PyReco - Offline Recommendation System for Python Programmers

Ashutosh Chaturvedi
Department of Computer Science
North Carolina State University
Raleigh, North Carolina, USA
achatur@ncsu.edu

Ayush Gupta
Department of Computer Science
North Carolina State University
Raleigh, North Carolina, USA
agupta25@ncsu.edu

Ankit Kumar
Department of Computer Science
North Carolina State University
Raleigh, North Carolina, USA
akumar18@ncsu.edu

Harshdeep Kaur
Department of Computer Science
North Carolina State University
Raleigh, North Carolina, USA
hkaur4@ncsu.edu

ABSTRACT

Often programmers are held up with doubts or coding issues due to not having access to Internet resources or any proper guidance. We had proposed to solve this issue by providing an offline solution to the user. PyReco solution is conceived for python programmers who work in environment that do not have access to Stack Overflow website. The three solutions proposed allow user to host data in their local system and access it as a plugin in Sublime text editor. In this paper we present problem review, proposed solutions, evaluations of these solution and future enhancements with conclusions.

GENERAL TERMS

Data Analysis, Research, Development, Study, Evaluation, Analysis, Business Intelligence, Recommendation

KEYWORDS

Recommendation system; Python; StackOverflow.com; programmers; syntax; IDE; Sublime Text

1. INTRODUCTION

A common problem that programmers have been facing since the advent of programming languages and face even today, is finding a solution to the syntactical errors while programming in different languages. Users who are new to a programming language, or migrating to different programming languages regularly face problem with correct syntax to be used while coding any feature. While many languages like Java, C++, C#, etc., have advanced IDEs like eclipse and Visual Studio which provide a solution to this problem to some extent, many other like python do not have such advanced IDEs and the programmers who program in these languages have to constantly refer to resources like tutorial documents or internet for solutions. Also, because python has an interpreter instead of a compiler, the task of finding errors at compile time becomes even more difficult and leaves no option to the user but to refer the stated resources. Documentation and Tutorial provide with information about the features of the language and how to use different features but they are incompetent when it comes to providing solution to general issues faced by the users. Generally, the sought solution is based on other user's experience with the similar issue and same is posted on different forums across internet. Most of the time people get solution for their problems by referring to these forums. Searching for the optimal solution across these platforms is pretty cumbersome and time consuming at times as the user might have to go through a number of solutions for the same problem. Also, in the absence of internet, there is pretty much no option left because of so much dependency on the internet these days. There is no dedicated offline tool to help in this cause. When a programmer move to a new language other than his primary programming language, or a new user is learning a new language, most of the time the issue faced by them is how to express their login in proper syntax as per the rules of the programming language at hand.

This paper presents three solutions to encounter the python syntax problems faced by programmers, through an offline recommendation system which takes input from a sublime text plugin and displays results according to that input. All the searches are based on data collected from stackoverflow.com as it is the most widely used platform for searching answers to programming problems as found out by our previous user studies.

The reason behind developing a sublime text plug-in for this is the fact that sublime is a widely used text editor with cross platform support. Also, developing a sublime plugin is easier and python based which is the primary language we have used for this whole project. The database we have used is MongoDB. The reason behind this is the text indexing support provided by MongoDB that enables us to search for words inside textual strings. The Text Search uses stemming techniques to look for specific words in the string fields by dropping stemming stop words like a, an, the, etc. At present, MongoDB supports around 15 languages.

The three solutions we have provided are basically three approaches or algorithms for searching the python keywords inside the stack overflow data to display relevant questions and their solutions. The first solution uses token analyzer in which questions are divided into tokens and unwanted words are removed using stopword removal method. These tokens are then stored in the MongoDB with questions and their respective IDs. In the second solution we are using the K-Means clustering algorithm which is discussed later in this paper. In our third solution we tried to improve the quality of search by supplying background data lying around the keyword the user is searching for. This helps us analyze and find posts which are more relevant to the search, thanks to the background data provided, to provides better search results to the user.
These three solutions are then analyzed through user experiments and surveys to determine which solution provides the most favorable results.

2. RELATED WORK

There are various applications being built that use the Stackoverflow.com data for different purposes. In specific, there is an application present to provide an Offline version of Stackoverflow.com called StackDump. This application requires the user to download the huge dump of Database from StackExchange.com and then use the application for offline search. But this application does not provide an efficient alternative for browsing through the various articles it provides in the search, i.e. it provides only basic search results as Stackoverflow.com provides. Also, it asks the user that they have a lot of memory free for the data dump but it would include a lot of overhead for programmers who just need syntactical help and hence articles related to that. Our aim is to provide solutions to these unanswered problems.

3. EVOLUTION OF PYRECO

3.1 Base Model:

For our solutions we used data from stack overflow [1] . Since our target programmers were only python programmers, we filtered the data for python and built our solutions on that data. The first solution proposed is by using Token analyzer. Each stack overflow post has parts containing a question, answer selected, tags etc... We divided the questions into tokens and removed unwanted words from it using stop word removal method. The stopwords bag is inbuilt stopword list in NLTK made up of 2,400 stopwords for 11 languages [2]. We also created our own bag of common words and added it to the stopwords list. Our own stopwords list contained about 950 words. After post processing of the tokens, we represent each of the questions with an id and their respective set of tokens. We store the resultant data in our Mongo DB database. We take advantage of MongoDB's inbuilt capability of Full-Text search [3] using text indexes for finding the related answers from the database for a given question. Creating text indexes on the MongoDB, allows MongoDB to tokenize and stem the indexed field's text content and sets up the indexes accordingly. Though this feature of MongoDB helps us to find results very fast as compared to any relational database, but the linear search methodology is slower and inefficient for all the queries.

3.2 Solution 2:

As we are dealing with big data we implemented our next solution by making use of unsupervised learning of data. K-means [4] is one of the simplest unsupervised learning algorithms that solve the well-known clustering problem. The procedure follows a simple and easy way to classify a given data set through a certain number of clusters (assume k clusters). The main idea is to define k centers, one for each cluster. These centers are placed as much as possible far away from each other because different location causes different result and placing them far is a better choice. The next step is to take each point belonging to a given data set and associate it to the nearest center by using some similarity metric. After assigning every data point to a cluster, k new centroids are recalculated from the clusters and step 1 and step 2 are repeated. As a result of this loop k centers change their location step by step until no more changes are done or in other words centers do not move anymore. Finally, this algorithm aims at minimizing an objective function known as squared error function given by:

$$SSE = \sum_{i=1}^{K} \sum_{x \in C_i} dist^2(m_i, x)$$

Data preprocessing is an important step in data mining process. Analyzing data that has not been carefully screened can produce misleading results. If there is much irrelevant and redundant information present then the knowledge discovery during the training phase is more difficult. Hence for k- means we used data preprocessed from our base

solution. And finally before being able to run k-means on a set of questions, the questions need to be represented as mutually comparable vectors. To achieve this task, the questions are represented using the tf-idf [5] score. The tf-idf, or term frequency-inverse document frequency, is a weight that ranks the importance of a term in its contextual document corpus [6] . Term frequency is calculated as normalized frequency, a ratio of the number of occurrences of a word in its document to the total number of words in its document. Usually, in k-means Euclidean distance is used for similarity but in our solution we used cosine similarity between vectors to find out the similarity. Cosine similarity is calculated as

$$\cos(\mathbf{a}, \mathbf{b}) = \frac{\mathbf{a}^T \mathbf{b}}{|\mathbf{a}|\,|\mathbf{b}|}.$$

Our k means model resulted into clusters of similar question types, which we used to search most relevant cluster based on the user query. Based on the user input query our solution finds the most relevant cluster from our clustering model. Since each cluster has large number of questions, we filtered out the most relevant question-answers based on various metrics that we obtained from stack overflow. We selected top 10 answers based on view count, vote count and rating for each answer.

We choose k-means algorithm for clustering because the algorithm is fast, robust and easier to understand. We evaluated the clustering performance and found areas where we have scope of improving. We found few overlapping clusters because of data with high dimensions and low separation.
Hence we implemented our next solution by improving the algorithm for overall better results.

3.3 Solution 3:

Our final solution has been developed as an improvement from solution 2. In this solution, when searching for a query, along with the highlighted text, we also send background data (i.e. contextual keywords from the user code) for a better search. This helps us to understand the context of the question words/ queries and provide better related answers.
Computation of similarity through word comparison is a really tedious task and highly inefficient. Hence, to implement this feature we added the SimHash algorithm to estimate the similarity between the questions in our DB and the background data supplied [7]. SimHash is an effective solution to compare two datasets as we can pre-compute the hash for the questions and store them as another collection in our DB and thus at runtime we only need to compute the hash for the background data supplied and then compute the distance between our hash and hashes corresponding to the various questions, which the clustering algorithm gave us. This distance can be computed really efficient by taking an XOR of the two hash values and then the minimum of these distances would provide us with questions having a better match with the background data. As hashing is a really efficient algorithm and takes a lot less time than comparing for common words between two large collection of words, SimHash enabled us to efficiently implement the desired feature of background data matching, to improve the search. The resultant posts are again filtered based on factors like Vote Count, Number of views, Rating to find the closest and the most appropriate as well popular answer for the search query.

4. EVALUATION

4.1 Design of Tests:

Our project aimed at developing an offline recommendation system that could provide users with better search results than a simple search on stackoverflow.com would and do it offline. We claimed that this would help them work offline and fix errors in their code easily and efficiently. Hence, to test the effectiveness of our solutions, we created a buggy file in python which contained 5 lines of code. Each line of code had a syntactical bug in it and the user's task was to fix that bug. A total of 20 users participated in our testing and we divided them into groups of 5, where each user in every group was asked to fix the same buggy file, but using different solutions. So Groups 1, 2 and 3 were required to use our different solutions, i.e. Solution 1, Solution 2, and Solution 3 respectively, and Group 4 was asked to use stackoverflow.com to fix the bugs. As all of our solutions provided only 10 results for each search, hence the users in Group 4 were asked to use any filter they want but look at only the top 10 results in their search to fix the code they were given.

4.2 Data Collection:

Data was collected by logging user activity on our application, through user surveys, and by evaluating the code fixes the users made to the buggy code we provided.
To log user activity, we designed our interface in a such a way that the user could only view the questions directly, and then needed to click on a tag called "Answer" to expand that tag and show the answer for that particular question. We further attached a button called "Relevant" with each question, and asked the user to click on it, only if they find the answer relevant to the bug they are trying to fix. This way we logged which answers the user expanded, and which answers they found relevant. This method could not be used

Sol/Metric	H	S	SH	R	RC	RA	RCT	RAT	BF
Sol 1	56	56	75	80	30	34	100	93	60
Sol 2	24	24	25	40	0	5	NONE	100	0
Sol 3	84	84	100	93	70	61	76	81	80

Table (1) : Parameters with Data Collected for Evaluation of Solutions

to log data for the users using a simple search on stackoverflow.com.

In the user surveys, we presented 4 questions to the users and asked them to answer each question based on their overall experience of using our tool to fix the bugs. All these different questions helped us analyze different aspects of our solutions from the user's point of view. 2 of these questions included rating some parameter on a scale of 1 to 5. These parameters were helpfulness (how helpful the search was to fix the bugs), and satisfaction (how impressive the solution was to the user). Two other questions were more qualitative and involved questions on which search method the user tested (Any of the 3 Solutions or a Simple Search on StackOverflow.com) and would the user recommend this solution (if Any of the 3 Solutions we developed was tested) to others or not (Options being Yes, No, and Maybe). The last questions required the user to report how many of the bugs were these searches able to fix using one of the 4 provided options (< 25%, 25%-50%, 50%-75%, >75%).

The third and the last method of data collection involved was comparing the files the users fixed, with the solution file we created, containing the correct solution for each bug we provided the user with. Data from this part of testing helped us evaluate the effectiveness of every solution in terms of fixing errors in code, and thus gave us an insight into the usefulness of our project.

4.3. Metrics of Evaluation:

We decided upon 9 metrics of evaluation for our project. These included 4 from user surveys and the remaining 5 from user activity logging and fixing of bugs by the user.

User surveys generated results (averaged across the feedback from the 5 users for each solution) on the basis of:

- H: Helpfulness
- S: Overall Satisfaction
- SH: Search hits (Each option from 1 to 4 was assigned a corresponding weight from 1 to 4)
- R: If the user would recommend the solution to others (Each option from 1 to 3 was assigned a corresponding weight from 1 to 3)

User activity logging generated results for 4 parameters:

- RC: The ratio of "Relevant" button clicks each solution received to the total number of "Relevant" button clicks (for all solutions)
- RA: The ratio of answers expanded using each solution to the total number of answers expanded (for all solutions)
- RCT: The ratio of "Relevant" button clicks each solution received for the top 5 search results to the total number of "Relevant" button clicks that solution received; and
- RAT: the ratio of answers expanded in the top 5 search results to the total number of answers expanded for that solution.

The bug fixing test generated result for 1 parameter:

- BF: Ratio of bugs fixed to the total number of bugs present for each solution (Average of results from 5 different users)

The output obtained for each of the metrics defined was recorded for each solution and then the solutions were compared based on those metrics. The values obtained (in percentage) for these metrics are reported in Table (1).

4.4. Results:

From the metrics computed, we were able to figure out that users preferred solution 3, and gave a higher rating on each of the metrics to this solution. Solution 1 was the second favorite and solution 2 was the worst of all.

Further based on the metrics computed from data logging, solution 3 provided more answers that the users found useful and hence got more answers expanded and consequently more "Relevant" button clicks. However, of all the useful answers generated by each solution, solution 1 was able to generate the needed solution, in the top 5 answers it provides, more number of times than solution 3. Solution 2 performed really bad in generating good answers and hence was the worst possible solution out of the 3.

Also, using solution 1 the users were able to fix 60% of the bugs (3/5), using solution 2 they were not able to fix any bug (0/5) and using solution 3 they were able to fix 80% of the bugs (4/5).

Solution 2 only got a higher RAT, which means that out of all the answers that the user expanded, all of them were in the top 5. However, this is not a correct measure of its performance, as the users only expanded two answers (both

being in the top 5) which makes the number of answers expanded very small and irrelevant in the context of comparison. Further, as the user was not able to fix any bug using this solution, we can safely assume that the user thought a particular answer was perhaps related and it did not turn out to be so.

4.5 Recommended Solution:

Solution 3 had higher RA and RC, got the best feedback from the users and they were able to fix most number of bugs (4/5 = 80%) using it. In comparison to this, Solution 1 had higher RCT and RAT, got moderate feedback from the users and the users were able to fix quite a few bugs (3/5 = 60%) using it. Further, Solution 2 was the worst of all which generated low values for all the parameters and the users were not able to fix any bugs using it. Using this analysis as our basis, we strongly recommend Solution 3 because of its good performance.

4.6 Comparison with Simple Search:

All the users who used stackoverflow.com were able to fix all the 5 bugs and gave a really high rating to stackoverflow.com. This shows that our best solution (Solution 3) falls short of recommending better results than stackoverflow.com does, but provides results close to what stackoverflow.com provides. Further, as another component of our project was an offline system, and as the user survey suggests that the users liked our approach very much, we think that our solution is useful but has its own limitations and needs some improvements.

5. FEEDBACK & FUTURE ENHANCEMENTS:

The solutions proposed use data analysis tools to predict the closest possible answers to the user queries. Clustering of data. even though comprehensive, does not always provide the most desired results. Also, the technical keywords frequency is low in the question text as compared to general (not helpful) words. The clustering techniques, which uses word frequency as the basis of creating clusters and do not have context knowledge, are unable to create clusters that are exclusive in the sense of sharing minimum keywords. It's noteworthy that the local optima to which our solution model converges, vary vastly depending on the random initialization. The model's parametric values depend on comprehensive analysis of the resultant clusters and their closeness. The number of iterations is an important factor here as they limit the runtime of the clustering algorithm, but in some cases the quality of the clustering will be poor because of an insufficient number of iterations. As the assignment of a document to a cluster does not change between iterations unless the local minimum calculated is poorly, multiple iterations may improve clustering but increase the run time thereby increasing the need for resources.

In K-Means algorithm there is unfortunately no guarantee that a global minimum in the objective function will be reached, this is a particular problem if a document set contains many outliers.

Through the feedbacks from our users and our own research we found out that even our model lacks complete optimum solution which is possible due to the outliers in our data or our chosen algorithm.

Therefore our future work lies on exploring other document clustering algorithms for finding better resultant clusters and data analysis algorithms for optimal matches from these clusters. We can also explore the possibility of compressing the dataset to make it easier to transfer and deploy in other systems.

6. CONCLUSIONS:

Through this project, we proposed a solution for Python programmers by providing them with offline solutions from Stack Overflow. Initially, we had many options at hand for data mining and data analysis. Given the huge size of the offline Stack Overflow data, it was clear that the pre-processing will be a tedious task and our model should be apt enough to handle it. Topic modelling was our naive approach but when combined with text indexing feature of MongoDB, it gave us fairly optimal solutions. Linear search nature of this approach made us think about our clustering solutions which uses K-Means. Solution 2 was quick and gave us local optimal solutions. We got the advantage of applying our own filtering algorithm on the result set. The pre-processing took a longer time but it gave us the advantage of quick search results at actual run time.

Evaluation of solution 1 and solution 2 showed a gap where we were missing the background context of the query while searching for the answer. The contextual data helps to find the closest answers to a question as we are only using few words to find the related questions. We implemented SimHash algorithm to further enhance our answer filtering and rating algorithm. The evaluation of this solution was extremely satisfactory and made it our choice of solution. Further enhancements as discussed can polish this solution even more and make it more optimal.

Sublime is a widely used cross platform text editor for programming, and combined with our PyReco Recommendation System, it can enable a programmer to code and refer to queries in any environment, even while they are offline.

7. ACKNOWLEDGEMENTS:

We express our sincere gratitude to Professor Dr. Tim Menzies for the continuous support of our study and related research. We would also like to thank our fellow colleagues, friends and professionals whom we interviewed initially and who took part in our surveys to provide us initial data to understand the problem well. And finally tested our solutions to provide us proper feedbacks which helped us to finally conclude with the near optimum solution.

8. REFERENCES:

[1] Tim Menzies: Improving IV & V Techniques through the Analysis of Project Anomalies: Text Mining PITS issue reports preliminary report.

[2]http://jonathanzong.com/blog/2013/02/02/k-means-clustering-with-tfidf-weights

[3] Zhang, GAO, Zhou and Wang: *Improving the Effectiveness of Information Retrieval with Clustering and Fusion.*

[4] Anton Leuski: *Evaluating Document Clustering for Interactive Information Retrieval*

[5]http://scikitlearn.org/stable/auto_examples/text/document_clustering.html

[6]http://programminghistorian.org/lessons/topic-modeling-and-mallet

[7]http://www.codeproject.com/Articles/439890/Text-Documents-Clustering-using-K-Means-Algorithm

[8]http://research.microsoft.com/en-us/um/people/jfgao/paper/clclp01-2.pdf

[9]http://people.ict.usc.edu/~leuski/publications/papers/ir-235.pdf

[10] Larsen and Aone: *Fast and effective text mining using Linear - time document clustering*

[11]https://www.kaggle.com/c/word2vec-nlp-tutorial/details/part-2-word-vectors

[12]http://python-notes.curiousefficiency.org/en/latest/python3/questions_and_answers.html

[13] http://stackapps.com/

[14]http://link.springer.com/article/10.1007/s10664-012-9231-y

[15] https://archive.org/details/stackexchange

[16]Wei Song and Soon Cheol Park: *A Novel Document Clustering Model Based on Latent Semantic Analysis*

[17] http://stats.stackexchange.com/questions/143547/how-are-the-clustering-algorithms-using-the-concept-of-latent-semantic-analysis

[18] Clayton Stanley and Byrne: *Predicting Tags for StackOverflow Posts*