Recommendation of APIs for Mashup creation

Priyanka Samanta (ps7723@cs.rit.edu)
Advisor: Dr. Xumin Liu (xl@cs.rit.edu)
B. Thomas Golisano College of Computing and Information Sciences
Rochester Institute of Technology, Rochester, New York, USA

ABSTRACT
The aim of this project is to improvise an existing recommendation system that recommends relevant APIs for creating a mashup. This system would recommend a list of relevant APIs based on the API’s functionality, its usage history and popularity. The system takes a user input in the terms of mashup description and then runs through the existing database of mashups and APIs to recommend the list. To improvise the existing the system, advanced topic modelling and collaborative filtering is used to leverage the functionality of API and its usage history respectively.

1. INTRODUCTION
1.1 Problem Introduction
With the advent of Web 2.0, creation of web mashup has gained importance and popularity. Today web is more about a repository of services rather than contents. As the term suggests, mashup refers to a composition created by blending multiple features into one. Web mashup refers to a web application that blends one or more web services to create a new service with enriched functionality and visualization. For eg. Mapskrieg is a web mashup that combines the services from Craigslist and Google maps to help end users look for housing with a map interface.

Mashup creation from existing Web APIs is a boost to the social software as it is created rapidly, involves software reusability and requires minimal technical skills. Today the repository of Web APIs has crossed 18000. With this large dataset and unstructured description of the APIs available, it is very challenging to look discover the relevant APIs from the giant growing repository of APIs online. To address this problem, many solutions have been proposed to recommend relevant APIs for mashup creation, that is primarily based on API functionality, usage history of APIs in existing mashups and popularity of the API. This paper is aimed to exploit advanced machine learning algorithms that would improvise the performance of the existing recommendation system that incorporates all these features in recommending list of relevant APIs.

1. There has been an unstoppable growth in the field of machine learning. To exploit the functionality of the APIs in creating the recommendation system, Hierarchical Dirichlet Process (HDP) has been used. This is advanced algorithm of probabilistic topic modelling that address few of the shortcomings of Latent Dirichlet Allocation (LDA). HDP is used to represent the existing APIs and Mashups as the distribution of the latent topics and later use a similarity measure to compute how similar the user specification of the mashup is to the existing mashup.

2. To leverage the usage history of APIs in the existing mashup, advanced version of matrix factorization has been used. This forms an integral part of collaborative filtering. This helps to provide a relevant list of APIs for a user input of mashup based on its usage history. This system can efficiently handle the cold start problem as well. Probabilistic matrix factorization helps to overcome few of the drawbacks of the traditional matrix factorization used in collaborative filtering.

3. Following, Bayes Theorem is applied to combine the result obtained from (i) and (ii) and the final list is recommended based on the popularity of APIs. Popularity acts as a Quality of Services (QoS) measure for the APIs.

1.2 Background and Definitions
- API - Web API (Application Programming Interface) is a web service that works on HTTP requests and responses. It is a method that gets called by HTTP requests (programs) to get the results back. It is software system that supports communication between machines across the network.
- Mashup - Mashup is a web application that blends one or more web services together. It gives enriched functionality in a new GUI. Mashups are easy to build and requires minimal technical exposure.

Following paper has been organized under sections as following: Section 2, provides the relevant work done in this field. Section 3, outlines the approach to implement this project. Section 4, details the implementation of the system followed by Section 5, 6 providing the results and the evaluation of the system respectively. Finally this work is concluded in Sect 7.
2. RELATED WORK

In this section, a literature review is presented based on the study conducted in the field of this project. Several papers and works have been reviewed that details the work done for the existing system along with few that talks about the advanced machine learning algorithms that would help to improve the existing system and build this project.

2.1 Functionality based Recommendation using tagging

The authors of [5] came up with the novel approach of recommending mashups by finding the user’s interest from their usage history of the services and developing a social network exploiting the relationships among mashups, web APIs and tags. According to the author the existing recommendation systems based on semantics, QoS (Quality of Services) and social network does not take into consideration the interests of the users in the services. The paper contributes by formulating the new system incorporating the user’s interest and the social relationships among the services. The dataset used in the experimentation was ProgrammableWeb. “The data consisted of 6076 mashups, 4492 Web APIs and 1736 tags”[5]. And the experiment proved to provide efficient and effective results than the existing systems in service discovery. The authors came up with 2 models, User Interest Model and Social Network Model in order to leverage the user’s interest and social relationship respectively. User Interest Model: TF/IDF (Term Frequency/Inverse Document Frequency) technology was employed to build the User Interest Model. This technology was used to determine the weightage of the terms in the corpus. Corpus here refers to the group of the description documents of the available mashups. These description documents consists of the details like name of the mashup, its functionality details, the web APIs used, the names of the developers and the tags. At first, the term frequency is calculated by determining the frequency of the ith term present in the jth mashup description document. Further it was normalized as the length of the description documents for the available mashups may vary. TF was calculated using the formula:

\[ TF = \frac{\text{Frequency of the } i^{th} \text{ term in the } j^{th} \text{ document}}{\text{Total number of terms present in the } j^{th} \text{ document}} \]

Secondly, IDF is used, to measure the degree of commonness of the ith term across all the mashup description document present. It is calculated using the formula:

\[ IDF = \log \left( \frac{\text{Total mashup description document in the corpus}}{\text{Number of documents where the } i^{th} \text{ term appears}} \right) \]

TF/IDF is calculated as the product of these terms as mentioned above. This formula was deployed on the user’s usage history of the mashups. The description documents of all the mashups which were used by user were combined to create a corpus. On the same TF/IDF was used to convert into vector of term weights. Similarly the vector of the mashup description was also formulated from the given mashup document. Followed by this, cosine similarity was applied to compute the similarity between user’s interest based on usage history and a given mashup service description.

**Social Network Model:** This model was built using the relationship existing between mashup and web API, and between mashup and tags, in three stages. Mashups are created using one or more Web APIs. Further a tag is a keyword assigned to an information, here assigned to a mashup or a web API that describes its features and functionality. So the relation between API and mashup can be termed as “invocation relationship” [5] and the one existing between tags and API or mashup can be coined as “social marking relationship”[5]. Similarity between the 2 mashups is determined using the invocation relationship and the social marking relationship. That is 2 mashups are said to similar to each other if they invoke similar APIs or if they are marked with similar tags. This is computed using Jaccard coefficient. Social Network is constructed in the form of undirected graph. This graph contains the mashups as its nodes, which are connected with edges, only if they possess similar tags and invoke similar APIs. The edges are marked with weights that defines the degree of similarity between the mashups -eq(2) Based on the User Interest Model and Social Network Model, mashup service is recommended by determining its similarity with similar mashups using the formula:

\[ a \times eq(1) + b \times eq(2) \ldots (a + b) = 1, (0 <= a, b <= 1) \]

After this model was developed, it was experimented on the ProgrammableWeb dataset as mentioned above. And it was observed that the author proposed model outperformed the normal process of service discovery based on user’s relevance search or social network based search. The accuracy of the traditional method for finding mashups was measured using DCG (Discounted Cumulative Gain), an algorithm to evaluate the information retrieval over the web. To conclude, this paper proposes a novel approach to recommend mashup to users based on the usage history of mashups by the users and the social relationship existing between the mashups and web APIs and tags.

The recommendation system can further be improved by exploiting the semantic meanings from tags[9], describes the approach of deriving semantic meaning from tags that helps in building “data-driven mashup”[9]. Tags are very powerful hierarchical keyword, that when attached to a piece of information, effectively describes the information. The authors of this paper, foresaw some of the existing problems in building a mashup, which were: (1) the need of developers for fast discovery of the services, as they build mashups to address situational requirements with minimal program-
ming skills, (ii) an efficient way to look for services as repository of the services has been growing exponentially. For addressing these problems, the authors proposed building up data-driven mashups based on semantic annotations of tags. The concepts of tags gained its importance from the use in web resources. With the help of tags, it becomes easy to discover a service and also to get an insight on the service and build up a mashup. Few features that the paper emphasizes on are: (i) Extracting the semantics from the tags and associating with the services input and output, (ii) data centric development of the mashup, requiring no details on the underlying programs.

The solution that the authors mentioned to solve the problems as mentioned above, is built with 3 major components: (i) a service repository, (ii) a service advisor and (iii) a user interface. The service repository stores all the web APIs and also allows publication of the same. APIs contains the tag information. Using several machine learning techniques, these tag information is extracted and semantically similar tags are grouped into clusters. The role of the service advisor is to facilitate the composition of the services to build up the desired goal based on the linking between the tags. Finally the user interface which is used to access the service.

As a part of the solution, the authors came up with the “Tag based service semantic mode” [9]. This model was built up based on the fact that the end-users and the developers build up mashups looking at the data consumed (input data) and the data generated (output data) from the web APIs. Thus set of tags are associated with the input and output data that makes the discovery of the services by the developers easy. But the use of tags are not accurate always. Its limitation lies in the fact that, similar tags may be used by the human to define different information or similar information may be tagged with different keywords (tags). To overcome this situation, an unsupervised learning technique was exercised to extract semantically similar meanings from tags and put them in cluster. For example the tags like apple, orange, banana can all be put in the cluster A fruit. This gave rise to the concept of “Feature Tag” [9], using which it can that represent all other similar tags from the cluster. Also there came in the idea of tag taxonomy, where if a tag ‘a’ expressed semantically higher notion than tag ‘b’ then tag ‘b’ was named as the sub tag of tag ‘a’. Using this tag based service semantic model, the author proposed, the design of data driven mashup composition using web APIs. Two web services, web service-1 and web service-2 are eligible to be composed together if the output data: tag ‘x’ of web service-2 is consumed as the input data by web service-1 or if web service-1 consumes tag ‘y’ where tag ‘x’ is the sub-tag of tag ‘y’. These two web services are then said to be mapped semantically with tags. Once the input and output data has been associated with tags, this model would facilitate the end-users to develop their mashup with ease, if they explain their desired goal in the form of tags which would make the discovery of the web services containing semantic tags, easy and effective. The experimentation with this model was conducted on “200 web services, 3100 mashups and 23,971 tags” [9]. As a result it was observed that, around 80% of the developers achieve their goals of service creation in top 3 list of recommendation. It may be concluded that the idea of using semantic-based tags in developing data-driven mashups proved to be efficient in providing the output (in terms of web services) needed by the developers to build their product. [13] Provides a similar approach to look for similarity between documents but it exploits the use of semantic hashing of tags. The paper proposes a new model of semantic hashing extracted from tags and topic modeling. This model is a solution to the limitations of the existing hashing methods. These are: (i) tags are widely used in the real world application to get associated with a piece of information, but it has not been exploited completely, (ii) the similarity based on keyword matching fails to reflect the semantic similarity existing among the keywords. The model proposed by the author incorporates both the usage of tags and probabilistic topic modelling to determine the keyword similarity.

With the growth of internet and increase in web resources in the terms of images, texts, data, audios and videos present, it has become significantly important to look for efficient techniques to determine similarity of search. When working with the large datasets, usually we face 2 challenges: (i) storing the large scale data and (ii) retrieving the data from this large dataset, making the application of traditional text similarity difficult. To address these limitations, semantic hashing has proved to be effective. It works by transforming each document to binary codes thus making similar documents to be mapped to the similar binary codes. Given the query document, it can also be transformed into binary codes and then Hamming distance may be computed. This is to determine the similarity existing between the documents. Thus this process is effective, involving low cost as the large dataset can be stored easily in the memory and the retrieval process would also be fast. Semantic hashing though capable of handling this problem, but it has the limitation as mentioned above. Thus the author proposed a new approach, Semantic Hashing using Tags and Topic Modelling (SHTTM). As a part of solution, a framework is built, to ensure that the hashing code of tags are consistent and the semantic meaning of the documents are preserved by using topic modelling. Experiment performed on the real world dataset with this solution proved to be far better than the existing methods. Paper sheds some light on various hashing techniques too. These are:

1. i. Semantic Hashing (SH): It uses fixed number of bits to represent documents in binary code. Thus it take very less time for computing the similarity between documents.

2. ii. Locality Sensitive Hashing (LSH): This is one of the frequently used hashing techniques, which maps the high dimension data, present in the Euclidean space to the low dimension by using random linear projections.

3. iii. Self-Taught Hashing (STH): This hashing technique is efficient than SH and LSH. But it is unable to address the 2 major limitations: (i) exploiting the use of tags and (ii) does not preserve the semantic similarity between the documents. This hashing mechanism uses the combination of supervised and unsupervised learning techniques to meet the goal. During unsupervised learning, using a fixed number of nearest neighbor it build up a similarity graph. Following, it adds the documents to the k dimensional space and converts them to binary codes.

4. iv. Semi-Supervised Hashing (SSH): This hashing technique is used to find out the similarity between a pair of
documents. While determining the similarity, it does make use of the tag information. If the documents share a common tag, then they are termed as Must-Link else termed as Cannot-Link. Using this concept, this hashing method generates the binary hashing codes, which are ought to be closer to each other for the Must-Link documents and further for the Cannot-Link document pair. Unfortunately, this method fails to work, when the tag information is missing.

Finally the author proposes the SHTTM approach which consists of 2 components, (i) “Tag Consistency component” [13], (ii) “Similarity Preservation component” [13] Tag consistency component is needed to get an insight on the tags associated with the information. It is essential to determine the correlation existing between the hashing code and tags (by using matrix factorization model) and also a way to determine the tags in case they are missing (by using confidence matrix), which is not rare in the real world scenario. Similarity Preservation component is needed to maintain the semantic similarity between the documents. If the documents are semantically similar, they must be mapped to similar codes, making the Hamming distance between the two minimal. This is handled using topic modelling that measures the similarity across the documents instead of the "features from original keyword" [13]. Using topic modelling, the documents can be reduced to few topics, thus making it easy for the similarity search Experiments conducted with this model on various real world data which suggests that this approach generates more accurate hashing codes than existing methods and it also showcases the benefits of incorporating hashing based on tag and preservation of semantic similarity between documents.

2.2 Recommendation based on social network and QoS using semantic similarity

[11] Suggests that an end user’s service composition has mashup as its important theory of support. In order to support the end users based on the social interactions capture and analysis, the author of this paper provides a new concept. To feed a service recommendation construction the authors concentrate on getting useful information extracted from social networks. The approach is based on social networks, which develops interactions based on a variety of a particular interest. This paper further provides recommendations obtained from social networks that makes use of implicit social interaction analysis. This social graph is an inference from the common composition interests of users.

The paper mainly focuses on defining the social structures in the services area and how these structures could be influenced to simplify the problems involved in SOA (Service Oriented Architecture). To address this issue the authors propose a framework known as Social Composer (SoCo) mainly for the composition and discovery of the services. The protocol mainly transfers the user-service interaction to user-user interaction where statistical processes are applied to provide recommendations in helping a user to construct a mashup. Like mentioned above the implicit social relation is obtained based on the activities of the users.

Experiments were conducted for evaluating the performance and the behavior of the framework proposed in the paper. These experiments showed that the mashup size is the main parameter to be considered and that it has a linear relation with the response time formaking a recommendation for a service. Two random datasets were generated with uniform statistical distribution. It was found that the response time is not all that important in the case of long-tail than that of uniformly distributed dataset. The results obtained through this research proved to be useful in interactive applications. But a major disadvantage is the fact that the algorithm response time is dependent to the mashup size that happens to be distributed among two and five services. Thus a new approach for service recommendation for mashup based on social network analysis was provided. Though the use of the SoCo framework has showed promising changes, there are certain challenging issues such as algorithm behavior based on the deficiency in learning information and auto completion of mashup is to be addressed yet.

[14] States that the existing recommendation system works by finding semantic similarity between the requirement description from the user and functional details of the API provided. Few models do work on the basis of QoS (Quality of Services) or on the basis of social relationship found between APIs and mashup or on the user interest model. But the authors pointed out 2 limitations that exist in the present in the recommendation system. These are: The services are recommended and ranked to the users for mashup creation but they are not categorized. Hence it becomes difficult for the users to understand the ranking of a specific API in a particular category of service. As the current recommendation system do not provide any insight on the categorization of the services, it becomes cumbersome for the users to look for the best service suited for their development task. To work with these limitations, the authors built up a three step model.

1. kMeans variant clustering algorithm: Unlike the traditional clustering algorithm. This variant algorithm first looks for the relevant service in every category and then clusters the remaining services based on their functionality. This helps in laying the basis of ranking based on categorization.

2. Service category relevance ranking model (SCRR): This models helps in explicit identification of the categories from the given mashup description. Further this is achieved by Category Topic Matching (CTM) and Category Affinity Propagation (CAP). CTM plays an important role in looking for the best categories given the functional requirement of mashup, using probabilistic model to find functional relevance of mashup description to every existing category. With the output obtained from the CTM, CAP helps in finding more relevant services based on collaborative filtering based on usage history of the services.

3. Category-aware Distributed Service Recommendation model: The distributed framework of this model, helps in finding the ranks of the services within categories. Thus this model helps in solving the problem of getting a list of ranked services consider to category separation.

The working model has been developed in 2 parts, a clustering model that works offline and a recommendation system that works online. Using clustering, the relevant categories are determined in the offline part. The existing mashups
and the services are clustered in categories based on the usage history. In the core of this process, LDA is performed between the mashup feature matrix and the service feature matrix. As a result an initial base of recommendation system gets formulated in this mode. In the online portion, when the user requirement of mashup creation is received, it is converted into description of feature matrix and compared against the existing services’ feature matrix. Following which the SCCR model is used to determine the relevant ranked APIs in different categories. Finally a category aware distributed recommendation system is built taking help from the ranking services obtained previously. After this model was built, the experiment was carried on the ProgrammableWeb dataset. The dataset consisted of 6,813 mashups and 7,186 services. For the performance metrics, Normalized Cumulative Gain (NCG), Mean Average Precision (MAP) were used. The model outperformed the existing recommendation system and proved to be efficient by showing 30% improvement on the accuracy rate.

2.3 Recommendation tools and algorithms

Papers reviewed under this section helps to get a better understanding on the advanced machine learning technologies present in the field of LDA and collaborative filtering. [7] provides some idea on Collaborative Filtering and how it should address the dynamic exponential growth of data. Collaborative filtering is one of the techniques that is used in building the recommendation systems. It builds the model based on user’s history of usage and users pattern of usage. Collaborative filtering allows reducing the error rate that may exist whilst trying to predict the hidden structure in the user’s ratings. An ideal recommendation system should be capable of providing users choice even when the data grows dynamically. But this demands constant tuning of the parameters. As predicting user’s ratings over a growing dataset is difficult. With the growth in data, there comes in a constant change in the number of user’s and the number of available ratings. To address this problem, the authors of this paper suggested Collaborative filtering to be a time dependent method that is built on kNN algorithm. This method outperforms the existing algorithm with when “user-per neighborhood” [10] sizes are added and updated to this approach. This method was experimented with several large datasets, and it proved to outperform existing methods by making use of sequential classification in the growing dataset using kNN.

After learning about the collaborative filtering, [10] helps in exploiting the matrix factorization method of recommendation system that exploits the structural semantics of an information. Matrix factorization is an effective method because it allows to find out the latent features underlying the interactions between users and items. Additionally, it just a mathematical tool. As the problem statement, the paper defines that very few matrix factorization model takes into consideration the structure of the content. For example users may express their preference over different kind of cuisine like Indian, Thai, Chinese, Continental and many more. But each of these cuisines can be sub divide in several categories. Making it essential to know about the hidden structural content to predict user’s interest model. The author predicted a model to improve the efficiency of the matrix factorization with the help of graph regularization on latent users. There are two known approaches for extracting the hidden structures, (i) using graph partitioning theories to formulate the permutation view and (ii) using regularization techniques to perform feature selection. The modern approach is to incorporate more information, in terms of structure of the item, social network along with the available resources. One of the structural constraint proposed by the author is Laplacian constraint. This constraint prevents from gaining an idea on the community structure of the latent users while using matrix factorization. But with the model, the authors suggested that with this advanced method of collaborative filtering, the structural constraints can easily be exploited. Additionally with Collaborative filtering, some research was on hierarchical topic modelling. [2] And [4], helps in understanding about hierarchical topic modelling with a real world example of nested Chinese Restaurants. This algorithm works in a way of assigning every word of a document to k topics. Following it treats the cluster of k topics with the group of words as a synthetic document and recursively looks for assigning words to the sub topics within the synthetic document. This helps in building a top down hierarchical model of topic modelling. This idea would be utilized in the upcoming milestone to focus on improving the existing LDA which is used to find the similarity between mashup descriptions and the users input to the system.

In a nutshell, [6] expresses the overall flow of the project, what is intended to work with but with improved algorithms that helps to increase the accuracy of the existing recommendation system. This paper suggests that, existing approach for recommending Web APIs is based on functionality of APIs, QoS (Quality of Services) or social network. Each of these approaches focuses mainly on one aspect. (i) Functionality based approach provides the list of relevant APIs by using the API descriptions, semantic markups, tags, API categories and topic models [5]. But the obtained list of these relevant APIs may have less QoS value. (ii) Additionally QoS based approach leverages the QoS value to find the relevant APIs. Again this approach needs the user input on the functionally related APIs to evaluate its corresponding QoS, which is troublesome given the large dataset. (iii) Finally social network based approach makes use of the networking among mashups, APIs and users. This too demands user information, which may be unavailable many a times. Overcoming the limitations of the above mentioned approaches, this paper comes up with a novel idea to recommend APIs for mashup creation using mashup’s textual description as the input. This approach uses the integration of functionality of API, usage history of API, and its popularity to estimate the recommendation relevant APIs. This well written paper demonstrates the advanced work conducted by its authors relating to the previous work done in this domain. The steps taken to solve the addressed problem can be briefly enumerated as below:

1. API discovery based on functionality: LDA (Latent Dirichlet Allocation) is used to identify functionally relevant APIs given the textual description of the mashup. The LDA model is trained based on topic modelling on existing APIs textual description. This trained model is used to find the topic distribution from the provided input of mashup description. Following, cosine similarity is performed between these two topic distributions and functionally relevant APIs are determined.
2. Adding additional API to the search space: To overcome the limitation of topic modelling in absence of rich textual description of APIs, collaborative filtering technique is applied to identify the historical usage of APIs in the existing mashups.

3. Application of the Bayes theorem: Bayes theorem is applied on the output of Step 1 and Step 2 to compute an API’s posterior probability that is being used in the new mashup. This step generates a list of top relevant APIs.

4. Popularity based API ranking: In this final step, the relevant APIs as obtained from Step 3 are ranked based on QoS, a nonfunctional property. It is assumed that the APIs with higher QoS are the ones that are most frequently used in the making of existing mashups. The output generated from this step is the solution of the addressed problem.

To conclude, this paper demonstrates the experimental results of the existing approach and the proposed approach. The proposed model generated a better performance with an increase in recall and precision over the existing model. But it leaves behind a scope to further improve the performance.

3. DESIGN

This section details the design of the proposed system. The system has been designed in 3 different phases that exploits the below features to build up the hybrid system.

1. Functionality of API
2. Usage history of API
3. Popularity of API

3.1 Phase 1: Topic Modelling: Exploiting API functionality

This phase concentrates on finding the relevant APIs from the user’s input based on the functionality. We use HDP algorithm in this phase.

Before understanding the working model of the hierarchical topic modelling, here is what topic modelling refers to. Today the amount of information available online is growing exponentially. And when it is to look for relevant information from the huge online resource, the operations performed are search and links. With this, the search engines returns relevant information from the online repository and helps in interactive interaction with the resources. It is manually impossible to mark or group documents based on themes and topics that would help in efficient and relevant retrieval of the documents. Instead the approach probabilistic topic modelling is used. Topic modelling is a statistical algorithm that explores the documents, analyzes the words in the original documents and discover latent topics from them, and their inter relations with each other. It helps in finding the main theme prevalent in the original documents and organize the original texts based on the structure of themes explored. With the growth in the information, and increasing requirement to retrieve the information efficiently, there has been development in the field of topic modelling, formulation of various topic modelling algorithms. In the base of all advanced topic modelling lies, LDA (Latent Dirichlet Allocation). LDA is a statistical model that tries to find the latent intuition portrayed in the documents. It is a generative process of finding out topics hidden in the documents. Topics are defined to be the distribution over a fixed set of words or vocabulary created from all the documents in consideration. The model assumes that this vocabulary is present prior to the start of the process of finding hidden topics from the documents and also that every document has the presence of multiple topics hidden in it. Following it performs a 2 step process: (i) initialize random distribution over topics and (ii) for every word in the documents, it randomly assigns a topic as obtained from step (i) and randomly select a word from the vocabulary that corresponds to that topic [3]. The main objective of topic modelling is to find the topics given a collection of documents (corpus). Every original document of the corpus has its own hidden structure of the distribution of the topics in it which Topic modelling (LDA) tries to dig out. LDA performs the analysis of the word using joint probability or posterior probability that helps in computing the distribution of hidden variables based on the observed variables. To mention, the words in the documents constitutes to the observed variables in and the latent topic structure in the same are considered as the hidden variables by LDA. Few terms which are used in LDA are (i) \( \beta \) that gives the distribution of topic over the fixed vocabulary, (ii) \( \theta \) that gives the proportion of a topic in a document, \( \theta_{ij} \) gives the proportion of the \( i^{th} \) topic in the \( j^{th} \) document, (iii) \( z_{nj} \), the assignment of the \( n^{th} \) word of the \( j^{th} \) document to a particular topic.

In reality, the number of topic structures possibly hidden in the documents can be exponentially large enough. For the same, several approximating algorithms have been developed, by taking into consideration the alternative distribution of words over the hidden topic structure in the documents. One such commonly used algorithm is sampling based algorithm \( \tilde{\text{AS}} \) Gibbs Sampling. In Gibbs sampling, a sequence of random variables are generated that are dependent on the previous variable in the sequence. This sequence generated is known as Markov chain. This enables the algorithm to approximate the distribution of topics by collecting samples from the limited distribution of words. In this project, the advanced topic modeling algorithm, HDP (Hierarchical Dirichlet Process) is used that uses both Gibbs sampling and LDA as its base.
**Working principal of HDP:** Hierarchical Dirichlet Process is a non-parametric Bayesian clustering algorithm. This algorithm aims to cluster groups of data possessing mixture model. It works on the assumption that certain components or features are shared across multiple dependent data groups, from which it tries to infer the shared properties effectively and generalize new group models. The algorithm assumes that each data group consist of n number of data points with mixed property. Now when different data groups exhibits varied features in various proportion, we can use the same combination of mixture proportion to discover statistical strengths that is being shared across multiple groups to generalize new groups. It considers 2 important parameters: (i) Base distribution that tells about the prior distribution of over data items and (ii) concentration parameters that tells about cluster numbers and the existing sharing amount among the groups. This approach when used in the field of topic modelling proves to be very effective. In the field of topic modelling, the data group is analogous to document that consists of distribution of words in it i.e N number of data groups or documents consists of M number or data points or words in it. Further the data group or document exhibits mixture of features i.e it contains various topics in mixed proportions. Topics are the resemblance of the clusters. Therefore, HDP looks to explore the shared features among the various data groups and generalize new topics that are unbounded given the set of documents. For this system formulation, HDP topic modelling approach is used to look for relevant APIs given mashup description based on APIs functionality.

**Algorithm:**

1. An appropriate initialization of a non-parametric prior. This is referred as the Hierarchical Dirichlet Process.
2. This prior is gradually used in the grouping of the mixture models as stated above.
3. \( \theta \) and \( \beta \) are defined a measurable space.
4. Set of random probability is distributed over the random measurable space.
5. Random probabilistic measure is assigned to each of the mixture model group termed as \( G_j \).
6. \( G_0 \) is the defined global probabilistic measure. It is distributed with concentration parameter \( \gamma \) and base measure of H.
7. \( G_j \) is conditionally independent given \( G_0 \).
8. The base probabilistic measure H has varied distribution of global measure that is ruled by the concentrated parameter.
9. The actual distribution \( G_j \) of a group varies from the global measure \( G_0 \). This is governed by the concentration parameter.
10. The varied deviation in each of the group’s distribution is represented by \( \alpha_j \).
11. Each group is assigned with the factor \( \theta_{ij} \), that corresponds to a single observation \( x_{ij} \).

Incorporating in the system: Using this HDP, the topic distribution of the APIs and mashups are computed. HDP contains of the training and testing phase. The model is trained using the name and description of the APIs. After the model is trained, the topic distribution of the APIs are inferred. In the testing phase, the trained model is used to predict the topic distribution of the mashups. For the mashups also its name and description has been considered.

**3.2 Phase 2: Collaborative Filtering: Exploiting API’s usage history**

Using only API names and description in topic modelling do not tend to provide a good result. As the description is not rich enough always. For the same, there is a need to exploit the use of usage history of APIs in existing mashups. The same is achieved by collaborative filtering.

**Collaborative Filtering:** Collaborative filtering is a widely used recommendation algorithm. It aims to predict and provide recommendation for an item or product based on the existing ratings provided by the user to the similar products. Further based on the profile of the current users in the system, an active user who share the similar profile like others would be recommended a product of choice similar to both current and active users. There are many variations of collaborative filtering available, majority of which first performs prediction of the user’s preference. With the same it produces their recommendations by ranking items by predicted preferences. The baseline prediction mechanism in use is the aggregation of the existing user’s ratings on product and computing the mean of the same. Collaborative filtering captures the essence of the user and product’s rating in the form of a matrix called Ratings matrix. Out of the various collaborative filtering techniques available, few are:
(i) User-user collaborative filtering: based on the profile and the past likings of the other users that is similar active user, the ratings on an unrated product is predicted for the current user. But this method suffers from scalability problem with the growth in the user space. (ii) Item-item collaborative filtering: this was to overcome the scalability problem from the previous method. Instead of using the user similarities for predicting the ratings of a product, the similarities between the products were considered. If a particular user has a liking towards multiple products of the similar type, then it is likely, that the same user would like a new product of the same type. For all these basic collaborative filtering methods, probabilistic models have been formulated. This is to compute the probability of the user’s fondness towards purchasing a new item or the probability distribution over the rating of user on an item. Collaborative filtering uses two methods- neighborhood or memory based and model based. In the memory based method, the entire user item matrix is consumed in order to predict. This model uses the statistical methods to find a nearest set of neighbors, whose profiles are used to predict the recommendation for the active or target user. But this technique is ineffective with the growth of the dataset and becomes sparse. In the model based approach, first a model based on users’ ratings is generated using machine learning mechanisms like Bayesian network or clustering. It considers the concept of matrix factorization where they consider the user item rating in a matrix. With this it aims to discover the latent features that could be hidden in the other users’ ratings and aid in making predictions. Many probabilistic models have been generated that contains the direct connection of hidden variables with the user ratings. It also relies on Singular Value Decomposition (SVD). It finds the matrix $R = U^TV$ such that the sum-squared distance to the target matrix $R$ get minimized. In the real world data we can have data sparsity and in many cases the entries in $R$ would be missing, the sum-squared distance is considered only for the existing entries of the target matrix $R$. For the formulation of this system, Probabilistic Matrix Factorization is used. It works on the Bayesian perspective and uses Gaussian distribution. It overcomes the drawbacks of the existing matrix factorization like robustness and sensitivity to noise or missing entries. Algorithm [1]

1. Initialize User and Item result vector to random values
2. Iterate over $n$ times to get the better average results
3. Divide testing set over multiple batches
4. Compute the prediction as weighted average with variable factor lambda using Bayesian distribution.
5. Compute Gradients.
6. Update the training result vector based on the prediction and gradient.
7. Compute the prediction after result update and calculate the RMSE (root mean square error).
8. As an output the item and user feature matrix is generated.

**Incorporating in the system:** For incorporating this method in this system, mashups are treated as users and APIs are treated as the items. Further collaborative filtering has the cold start problem, i.e, the system aims to find the list of relevant APIs based on the user input of the mashup. Again this mashup would be non-existing in the system, making it difficult for the formation of the user item matrix. But this system overcomes the cold start problem by using topic modelling, where based on the user input of mashup description, the similar mashups from the system is obtained. This is achieved by getting the topic distribution of the user input based on the trained model of HDP and computing the similarity with the topic distribution of the existing mashup.

3.3 **Phase 3: Popularity Based API ranking**

As per the proposed system, we have obtained by relevant APIs from Hierarchical Dirichlet Process using the cosine similarity with the mashup. This helps to leverage the functionality of the APIs in providing the recommendations. The output of this model are APIs with probabilistic scores. The greater the score, more relevant they are. Similarly, to leverage the historical usage of APIs in mashup, we are using collaborative filtering with probabilistic matrix factorization. As an output of this collaborative filtering, we can obtain the relevant APIs in a mashup based on its probabilistic score. The outputs obtained from these components of the system can be treated as some score in the range of $[0, 1]$. To generate the final relevant list of APIs from these 2 systems, we can use apply the Bayes theorem.

The probability of an API getting selected from HDP, given a mashup can be treated as $p(a|m)$. Similarly the one from collaborative filtering can be treated as $p(a|m)$. Assuming the conditional dependency, we can say:

$$p(a|m) = p(a|m) \times p(a|m)$$

But we are looking forward to compute given an API, how likely it would be used in a given mashup. This can be derived using the posterior probability: $p(m|a)$. With the application of Bayes theorem, we can compute the same as:

$$p(a|m) = p(a|m) \times p(m)$$

Where $p(m)$ is the prior probability of observing the mashup [10]. This value can be assumed to be as $1/M$, $M$ being total number of mashups used in the experiment.

As an output from Bayes theorem, we may obtain the list of relevant APIs, whose score are identical and they do offer identical functionality as well. To handle such scenario, we must the rank the APIs based on its popularity as well. This non-functional measure of popularity cab be termed as QoS (Quality of Service). Popularity can be defined as the frequency of the API that has been used in any of the mashups present in the database. The more the number of times the API has been used in an existing mashup, the better the rank it has got. To conclude, following all these steps, the proposed system would be able to recommend top-k relevant APIs for a mashup creation based on both functional and non-functional requirement.

4. **IMPLEMENTATION**

4.1 **Data Preprocessing**

The data used in this project was taken from ProgrammableWeb. The data is base on collected from summer 2014 using ProgrammableWeb APIs. The dataset consists of 11199 APIs and 7391 mashups. The data available is the form of
4.2 Building HDP Model

The algorithm requires input data in a particular format to train the model. The format of the input data is: [No of unique terms in an API] [Index of first word]:[frequency of the first word] [Index of second word]:[frequency of the second word] ... [Index of n-th word]:[frequency of the n-th word]. Thus the number of lines in the input file is equal to the number of documents or API in our dataset. The index of the word was obtained from the vocabulary generated from the descriptions of all the APIs. Java code was written to generate the vocabulary from the APIs's description and also to build up the input file data.

1. Environment setup: HDP algorithm package is built on c++. And it is dependent on the GSL (GNU Scientific library) package. For making the algorithm work and configure, a virtual machine was built to run on Ubuntu 14.0. Following the GSL package was installed. Necessary changes were also made in the library paths and environment variables.

2. Training the model: The vocabulary file and input raw text file. Each line in the text file corresponds to each of the API and Mashup.

Preparing HDP input: Only the data corresponding to the attribute ‘description’ and ‘name’ were required. Those for the APIs were needed for building the trained model and the same corresponding to the mashups were needed to infer from the trained model. As a part of data pre-processing, only the description value was retrieved from the raw text file. Further the unwanted characters were erased from the description. Unwanted characters included: (:;N|&%$* ]([)] [0-9]-. Following, the stop-words were removed from the description. Stop words are the words that occur very frequently in a language and requires to be filtered out before the processing of natural language (eg: as, and, the, was, are, etc). Python pre难点 module was used to validate the words against English dictionary. Finally the tool POSTagger was used. It is a natural language processing tool. This tool helps in tagging individual word under noun, adverb, adjective, verb, preposition, and more. For our purpose of building the vocabulary, only the nouns were considered. This was to develop a meaningful vocabulary.

3. Output obtained: Once the model has been trained, it generates several files in the result directory. Following 2 files were studies to get an insight on the result and the performance of the HDP algorithm. Mode-topics file contains the number of topics generated and the frequency of each words from vocabulary. Each line in this file corresponds to the single topic. The format of the mode-topics file: [Frequency of the 1st word from the vocabulary in topic i] [Frequency of the 2nd word from the vocabulary in topic i] [Frequency of the 3rd word from the vocabulary in topic i] ... [Frequency of the nth word from the vocabulary in topic i], where (i ranges from 1 to the number of topics) (n corresponds to the 1 to the number of words present in the vocabulary) Eg: Considering 2 topics generated from the vocabulary of 4 words: 0001 0030 0060 0012 0023 0002 0012 0007

This illustrates that from the entire corpus of documents (APIs), 2 topics got generated. 0 topic contains only 1 occurrence of 1st word from the vocabulary, 30 occurrence of the 2nd word from the vocabulary, 60 occurrences of 3rd word from the vocabulary and 4th word occurs only 12 times. Similarly the distribution of the words in topic1. Mode-word-assignment file contains the distribution of every word from every document to the topics generated. The file stores a matrix in the form of [d w t] where d is the document-id (ranging from 0 to the 1 less than the number of APIs), w is the index of the word from the vocabulary as present in the corresponding document and t is the topic id (ranging from 0 to 1 less than the number of topics formed while training the model).

4. Visualization of the output: We can try and visualize the topics generated while building the training model. R script was run to generate the list of top few words which appear most frequently in each of the topics. Command to run the script: ./printTopics.R mode-topics.dat <vocabulary file> <output file>

5. Testing of the trained model: After the training model was built, it was tested with the user input of mashup description and also the existing mashups. This inferred the topic distribution of the mashups and user input. To run the trained model to test the user input, command used was: ./hdp —algorithm test —data <user test input-data> —saved-model <binary of the trained model generated> —directory <test results directory>. Similarly output files were generated.

6. Computing the cosine similarity: Once we obtained the word topic distribution of the APIs, the training model was used to infer the topic distribution of the existing mashups. This helps to find the relevant mashups based on the user-input and then look for the relevant APIs. The similarity measure used is cosine similarity.

Figure 4: Data pre-processing steps

Output obtained:

3. Output obtained: Once the model has been trained, it generates several files in the result directory. Following 2 files were studies to get an insight on the result and the performance of the HDP algorithm. Mode-topics file contains the number of topics generated and the frequency of each words from vocabulary. Each line in this file corresponds to the single topic. The format of the mode-topics file: [Frequency of the 1st word from the vocabulary in topic i] [Frequency of the 2nd word from the vocabulary in topic i] [Frequency of the 3rd word from the vocabulary in topic i] ... [Frequency of the nth word from the vocabulary in topic i], where (i ranges from 1 to the number of topics) (n corresponds to the 1 to the number of words present in the vocabulary) Eg: Considering 2 topics generated from the vocabulary of 4 words: 0001 0030 0060 0012 0023 0002 0012 0007

This illustrates that from the entire corpus of documents (APIs), 2 topics got generated. 0 topic contains only 1 occurrence of 1st word from the vocabulary, 30 occurrence of the 2nd word from the vocabulary, 60 occurrences of 3rd word from the vocabulary and 4th word occurs only 12 times. Similarly the distribution of the words in topic1. Mode-word-assignment file contains the distribution of every word from every document to the topics generated. The file stores a matrix in the form of [d w t] where d is the document-id (ranging from 0 to the 1 less than the number of APIs), w is the index of the word from the vocabulary as present in the corresponding document and t is the topic id (ranging from 0 to 1 less than the number of topics formed while training the model).

4. Visualization of the output: We can try and visualize the topics generated while building the training model. R script was run to generate the list of top few words which appear most frequently in each of the topics. Command to run the script: ./printTopics.R mode-topics.dat <vocabulary file> <output file>

5. Testing of the trained model: After the training model was built, it was tested with the user input of mashup description and also the existing mashups. This inferred the topic distribution of the mashups and user input. To run the trained model to test the user input, command used was: ./hdp —algorithm test —data <user test input-data> —saved-model <binary of the trained model generated> —directory <test results directory>. Similarly output files were generated.

6. Computing the cosine similarity: Once we obtained the word topic distribution of the APIs, the training model was used to infer the topic distribution of the existing mashups. This helps to find the relevant mashups based on the user-input and then look for the relevant APIs. The similarity measure used is cosine similarity.

Figure 4: Data pre-processing steps

Output obtained:

3. Output obtained: Once the model has been trained, it generates several files in the result directory. Following 2 files were studies to get an insight on the result and the performance of the HDP algorithm. Mode-topics file contains the number of topics generated and the frequency of each words from vocabulary. Each line in this file corresponds to the single topic. The format of the mode-topics file: [Frequency of the 1st word from the vocabulary in topic i] [Frequency of the 2nd word from the vocabulary in topic i] [Frequency of the 3rd word from the vocabulary in topic i] ... [Frequency of the nth word from the vocabulary in topic i], where (i ranges from 1 to the number of topics) (n corresponds to the 1 to the number of words present in the vocabulary) Eg: Considering 2 topics generated from the vocabulary of 4 words: 0001 0030 0060 0012 0023 0002 0012 0007

This illustrates that from the entire corpus of documents (APIs), 2 topics got generated. 0 topic contains only 1 occurrence of 1st word from the vocabulary, 30 occurrence of the 2nd word from the vocabulary, 60 occurrences of 3rd word from the vocabulary and 4th word occurs only 12 times. Similarly the distribution of the words in topic1. Mode-word-assignment file contains the distribution of every word from every document to the topics generated. The file stores a matrix in the form of [d w t] where d is the document-id (ranging from 0 to the 1 less than the number of APIs), w is the index of the word from the vocabulary as present in the corresponding document and t is the topic id (ranging from 0 to 1 less than the number of topics formed while training the model).

4. Visualization of the output: We can try and visualize the topics generated while building the training model. R script was run to generate the list of top few words which appear most frequently in each of the topics. Command to run the script: ./printTopics.R mode-topics.dat <vocabulary file> <output file>

5. Testing of the trained model: After the training model was built, it was tested with the user input of mashup description and also the existing mashups. This inferred the topic distribution of the mashups and user input. To run the trained model to test the user input, command used was: ./hdp —algorithm test —data <user test input-data> —saved-model <binary of the trained model generated> —directory <test results directory>. Similarly output files were generated.

6. Computing the cosine similarity: Once we obtained the word topic distribution of the APIs, the training model was used to infer the topic distribution of the existing mashups. This helps to find the relevant mashups based on the user-input and then look for the relevant APIs. The similarity measure used is cosine similarity.
similarity. It is first computed between the user-input topic distribution and the mashup distribution. Once obtained, the mashups showing similarity more than 0.7 were considered. With this candidate mashups, the similarity was computed with every existing APIs. And the APIs that showed similarity more than 0.7 were considered to be the candidate list from HDP phase. Cosine similarity is computed between 2 documents using a Java code that takes input as vectors and computes the similarity. The value of similarity ranges from 0 to 1, 1 being completely similar.

4.3 Performing Collaborative Filtering

Matlab package [1] for probabilistic matrix factorization was used. For this experiment, the list of candidate mashups considered were the ones that uses at-least 4 APIs. This reduced our dataset for collaborative filtering to 909 mashups with 852 APIs. This takes the input in the form a matrix which has the combination of Mashup id and API id with a static value in every row. Java code was written to formulate this input matrix for the algorithm. The code was run for various epochs and RMSE (Root Mean Square Error) was calculated. The model giving the minimum RMSE was made to generate the API feature matrix and Mashup Feature Matrix. Followed by this java code was written to process these 2 matrices and generate the actual ratings matrix that helped in finding the mostly used APIs in the mashup creation. The value in the matrix is probabilistic value ranging from [0, 1]. To look for the relevant APIs for a particular mashup, the ones with the values more than 0.7 were considered.

4.4 Bayes Theorem and Popularity Based API Ranking

Based on the candidate list of APIs obtained from HDP and Collaborative Filtering, Bayes Theorem was applied to get the combined list. The candidate list obtained represent each API with a score: the probabilistic value between [0, 1]. Using the same in the below formula the score for all the APIs were obtained. Finally they were ranked based on the popularity. The popularity of an API was measured based on the frequency of it has been used in the existing mashups.

\[ p(a_t, a_u|m) = p(a_t|m) \times p(a_u|m) \times p(m) \]

5. RESULT

HDP modelled applied on the names and description of the APIs, yielded 73 topics. Below is the visualization of the topic probability distribution across few documents. Below are few of the topics with the words appearing the most frequently in the topics. Different color shows the different frequency of the words in the topic. The frequency of the words increases from right to left.

As a result of collaborative Filtering it was found that Google Maps, YouTube, Facebook were among the most used APIs by the majority of the Mashup.

6. EVALUATION

To evaluate the system, we divided the current mashups present in the system in the training set and testing set. The ration of division was 8:2. For the evaluation metrics, we used recall, precision and F-score.
Recall can be defined as the ratio of the number of correctly recommended APIs by the system to that of total number relevant APIs.

$$Recall = \frac{|Relevant\ APIs| \cap |Recommended\ APIs|}{Relevant\ APIs}$$

Precision can be defined as the ratio of the number of correctly recommended APIs by the system to that of total number recommended APIs.

$$Precision = \frac{|Relevant\ APIs| \cap |Recommended\ APIs|}{Recommended\ APIs}$$

F-score considers both recall and precision.

$$F-Score = \frac{Recall \times Precision}{Recall + Precision}$$

For the computation of these evaluation metrics, we used total 300 mashups and computed the average the value. The computation was carried across different values of k ranging from 10-100. Having an interval of 10. This helped in getting the recall and precision when only k-top relevant APIs are needed. Followed by this system has also been compared with the recommendation system based on only HDP and collaborative filtering using probabilistic Matrix Factorization. We also compared this work done with the previous work published along with the system that used LDA and traditional collaborative filtering instead. Below figures shows the graphs for the same.

From the graphs we see that HDP does not perform quite well, because it only considers the functional requirement without taking into the count the usage history of the mashups. On the other hand we see little improvement in collaborative filtering with matrix factorization as it considers the usage history of the mashup. But it suffers from the cold start problem. But with the hybrid system, it helps to provide a better recommendation. We also add to this the popularity based QoS factor that refines the recommendation. This work has been compared with the other 2 works done in [6] and [8]. Similar metrics have been used in the evaluation in this system as mention in [8]. This is to ensure there is a
fair comparison. Below table shows the comparison of the recall and average precision across all the 3 systems.

7. CONCLUSION

It can be concluded that proposed approach yields better result over traditional methods of topic modelling and collaborative filtering. In this paper, new advanced algorithms for topic modelling and collaborative filtering were tried. It tried to overcome many problem of the traditional process and yielded a better result in providing recommendation of APIs for mashups.

8. FUTURE WORK

Future work involves improving the performance of the system by use of other advanced algorithms. This can also be done by careful study of the data attributes. As only ‘name’ and ‘description’ attributes has been considered in the current approach, this can be extended to use of category, tags, ratings and other attributes present in the dataset. Using these data attributes may show some interesting relationship. Further the concept of web composibility may be leveraged in increasing the effectiveness of the system. This can be explained with a simple example: Google maps and yahoo maps almost cater the same functionality but they are different in their own way. When building a mashup that requires the use of map API, Google API is more likely to be recommended as its usage history in the existing mashups is more. But sometimes may be the Yahoo map API may best suit the requirement of the user compared to the Google map API. This issue can be resolved using the idea of web composibility.

9. ACKNOWLEDGEMENT

Thank you to Dr. Xumin Liu, because of whom, this project exists today and who has helped me in enriching my knowledge in this domain. Thank you to my friends and family, because of whom, I work harder every day and for their continuous support in hard times. Thank you to the RIT CS Department, because of which, I have my Master’s degree.

10. REFERENCES