Commit Analysis

Sourish Roy
Dept. of Computer Science
University of Calgary
Calgary, Canada
sourish.roy@ucalgary.ca

Suchina Parihar
Dept. of Electrical and Computer Engineering
University of Calgary
Calgary, Canada
suchina.parihar2@ucalgary.ca

Abstract— Researchers have carried out multiple techniques to extract underlying topics that relate to software development artifacts. This paper focuses on the importance of analyzing commit messages using a highly statistical approach. We have used multiple machine learning techniques to perform topic modelling on these messages. Current topic modeling techniques assume manual labelling and do not use automated or domain-specific knowledge to improve, contextualize, or describe results for the developers. We would investigate to see if there is a way to do it less manually. We have also noted that date features, such as the number of days since the last commit, are good predictors of project aliveness. We would also analyze the number of pushed or pulled activities. The former activity is exclusively performed by developers granted write permission on the project, while the latter can come from anybody. This might give us an idea of the number of real developers for an application. Finally, we would like to review the labelled topics and do an analytical study of the software development lifecycle processes followed by several applications.

Keywords—topic modelling; commit; LDA; machine learning

I. INTRODUCTION

The tricky thing about commit messages is that they seem unimportant when we are writing them because we know what changes we just made and why we made them and the code is right there so it is obvious. Those are incorrect assumptions. Commit messages are important because when someone is reviewing a developer's work they set the stage for what the change set contains. Thus these messages can be analyzed and some logical information can be inferred from these. We could also apply topic modelling techniques on these messages. Topic modeling is a machine learning technique which creates multinomial distributions of words extracted from a text corpus. This technique infers the hidden structure of a corpus using posterior inference: the probability of the hidden structure given the data. Topic models are useful in software maintenance because they summarize the key concepts in a corpus, such as source code, commit comments, or mailing-list messages, by identifying statistically co-occurring words.

Among other uses, it can quickly give developers an overview of where significant activity has occurred, and gives managers or maintainers a sense of project history.

Though machine learning techniques can automatically identify clumps of commonly recurring terms, devising an appropriate summary label for each clump/topic is harder.

A given topic extracted from a set of commit logs might consist of the following terms: "host listener change remove add multiply". This topic might reasonably be labelled as "event handling" by a developer who understands the domain well, despite the fact that this label does not appear in the word list itself. Presently the approaches to label topics rely on manual intervention by human experts, and also are limited to project-specific topic labels. In this paper, we would try to analyze the possibility of labelled topic extraction, an approach to topic labelling that creates labels automatically. We would like to see if that approach is feasible and easily computable.

We would also like to look for **project aliveness** based on the commit dates. Aliveness can be judged with individual features to better understand how certain factors correlate or fail to correlate. For example, we can look into the commit messages to check for pull or push requests to monitor the developer activity for each application.

In our approach we use a dataset of around 1400 applicaions with over 5000 commit messages. A large dataset like this gives us an abundane of data and good validation points for the assumptions we make. We also conduct certain analytical studies to visualize our results using various data representation techniques like bar graphs. pie charts, line graphs, histograms, box plots, dendrograms, scatter plots, network graphs and various other machine learning techniques to consolidate and further strengthen the understanding of anlayzing and managing software projects by making them as visual as possible and thus helping readers gain a valuable insight in this domain irrespective of their first hand knowledge in this field.

2. RELATED WORK

Marcus et al. [1] use Latent Semantic Indexing (LSI) to identify commonly occurring concerns for software maintenance. The concerns are given by the user, and LSI is used to retrieve them from a corpus. Topic modelling generates topics that are independent of a user query, and relate only to word frequencies in the corpus. In some cases, topics are named manually: human experts read the highest-frequency members of a topic and assign a label accordingly. As discussed earlier, given the topic "listener change remove add fire", a label would be assigned like event-handling.

The labels are reasonable enough, but still require an expert in

the field to determine them.
Furthermore, these labels are project-specific, because they are generated from the data of that project. A topic analysis tool such as LDA will try to find N independent word distributions within the word distributions of all input messages. Latent Dirichlet Allocation (LDA) is thus a generative probabilistic model for collections of discrete data such as text corpora. LDA is basically a three-level hierarchical Bayesian model, in which each item of a collection is modeled as a finite mixture over an underlying set of topics. Each topic is, in turn, modeled as an infinite mixture over an underlying set of topic probabilities. In the context of text modeling, the topic probabilities provide an explicit representation of a document. Principal component analysis (PCA) latent semantic indexing, and independent component analysis (ICA) are key methods in the statistical engineering toolbox. They have a long history, are used in many different ways, and under different names. They were primarily developed in the engineering community where the notion of a filter is common, and maximum likelihood methods less so. They are usually applied to measurements and real valued data. Text clustering is the key technology for topic detection, and topic detection is essentially similar to the unsupervised clustering. However, general clustering is based on global information, and clustering in the topic detection is based on incremental ways. So we should study topic detection according to clustering algorithm, and it is necessary for clustering algorithm to be in-depth and extensive research. And K-means is a well-known and widely used partitional clustering method. Thus, we develop a topic detection prototype system to study how K in K-means affects topic detection [2]. In this study, we also focus on one aspect of project success by measuring the chances that a project will be alive in the future, will receive updates, new features, and will not be ignored. We would also attempt to do lifecycle modelling of the topics derived but unfortunately there is no literature available in this particular domain.

3. STUDY DESIGN AND EXECUTION

3.1 RESEARCH QUESTIONS

To formally define the directions of this study, we describe our goals in four research questions, shown below:

RQ1: What is the frequency distribution of the number of releases across various apps? What is the frequency distribution of the number of days, weeks or months of app releases on Github?

An application has multiple releases. And there is a certain time span between each release. Again inside each release there are multiple commit messages. Analyzing the span between releases and later on analyzing the commit messages of each release will help us understand release activity of all the apps.

RQ2: How do you perform a cluster analysis and model your topics based on the data provided to you and why is it useful?

Several machine learning techniques have been used to perform topic modelling. Also it underlines the very importance of topic modelling and the drawbacks of unsupervised machine learning.

RQ3: Can we define lifecycle stages of an app based on the topics after the topic modelling process and how can we judge project aliveness?

Would help us get a sense of the Software Development Lifecycle Process followed by the apps. This field of research is relatively new and manual labelling will be followed to generate our results. We look into the commit dates to find the date since last commit to judge how alive a project is.

RQ4: Does popularity correlate with characteristics of a repository like number of commits and the keywords in the commit messages?

Inside the commit messages we can see key words like "pull" and "pushed". These keywords might help us investigate the number of developers involved in the application and if the application is popular among developers or not.

3.2 GENERATING THE DATA

We were provided a data set of approximately 1400 android applications. Each of these applications had a varied number of commit messages a total of approximately 5000. These were entirely raw csv files and in order to make meaning out of it we had to create a text corpus. Our dataset looks like the following.

	Our dataset sample		
	Commit id	***Commit Date***	***Commit message***
	95901b8b60cb29c6c9d3cffdb7150cb65e34389e	2012-03-14 18:09:51-07:00	initial commit
	fbae248ff3262162204cfdba784f242d9e2c68bc	2012-03-15 20:47:21-07:00	clipping bugfix
	f8fce7d26fbdab6d52db4cf6082c4ed9ab424a50	2012-03-15 21:43:31-07:00	condense train display by destination
	d9188b723c557e61f6a4410bef27a5c5eaa85bd5	2012-03-25 21:57:53-07:00	Merge pull request #1 from Miserlou/master Icon v1

Table I

From the corpus we tokenized the documents, removed common words (using a stoplist) as well as words that only appear once in the corpus. Next we assigned a unique integer id to all words.

To convert documents to vectors, we'll use a document representation called **bag-of-words**. In this representation, each document is represented by one vector where each vector element represents a question-answer pair

Then we count the number of occurrences of each distinct word, convert the word to its integer word id and return the result as a sparse vector. Next we transformed documents from one vector representation into another. This process serves two goals. It brings out hidden structure in the corpus, discover relationships between words and use them to describe the documents in a new and more semantic way. It makes the document representation more compact. This both improves efficiency and efficacy (marginal data trends are ignored, noise-reduction). Next we transformed our training corpus via **Latent Semantic Indexing** into a latent 2-D space. Next we did a **Latent Dirichlet Allocation, LDA** is yet another transformation from bag-of-words counts into a topic space of lower dimensionality. LDA is a probabilistic extension of LSA (also called multinomial PCA), so LDA's topics can be interpreted as probability distributions over words. These distributions are, just like with LSA, inferred automatically from a training corpus. Documents are in turn interpreted as a (soft) mixture of these topics. This is the enitre process we followed to get our modelled topics from the commit messages. The commit dates would be used later to check the aliveness of a given software application. Keywords in the commit messages themselves were used to monitor the developer activity.

4. OUR APPROACH AND A COMPARATIVE STUDY WITH SIMILAR PAST RESEARCH

4.1: RQ1: What is the frequency distribution of the number of releases across various apps? What is the frequency distribution of the number of days, weeks or months of app releases on Github?

- We generated a visual pattern of the number of releases and their frequencies
- We analyzed and approximated, the number of releases that are common for apps.

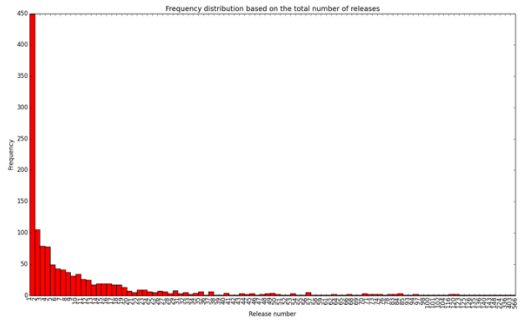

Figure I: Frequency distribution of number of releases

We have observed that the frequency of release is considerably small for apps with large number of releases. If we dig deeper it is clear from the data provided to us, that apps with less than 40 total releases tend to have higher frequencies. Let us see the following graph for 30 total releases. We've scaled the y axis to show a frequency of up to 150 apps and the x axis to 30 releases.

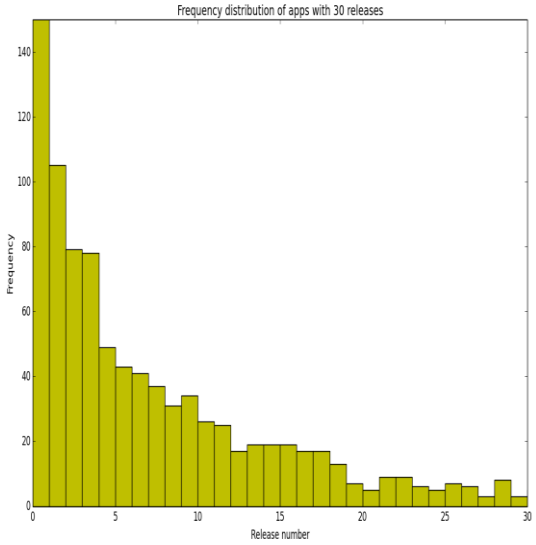

Figure 2: Scaled Frequency distribution of number of releases

A **percentile or quartile** distribution would give us a good understanding of the population of the percentage trend that the release numbers follow in the data provided to us. The maximum number of releases of any app is 566. (There is only one such app). We observe that the 25th percentile or 1st quartile has a value of 2.0. The 50th percentile or 2nd quartile has a value of 17.0. The 75th percentile or 3rd quartile has a value of 53.25. The 100th percentile or 4th quartile has a value of 566

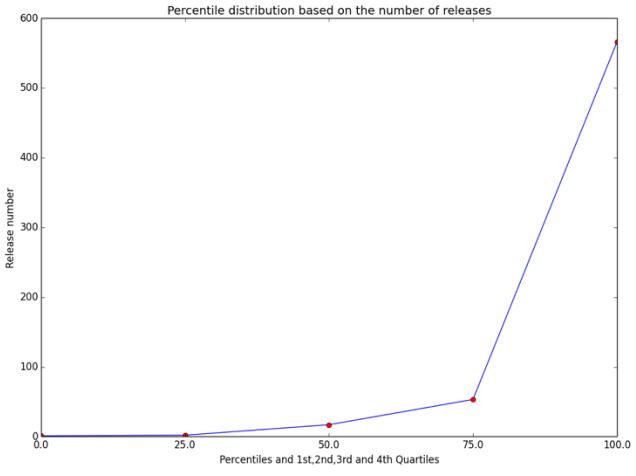

Figure 3: Percentile distribution based on number of releases

Thus we observe that 25% of the apps have less than 3 releases, 50% have less than 10 releases and 75% have less than 30 releases. Next, we did a Pareto Analysis to show that 15 releases already contribute to 80% of the frequency of releases.

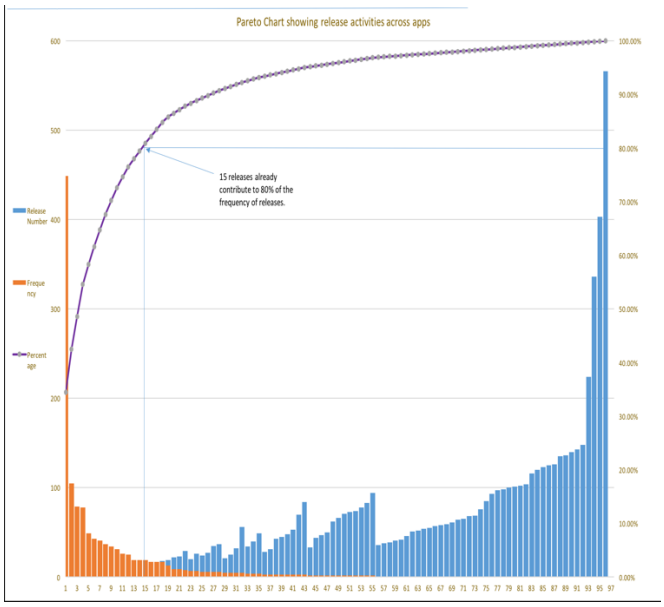

Figure 4: Pareto Analysis showing release activities across apps

It thus becomes necessary to understand the spread of releases (and the median distance) across various apps. So, what kind of an analysis would be ideal?

Let us take a look at the Box and Whisker analysis for apps with the largest number of releases (releases up to 566) and then finally a scaled plot for the apps with a maximum of 40 releases.

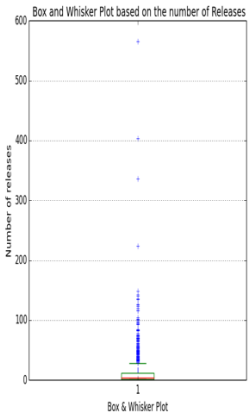

Figure 5(a)

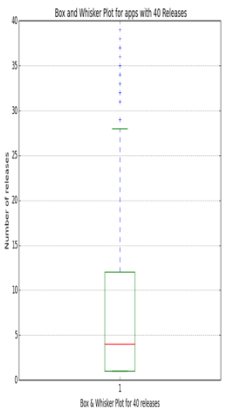

Figure 5(b)

Figure 5(a) & (b): Box and whisker analysis based on the number of releases

These graphs show us an approximation for the spread of the pattern of the number of releases across various apps. We observe that almost 50% of the apps have 4 or less releases. (The median of the spread). Also 75% of the apps have less than 15 releases. And almost all of them (approximately 100%) have less than 30 releases. And very few apps have releases ranging from 31 to 566.

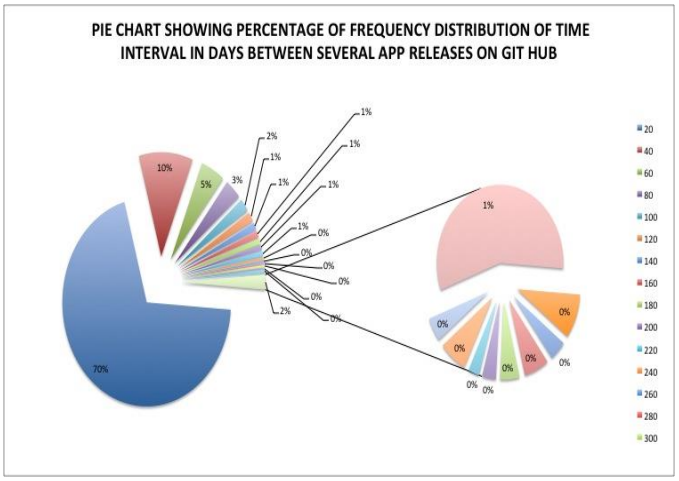

Figure 6: Pie Chart showing time interval between app releases

The pie chart in figure 6 indicates that 70% of the total apps that are taken into account, their release time difference is between 0 to 20 days. It implies that apps make more frequent releases on github between 0 to 20 days. Further analysis has shown that 70% of the revisions take place in first 20 days of app release, 64% of the total apps have their release time difference of up to 5 weeks and 70% of the total apps have their release time difference of up to 2 months. There is a decreasing trend as the number of days/weeks/months are increased.

People are not very good at looking at a column of numbers or a whole spreadsheet and then determining important characteristics of the data. They find looking at numbers to be tedious, boring, and/or overwhelming. Exploratory data analysis(EDA) techniques have been devised as an aid in this situation. Most of these techniques work in part by hiding certain aspects of the data while making other aspects more clear[19]. This research focuses a lot on exploratory analysis We should always perform appropriate EDA before further analysis of our data. This way we get more familiar with our data, check for obvious mistakes, learn about variable distributions, and learn about relationships between variables. That is why this aspect actually contributes in giving our research more visual clarity, to say the least and also builds the foundation for analytical software project management.

4.2: RQ2: How do you do a cluster analysis and model your topics based on the data provided to you and why is it useful in case of commit analysis?

After our data collection process as discussed earlier, we created a word cloud to get an understanding of the most frequent words.

Figure 7: Word cloud showing the most frequent words in commits

We then ran the full LDA transform against the BoW corpus we had , with the number of topics set to 5. And for LSI, we load up the corpus and dictionary from files, then apply the transform to project the documents into the LDA topic space. A topic analysis tool like LDA will try to find N independent word distributions within the word distributions of all the messages. Linear combinations of these N word distributions are meant to represent and recreate the word distributions of any of the original messages. These N word distributions effectively form topics: cross-cutting collections of words relevant to one or more of our commit messages. LDA extracts topics in an unsupervised manner; the algorithm relies solely on the source data, word distributions of messages, with no human intervention.

LDA and LSI are conceptually similar - both are transforms that map one vector space to another. A later analysis of K Means suggested to choose the value of K as the number of topics we need for our analysis. So in this case that was 5 [2]. We didn't know (and didn't have an opinion about) how many topics this corpus should yield so we decided to compute this by reducing the features to two dimensions, then clustering the points for different values of K (number of clusters) to find an optimum value. Gensim offers various transforms that allow us to project the vectors in a corpus to a different coordinate space [8]. The results are as follows. It has been seen, that each topic is made up of a mixture of terms. Thus probability of the terms in each topic are as follows:

i. 2016-11-06 19:19:18,162 : INFO : topic #0 (0.200): 0.045*"usher" + 0.040*"service" + 0.036*"usherservice" + 0.035*"initial" + 0.031*"updaterouteresponsewithetd" + 0.030*"route" + 0.027*"view" + 0.026*"bugfix" + 0.024*"timers" + 0.024*"leaking"

ii. 2016-11-06 19:19:18,163 : INFO : topic #1 (0.200): 0.038*"->" + 0.037*"train" + 0.034*"bart" + 0.033*"new" + 0.033*"special" + 0.032*"refactor" + 0.032*"display" + 0.031*"fix" + 0.028*"destination" + 0.028*"usherservice"

iii. 2016-11-06 19:19:18,163 : INFO : topic #2 (0.200): 0.040*"notification" + 0.036*"stations" + 0.035*"main" + 0.033*"begin" + 0.032*"route" + 0.028*"response" + 0.027*"recent" + 0.026*"about" + 0.026*"messages" + 0.025*"tweakin"

iv. 2016-11-06 19:19:18,164 : INFO : topic #3 (0.200): 0.041*"from" + 0.041*"add" + 0.040*"icon" + 0.034*"service" + 0.033*"improved" + 0.031*"merge" + 0.029*"handling" + 0.029*"crittercism" + 0.026*"update" + 0.024*"response"

v. 2016-11-06 19:19:18,164 : INFO : topic #4 (0.200): 0.050*"service" + 0.038*"visual" + 0.037*"check" + 0.030*"remove" + 0.030*"sjxp" + 0.027*"properly" + 0.026*"timer" + 0.024*"test" + 0.023*"install" + 0.023*"allow"

We also analyzed the top 5 terms of each topic. A very small part of it has been shown in table 2.

Topic Number	***Top 5 terms***
1)	usher, service, usherservice, initial, updaterouteresponsewithetd
2)	->, train, bart, new, special
3)	from, add, icon, service, improved

Table 2

Next we clustered the points in the reduced 2D LSI space using K Means, varying the number of clusters (K) from 1 to 10. The objective function used was the Inertia of the cluster, defined as the sum of squared differences of each point to its cluster centroid. This value is provided directly from the **Scikit-Learn K Means** algorithm. Other popular functions include Distortion (Inertia divided by the number of points) or the Percentage of Variance Explained. Let's see the output of our K-Means graph.

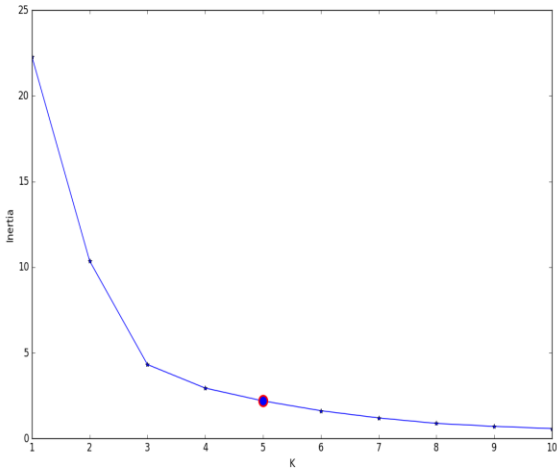

Figure 8: A K-Means Graph

- The elbow point happens here for K=5 and is marked with a red dot in the graph below.
- K-means is a simple unsupervised machine learning algorithm that groups a dataset into a user-specified number (k) of clusters [3].
- Therefore, when using k-means clustering, users need some way to determine whether they are using the right number of clusters.
- One method to validate the number of clusters is the *elbow method*. The idea of the elbow method is to run k-means clustering on the dataset for a range of values of *k* (say, *k* from 1 to 10 in our analysis above), and for each value of *k* calculate the sum of squared errors (SSE).

We then re-ran the K Means algorithm with K=5 and generated the clusters in figure 9.

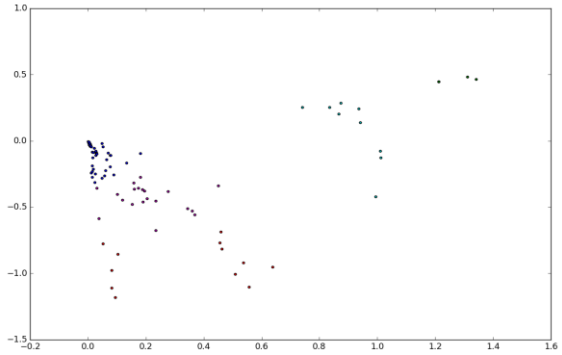

Figure 9: Generating Clusters

The cluster analysis gives us the following advantages: Fast, robust and easier to understand. Gives best result when data set are distinct or well separated from each other.

Previous research also shows that LSI uses a Singular Value Decomposition (SVD) of the term-document matrix X to identify a linear subspace (so-called latent semantic space) that captures most of the variance in the data set. PLSI models each word in a document as a sample from a mixture model, where the mixture components are multinomial random variables that can be viewed as representations of topics. We need to use LapPLSI, in order to visualize the hidden topics in the document. Next we conducted a Principal Component Analysis (PCA). It is useful for eliminating dimensions. So, we've plotted the data along a pair of lines: one composed of the x-values and another of the y-values. PCA components are the underlying structure in the data. They are the directions where there is the most variance, the directions where the data is most spread out.

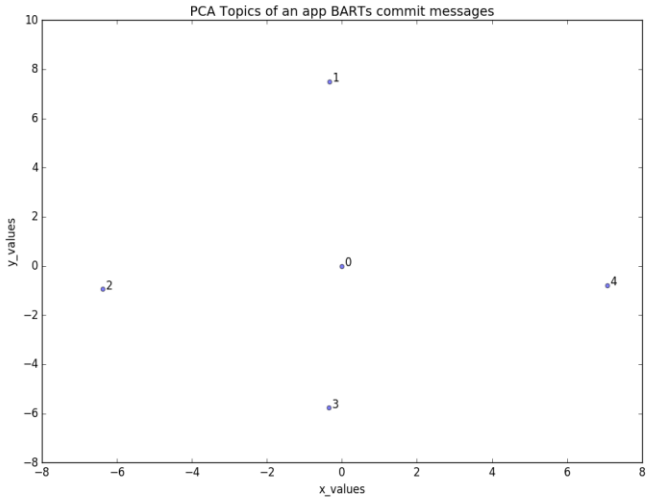

Figure 10: A PCA graph based on the topics

When we get a set of data points, we can deconstruct the set into eigenvectors and eigenvalues. Eigenvectors and values exist in pairs: every eigenvector has a corresponding eigenvalue. An eigenvector is thus a direction (vertical, horizontal, 45 degrees etc.). An eigenvalue is a number, telling us how much variance there is in the data in that particular direction, in the results generated the eigenvalue is a number telling us how spread out the data is on the line. **The eigenvector with the highest eigenvalue is therefore the principal component.** Thus we computed our PCA graph (figure 10) for the topics generated from the commit messages of an application named batphone.

We also created a network graph in figure 11, by increasing the number of topics to 10, so that we could get a better visualization which clearly illustrate the links between the top 10 Topics. It shows us how strong the connections between the topics are and where the connections are most. Thereby implying which topics tend to cluster more.

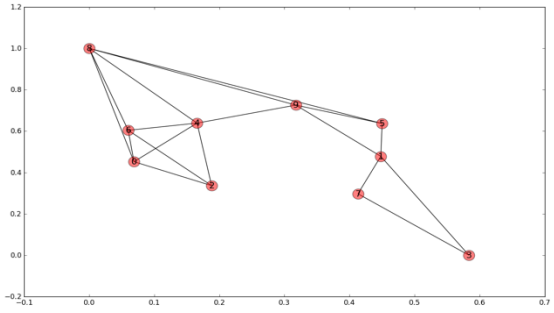

Figure 11: A network graph based on 10 topics

We also performed a Hierarchical cluster Analysis on topics generated from the commit messages of the BART application. The dendrogram in figure 12 is a tree diagram, showing taxonomic relationships among the topics.

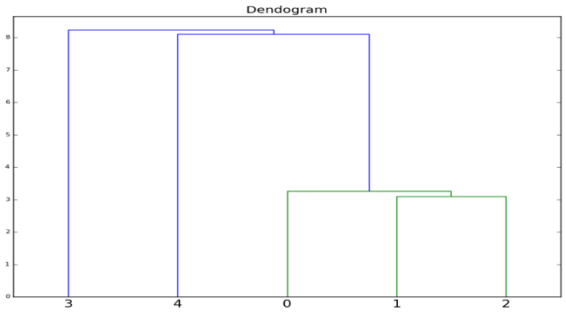

Figure 12: A Dendrogram for the 5 topics generated from the BART application commits

The arrangement of the clades tells us which leaves are most similar to each other. The height of the branch points indicates how similar or different they are from each other: the greater the height, the greater the difference. In our analysis, we compare the distribution of different topics among whole texts or segments of texts. Thus our results: Chunks 1 and 2 are most similar to each other than they are to 3 and 4. Similarly we see that chunk 4 is a bit similar to the combination of chunks 0 and (1,2). Finally we can say that chunk 3 is most different from chunks 1 and 2.

Blei et al.[5,6] showed the importance of topic modelling in multiple scenarios. How immensely LDA helps in topic modelling. His contributions provide us with valuable insights for our research. Our research on topic modelling is extremely crucial in real world scenarios, because to have a better way of managing the explosion of electronic document archives these days, it requires using new techniques or tools that deals with automatically organizing, searching, indexing, and browsing large collections. On the side of today‘s research of machine learning and statistics, it has developed new techniques for finding patterns of words in document collections using hierarchical probabilistic models. These models are called topic models. Discovering patterns often reflect the underlying topics that united to form the documents, such as hierarchical probabilistic models are easily generalized to other kinds of data; topic models have been used to analyze things rather than words such as images, biological data, survey information and even commit messages. The four methods that topic modeling rely on are Latent semantic analysis (LSA), Probabilistic latent semantic analysis (PLSA), Latent Dirichlet allocation (LDA) and Correlated topic model (CTM). The basic drawback of unsupervised machine learning is that labelling the topics is a tiresome prospect. And a certain amount of knowledge in the field of software engineering is necessary to label these GIT commit message topics.

5. FUTURE WORK

In the next phase we would be manually labelling the several topics generated from the multiple commit messages of innumnerable apps. We would also check if automated topic labelling is a feasible option or not [18]. We would also be analyzing these labelled topics to figure out the software development lifecycle process followed by individual apps so that we can conduct an analysis of the most frequent lifecycle processes followed across the apps. We would also like to check for the aliveness of our apps by digging into the commit dates and looking for the inteval since the last commit. If possible, we would further strengthen our analysis by looking into the number of commits and check for the mention of pull and push requests inside the messages. This is might help us in two things. To check how popular an app is among developers and thus if populatory correlates with characteristics of a repository.

6. REFERENCES

[1] Recovering Documentation-to-Source-Code Traceability Links using Latent Semantic Indexing (Andrian Marcus, Jonathan I. Maletic)

[2] The key technology of topic detection based on K-means (Shengdong Li ; Xueqiang Lv ; Tao Wang ; Shuicai Shi)

[3] Analyzing Factors Impacting Open-Source Project Aliveness (Rudi Chen; Ivens Portuga)

[4] An Exploratory Study of the Pull-based Software Development Model (Georgios Gousios; Martin Pinzger; and Arie van Deursen)

[5] Latent Dirichlet Allocation (David M. Blei; Andrew Y. Ng ; Michael I. Jordan)

[6] Dynamic Topic Models (David M. Blei; John D. Lafferty)

[7] Applying Discrete PCA in Data Analysis (Wray Buntine ; Aleks Jakulin)

[8] https://radimrehurek.com/gensim/

[9] A Survey of Topic Modeling in Text Mining (Rubayyi Alghamdi; Khalid Alfalqi)

[10] An Insight into the Pull Requests of GitHub (Mohammad Masudur; Rahman Chanchal K. Roy)

[11] What Do Large Commits Tell Us? A taxonomical study of large commits (Abram Hindle; Daniel M. German; Ric Holt)

[12] http://www.itl.nist.gov/div898/handbook/eda/section1/eda11.html

[13] Release Practices for Mobile Apps –What do Users and Developers Think? (Maleknaz Nayebi; Bram Adams; Guenther Ruhe)

[14] Modeling Hidden Topics on Document Manifold (Deng Cai, Qiaozhu Mei, Jiawei Han, Chengxiang Zhai)

[15] Automated topic naming to support analysis of software maintenance activities (Abram Hindle ,Neil A. Ernst, Michael W. Godfrey ,John Mylopoulos)

[16] Understanding the Factors that Impact the Popularity of GitHub Repositories (Hudson Borges, Andre Hora, Marco Tulio Valente).

[17] Applying Discrete PCA in Data Analysis (Wray Buntine, Aleks Jakulin)

[18] Automated Topic Naming Supporting Cross-project Analysis of Software Maintenance Activities (Abram Hindle, Neil A. Ernst, Michael W. Godfrey , John Mylopoulos)

[19] Applications, basics, and computing of exploratory data analysis (PF Velleman, DC Hoagi)