This makes the API more likely to attract developers. We run two kind of queries on **Google** search engine to measure the level of spread of an API. The first query counts the number of entries in the index that contain a given API name; the second query counts the number of entries containing a given API endpoint (Table 2).

The features retrieved by SDE are returned on different scales. For example, the number of issues reported for an API on **GitHub** may reach several hundreds; the number of SDKs available for an API on **ProgrammableWeb** is typically a one or two-digit value; the number of users interested in an API on **GitHub** may go beyond thousands. To normalize features on a common scale, we use the following formula:

$$\hat{x} = \frac{(x_{max} - x_{min}) * (r_{max} - r_{min})}{(x_{max} - x_{min})} + r_{min}$$

³ <https://www.w3.org/TR/WD-DOM/>.

Where: \hat{x} refers to the normalized value; x_{max} and x_{min} refer to the feature maximum and minimum values, respectively; r_{max} and r_{min} refer to the maximum and minimum new range values, 1 and 0 for our case.

3.3 API Influencer Score (AIS)

We define a metric, called *API Influence Score* (AIS), to model the degree to which community members use an API to develop mashups and service-oriented applications. For that purpose, we use the number of mashups and repositories that adopt the API on *ProgrammableWeb* and *GitHub*, respectively. However, some developers may display negative experiences using an API. To capture developers’ opinions, the AIS score includes the average weighted sentiment. As shown in the formula given below, the AIS score is calculated using three of the API features extracted from community platforms (Table 2). The remaining extracted features are used to predict the AIS score as shown in Sect. 3.4. The AIS score is formally defined as follows:

$$AIS_{(i)} = \sum \#M_i + \#R_i + Sentiment(API_i)$$

Where: $\#M_i$ is the number of mashups that use API_i on *ProgrammableWeb*; $\#R_i$ is the number of repositories that use API_i on *GitHub*; $Sentiment(API_i)$ is the overall sentiment on API_i as defined in Sect. 3.1.

Using the computed AIS scores, we define influencer tiers to categorize APIs according to their influence level. Figure 2b shows the five tiers: *Nano*, *Micro*, *Mid-Tier*, *Mega*, and *Celebrity*. Figure 2a depicts the distribution of all APIs across the five influence tiers. The *Nano* tier regroups the least influential APIs. APIs in this tier have a score below 0.015. This category has the highest proportion of APIs, with about 600 identified APIs. Examples of *Nano* APIs are *Blinksale*, *Plunker* and *MyWot*. The *Micro* tier contains APIs with a score between 0.015 and 0.15. *Hoiio Voice*, *Kiva* and *Songkick* are examples of APIs in this category. *Mid-Tier* refers to APIs with a score in the [0.15, 0.5[range such

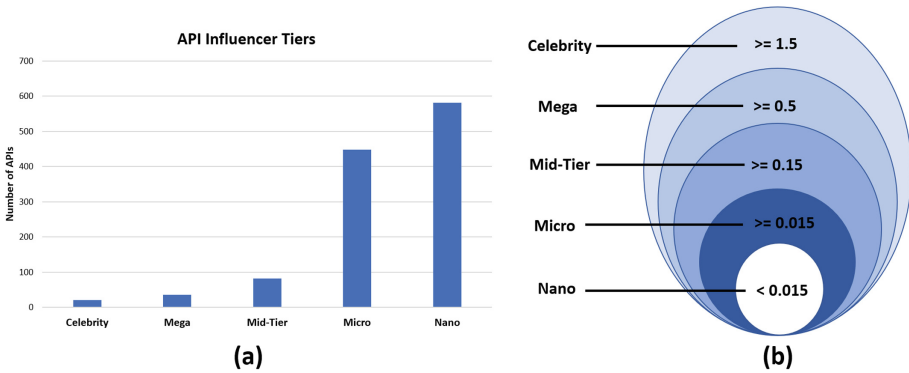


Fig. 2. Influencer tiers

as `LinkedIn`, `Zillow` and `Evernote`. *Mega* regroups APIs with a significant influence score ($AIS \in [0.5, 1.5[)$), such as `Flickr`, `Last.fm`, and `Reddit`. *Celebrity* represents APIs with the highest influence ($AIS > 1.5$). Celebrities appear in the highest number of mashups and repositories. They also subject to positive sentiments among developers. Examples of celebrities are `Google Maps`, `Twitter` and `YouTube`. Figure 2a shows that this tier has the lowest proportion of APIs, with about 20 identified APIs.

3.4 Influence Score Prediction

We compute the AIS score of an API using three features: number of mashups, number of repositories, and overall developers' sentiment. However, it is difficult for API providers to have direct control on those features to improve the adoption of their APIs by developers. To help API providers enhance the influence of their APIs, we conduct a statistical study to identify the most relevant API features that correlate the most to AIS scores. Once API providers understand which of the remaining features (other than number of mashups, repositories, and sentiment) impact the AIS score, they can come-up with a strategy to boost-up the influence of their APIs.

We use Non-Negative Least Squares (NNLS) regression [8] to learn a weight value for each API feature. NNLS assigns weights to features according to their correlation degree to AIS scores. The most relevant features are given high coefficients, while non relevant ones are given negative coefficients. NNLS replaces negative coefficients by 0. This will automatically get rid of non relevant features from the model. Figure 3 summarizes the coefficients assigned to each API feature. Features with the biggest coefficient values have the highest impact on AIS scores. For instance, the number of articles in `ProgrammableWeb` is strongly related to the AIS score. This shows that more articles published in the developer community may increase API influence. Figure 3 also states that `StackOverflow` features have little impact on AIS scores.

The next step is to define AIS prediction models. These models are useful to assign initial influence scores for newly deployed APIs, hence dealing with the traditional cold start problem. They also assist API providers in predicting the evolution of their API scores. We introduce three prediction models (Table 3). To evaluate and compare the models, we calculate the *adjusted R-squared* [14] and *Akaike Information Criterion* (AIC) [1]. The adjusted R-squared estimates the variance between predicted and real scores. AIC measures the goodness of the fit for the model. The model with the smallest AIC value and highest adjusted R-square is selected as the best-fitting model.

Table 3 summarizes our three prediction models. Model 1 uses all extracted features to predict the AIS score. It has a low adjusted R-squared value: 0.5788. Hence the model does not fit the trend perfectly. This is because AIS scores depend on developers' sentiments, which are hard to predict. To deal with this issue, we introduce the *adjusted AIS score* ($AIS_{adjusted}$). $AIS_{adjusted}$ is a variant of the original AIS score that eliminates developers' sentiments. The following formula computes API_i 's adjusted AIS score.

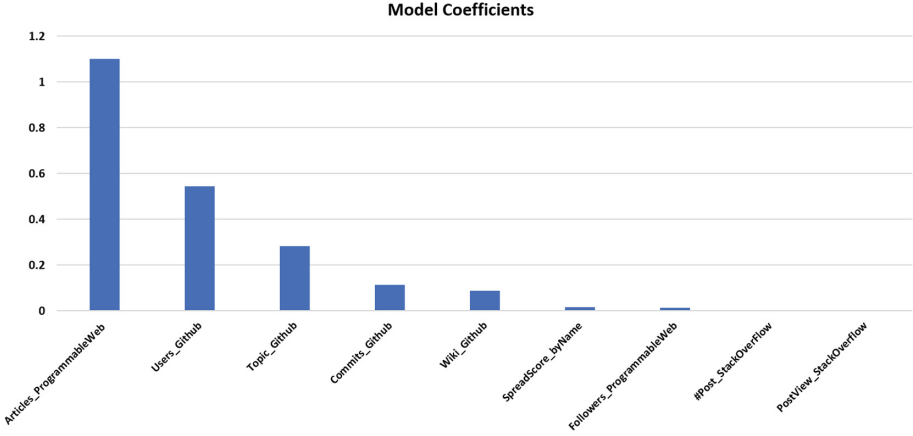


Fig. 3. Impact of API features on the AIS score

$$AIS_{adjusted(i)} = \sum \#M_i + \#R_i$$

The second and third models predict the AIS adjusted scores. Model 3 uses all extracted features. Model 2 omits the features extracted from *StackOverflow* since our study shows that *StackOverflow* has little impact on API adoption across communities (Fig. 3). Both models display high adjusted R-squared: 0.77 for Model 2 and 0.78 for Model 3. The models also have low AIC values: 1346.347 for Model 1 and 1323.208 for Model 2. This makes both models suitable for predicting the adjusted AIS score, with a slight advantage to Model 3 as it uses more API features than Model 2.

4 Experiments

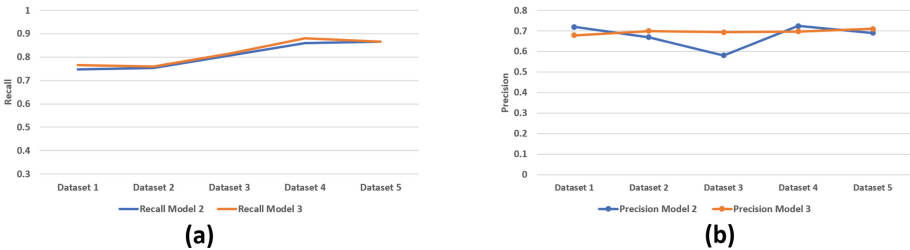
The goal of our experiments is to assess FAME’s ability to accurately predict influencer APIs. We evaluate the second and third prediction models (Table 3) using five independent data-sets. The data-sets regroup APIs deployed during five different periods between 2005 to 2019. For each API, we compute the adjusted AIS score and use the models (2 and 3) to predict that score. We then compute the recall and precision for each data-set using both models.

The recall refers to the fraction of influencer APIs that are correctly identified within each data-set. It is the number of influencers that are successfully predicted divided by the number of all APIs that are identified as influencers. It can be also seen as the percentage of influencer APIs that are successfully predicted. Figure 4a shows that up to 86% and 88% of influencers are successfully recalled (i.e., predicted) by Model 2 and Model 3, respectively. Both models have a stable recall, but leveraging more features in Model 3 allows a better prediction than Model 2.

Table 3. Prediction models

	Prediction	Available Features	Number of Selected Features	Selected Features	Adjusted R-squared	AIC
Model 1	AIS	18 (SDA +UDA)	12 Features	<ul style="list-style-type: none"> ➤ <i>ProgrammableWeb</i> #Articles, #Libraries, #Followers and ChangeLog ➤ <i>Google Search Engine</i> Spread Score by EndPoint ➤ <i>Hacker News</i> #Comments ➤ <i>GitHub</i> #Commits, #Users, #Wiki and #Topic ➤ <i>StackOverflow</i> #Post and Ave Post View 	0.57	4400.073
Model 2	AIS-adjusted	15 (SDA)	7 Features	<ul style="list-style-type: none"> ➤ <i>ProgrammableWeb</i> #Articles and #Followers ➤ <i>Google Search Engine</i> Spread Score by Name ➤ <i>GitHub</i> #Commits, #Users, #Wiki and #Topic 	0.77	1346.347
Model 3	AIS-adjusted	18 (SDA+UDA)	9 Features	<ul style="list-style-type: none"> ➤ <i>ProgrammableWeb</i> #Articles and #Followers ➤ <i>Google Search Engine</i> Spread Score by Name ➤ <i>GitHub</i> #Commits, #Users, #Wiki and #Topic ➤ <i>StackOverflow</i> #Post and Max(Post) View 	0.78	1323.208

The precision checks the accuracy with which scores are predicted for APIs. It is the number of precisely predicted influencer APIs divided by all recalled APIs. It can be seen as the percentage of precisely recalled influencers. If the weight difference is less than a threshold value, the influencer is assumed to be precisely identified. We used 0.03 as a threshold; this value represents the average of the difference between the predicted and computed scores. Figure 4b shows that both models identify influencer APIs with up to 71% precision.

**Fig. 4.** Recall and precision

5 Related Work

The identification of influential nodes in distributed environments such as social networks and forums has been the subject of many research efforts [6, 7, 11, 16]. Few research proposals [3, 10] study influencers in software development. However, existing research considers developers as influencers *not* APIs. [11] proposes

a methodology to identify influencer nodes that are likely to affect other nodes in social networks. It computes the centrality degree of nodes and analyzes node activities. [11] focuses on the position of nodes in the network. Our approach instead leverages both structured (e.g., number of mashups, number of articles) and unstructured (e.g., user feedback) across multiple developer platforms to identify influencers. We also show that influencer identification precision gets better as we leverage a larger number of features. [16] presents a study for finding influential authors on **Twitter** forums. It combines both user profile information and user interaction features with decision tree to identify influencer authors. Our approach identifies APIs as influencers *not* users. Moreover, we use a multi-objective function that combines multiple attributes collected from various sources. [7] proposes a study to understand influencers who lead development and dictate how projects evolve. It shows that analyzing influencer behaviors allows understanding the evolution of software ecosystem and even predict future evolution. The main focus of our approach is to identify influencer APIs and the attributes that contribute to their emergence, rather than assuming the existence of those influencers and studying their behavior. [2] shows that influence score depends on engagement, sentiment, and growth. [6] shows that originality and uniqueness of user content are crucial factors to identify influencers in **Instagram**. [2] and [6] rely mainly on social metrics to determine influencers. Our approach extends the analysis to encompass attributes from various sources. Besides, it considers APIs as influencers in programming platforms instead of users in social networks. [17] computes influence score for users across several social networks. It evaluates the quantity and quality of reactions a user action prompted to assess the extent to which the user is influential. [3] and [10] identify the most influential developers, repositories, technologies and programming languages in **GitHub**. [3] shows that the analysis of social networks, particularly the relations among developers, developers and repositories, and developers and followers helps identify developers' influencer index. [10] proposes an approach to measure user influence in **GitHub**. It analyzes relationships between users, as well as between users and projects. In contrast to our approach, [10] and [3] are restricted to **GitHub** and **stackoverflow** data. In our approach, we show that using a multi-objective function that combine both structured and unstructured features from diverse platforms substantially enhances the precision of the influencer identification process. We also introduce models to predict the evolution of influencer scores for newly developers and existing APIs.

6 Conclusion

We propose FAME (*inFluencer Apis in developer coMmunitiEs*), a novel approach for the identification and prediction of influencer APIs in service-oriented environments. To the best of our knowledge, FAME is the first influencer model that treats APIs as first-class citizens. We define influence as the degree to which an API is used in mashups and service-oriented applications. We extract and analyze several structured and unstructured features from various programming communities. We use the Stanford NLP parser to perform sentiment analysis and

evaluate developers' opinions towards using APIs. Such opinions are expressed through posts in `StackOverflow`, commit comments in `GitHub`, and articles in `ProgrammableWeb`. We aggregate API features to compute influence scores and cluster APIs into influencer tiers according to those scores. We use Non-Negative Least Square (NNLS) regression to identify the most significant API features and predict the evolution of influence scores for newly deployed and existing APIs. Finally, we conduct extensive experiments on real-world and large datasets extracted from multiple programming community platforms. Experiments reveal that the proposed approach predicts up to 87% influencer APIs.

References

1. Akaike, H.: A new look at the statistical model identification. In: Parzen, E., Tanabe, K., Kitagawa, G. (eds.) *Selected Papers of Hirotugu Akaike*. Springer Series in Statistics (Perspectives in Statistics), pp. 215–222. Springer, New York (1974). https://doi.org/10.1007/978-1-4612-1694-0_16
2. Arora, A., Bansal, S., Kandpal, C., Aswani, R., Dwivedi, Y.: Measuring social media influencer index-insights from facebook, twitter and instagram. *J. Retail. Consum. Serv.* **49**, 86–101 (2019)
3. Bana, R., Arora, A.: Influence indexing of developers, repositories, technologies and programming languages on social coding community github. In: 2018 Eleventh International Conference on Contemporary Computing (IC3), pp. 1–6 (2018)
4. Binzagr, F., Medjahed, B.: Crowdmashup: recommending crowdsourcing teams for mashup development. In: Pahl, C., Vukovic, M., Yin, J., Yu, Q. (eds.) *ICSOC 2018*. LNCS, vol. 11236, pp. 679–693. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-03596-9_49
5. Bouguettaya, A., et al.: A service computing manifesto: the next 10 years. *Commun. ACM* **60**(4), 64–72 (2017)
6. Casaló, L.V., Flavián, C., Ibáñez-Sánchez, S.: Influencers on instagram: antecedents and consequences of opinion leadership. *J. Bus. Res.* (2018)
7. Farias, V., Wiese, I., dos Santos, R.P.: What characterizes an influencer in software ecosystems? *IEEE Softw.* **36**(1), 42–47 (2019)
8. Flammarion, N.: Stochastic approximation and least-squares regression, with applications to machine learning. PhD thesis, École Normale Supérieure, Paris, France (2017)
9. Gao, L., Yue, W., Xiong, X., Tang, J.: Discriminating topical influencers based on the user relative emotion. *IEEE Access* **7**, 100120–100130 (2019)
10. Hu, Y., Wang, S., Ren, Y., Choo, K.-K.R.: User influence analysis for github developer social networks. *Expert Syst. Appl.* **108**, 108–118 (2018)
11. Kim, E.S., Han, S.S.: An analytical way to find influencers on social networks and validate their effects in disseminating social games. In: 2009 International Conference on Advances in Social Network Analysis and Mining, ASONAM 2009, Athens, Greece, 20–22 July 2009, pp. 41–46 (2009)
12. Labbaci, H., Medjahed, B., Aklouf, Y., Malik, Z.: Follow the leader: a social network approach for service communities. In: Sheng, Q.Z., Stroulia, E., Tata, S., Bhiri, S. (eds.) *ICSOC 2016*. LNCS, vol. 9936, pp. 705–712. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46295-0_50

13. Liu, X., Kale, A., Wasani, J., (Cherie) Ding, C., Yu, Q.: Extracting, ranking, and evaluating quality features of web services through user review sentiment analysis. In: 2015 IEEE International Conference on Web Services, ICWS 2015, New York, NY, USA, 27 June–2 July 2015, pp. 153–160 (2015)
14. Miles, J.: R squared, adjusted R squared. Wiley StatsRef: Statistics Reference Online (2014)
15. Noor, T.H., Sheng, Q.Z., Ngu, A.H.H., Dustdar, S.: Analysis of web-scale cloud services. *IEEE Internet Comput.* **18**(4), 55–61 (2014)
16. Purohit, H., Ajmera, J., Joshi, S., Verma, A., Sheth, A.: Finding influential authors in brand-page communities. In: Sixth International AAAI Conference on Weblogs and Social Media (2012)
17. Rao, A., Spasojevic, N., Li, Z., Dsouza, T.: Klout score: measuring influence across multiple social networks. In: 2015 IEEE International Conference on Big Data (Big Data), pp. 2282–2289. IEEE (2015)