# Evolutionary Topic Modelling for User Issues

CS454 AI Based Software Engineering Final Project

Group 6: Inhyuk Kwon, Ivan Chuang, Jaegeon Jo, Nicholas Tay

Github : https://github.com/vnck/EvoTopic

## Motivation

Today, software reports are often written in the form of natural language texts. This is because software problems or requirements have to be implemented by another human and thus, these desired functionalities or problems faced while using the software should be explained in natural language. The person can then understand what the requirements are and implement it accordingly. However, when projects grow and get larger, it results in huge quantities of such reports. In huge quantities, analysis of such reports become computationally expensive. Therefore, in order to try to solve this problem, we propose a topic modelling technique to cluster reports and extract trends that can guide bug fixing and requirement construction for software evolution. Our aim is to be able to provide meaningful clusters so as to alleviate this problem that software development face today.

## Data Collection

Data collection is important for our project as it concerns providing meaningful data that would allow us to build a topic modeller that can provide meaningful clusters. We chose to make use of two main sources for our data, Github issues as well as Android app reviews. As for the Github issues, a python script was written using beautifulSoup4 library for web scraping. This allowed us to extract the data which came mostly from https://github.com/facebook/react/issues. We thought that open source Github repository issues would be the best form of data we can get for software development problems. As for the Android app reviews, we were able to find already extracted datasets so we made use of the datasets provided here: https://github.com/amitt001/Android-App-Reviews-Dataset Note that for this project, we have chosen to only make use of negative reviews for Android apps as our focus was to search for software bugs or problems faced by users. Good reviews would unlikely explain or document any problems in the app.

## Search Problem

We used LDA(Latent Dirichlet Allocation) for topic modelling. LDA is a generative statistical model. It uses two kinds of probability distribution. First one is distribution of topic in documents. This distribution is dirichlet distribution. So, It needs hyperparameter α which is prior belief of probability distribution. Second one is distribution of words in documents. This distribution is also dirichlet distribution. Hyper parameter is called β which is prior belief of probability distribution of words. LDA doesn't find out optimal number of topics. We have to specify the number of topics when running the LDA. Therefore, our search problem is optimizing 3 hyper-parameters - number of topics(n), hyper-parameter for topic distribution(α), hyper-parameter for word distribution(β). There is no efficient heuristic for this hyper-parameter optimization. We decide to use genetic algorithm(GA) for our project.

## Gene Representation

As we mentioned above, there are 3 hyper-parameter for tuning. First one is number of topics(n). Range of n is 1 to number of words. It represented as an integer. Second one is prior belief of probability of each topic(α). It represented as an array of probability which has length n. Last one is prior belief of probability of each word(β) in each topics. By definition, the size of β is number of words x number of topics. But we thought that It is too large to optimize. So, we assume that prior distribution of words is same in all topics.

## Genetic Operators

### Selection Operator

We applied elitism for selection operator. We selected top 20% of population and generated remaining 80& of population with crossover and mutation operator.

### Crossover Operator

It is desired that generated new gene has goodness of two parent genes in crossover operation. In our gene representation, α and β represent probability distribution. The relative value is more important than absolute value. To maintain the characteristic of probability distribution, we decided to use average. We rounded up when averaging n which is number of topics.

## Mutation Operator

Using average in crossover operator causes fast convergence of population. It is easy to stuck in local optimum. Because of this reason, we need mutation. But we did not want to mutate gene dramatically. However, sometimes dramatic mutation is required for escaping local optimum. So, we used two kinds of mutation - Large mutation and Small mutation. Small mutation is more frequently occured than other mutation but the mutated range is small. Method of small mutation is choosing k elements from α or β and shuffle them. Method of large mutation is redistributing all elements in α and β.

## Fitness Function

Topic modelling is an unsupervised approach. But we still desire a metric to determine the goodness of our topic clusters. We mainly considered two kinds of fitness function. First one is topic coherence measure. We used u_mass coherence measure which is known as an intrinsic measure.

$$\sum_{i<j} log\left(\frac{1+D(w_i, w_j)}{D(w_i)}\right)$$

$w_i$ and $w_j$ are words in corpus. D(w) is frequency of word w in corpus. $D(w_i,w_j)$ is frequency of appearance of $w_i$, $w_j$ in the same document. Constant is used for smoothing count. When value of this function goes to 0, it means $D(w_i,w_j)$ and $D(w_i)$ are almost same. So $w_i$ and $w_j$ are highly related to each other. On the other hands, When value of this function goes to negative infinity, it means $D(w_i,w_j)$ is much smaller than $D(w_i)$. So $w_i$ and $w_j$ are not so related in this case. Therefore, we have to maximise this value for good clustering.

Second one is silhouette score. It can be calculated by following formula.

$$max_k\left(\frac{1}{|C_k|} \sum_{i\in C_k} \left(\frac{b(w_i) - a(w_i)}{max(a(w_i),\ b(w_i))}\right)\right)$$

$w_i$ and $w_j$ are word in corpus, $C_k$ is kth corpus. $a(w_i)$ is the mean distance between $w_i$ and all other words in the same corpus. We used cosine distance. $a(w_i)$ represents similarity of words in the same corpus(i.e. coherence). Small value of $a(w_i)$ means that corpus has high coherence. $b(w_i)$ is the smallest mean distance of $w_i$ to all words in any other corpus. b($w_i$) represents how far from nearest corpus(i.e. separation). Therefore, silhouette score goes to 1 when corpuses have high coherence and high separation. And It goes to -1  when corpuses have low coherence and low separation. We have to maximise the silhouette score for good clustering.

We know that silhouette score represents more information than umass coherence measure. So using silhouette score might be give us better result than umass coherence measure. We tried both of fitness function. But we noticed that computational cost of silhouette score is much more higher than umass coherence measure. Our search problem is hyper parameter optimization. We have to calculate lot of fitness of population. Therefore, we decided to use umass coherence measure for fitness function.

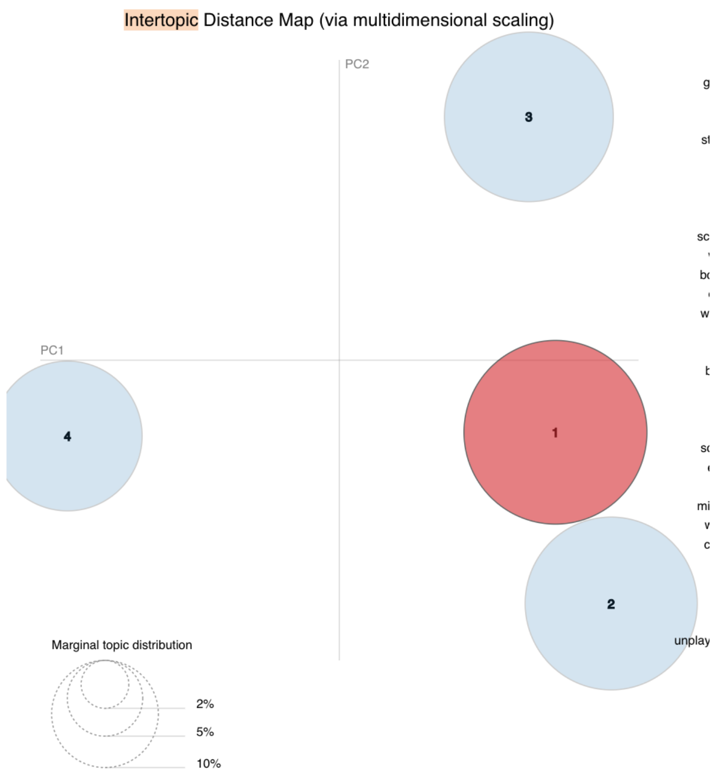
## Results and Analysis

### React Github Reported Issues Dataset

We scraped approximately 4000 github issues from the Facebook React Repository. Using the title and the first comment of each issue, we applied text normalisation and converted them into a Bag of Words (BOW) vectorised format.

Our hyperparameters for the automatic clustering algorithm was tuned as such: population size of genes was set to 30, fitness evaluation budget was set to 1000, mutation and crossover rates were set to 0.2. Afterwhich, we produced and LDA model for each gene from the population and the BOW vectors, and evaluated the goodness of the clustering using coherence topic score.

Our approach discovered that the optimal number of topics was 5 and the topic clusters produced was visualised in as such:



We observe that the clusters have minimal overlap in the visualisation, whose axes are the principal components of the clusters. Therefore, the model had found a good separation between each cluster. Then, we looked at the contents of each cluster by extracting the most representative words of each cluster. This was done through TF-IDF ranking, where TF-IDF is representative of the importance of a word to a cluster. From the keywords, we were able to label each cluster. Thus, our topic clusters can be observed as such:

| Topic | Keywords | Label |
|---|---|---|
| 0 | component, babel, return, div, const, render | Component Rendering |
| 1 | component, bug, version, prop, current, behavior | Prop Behaviour |
| 2 | object, http, var, www, src, request | API |
| 3 | module, expected, behavior, case, equal, node_modules | Node Modules |
| 4 | main, extension, chrome, build, object, chrome_extension | Chrome Extension |

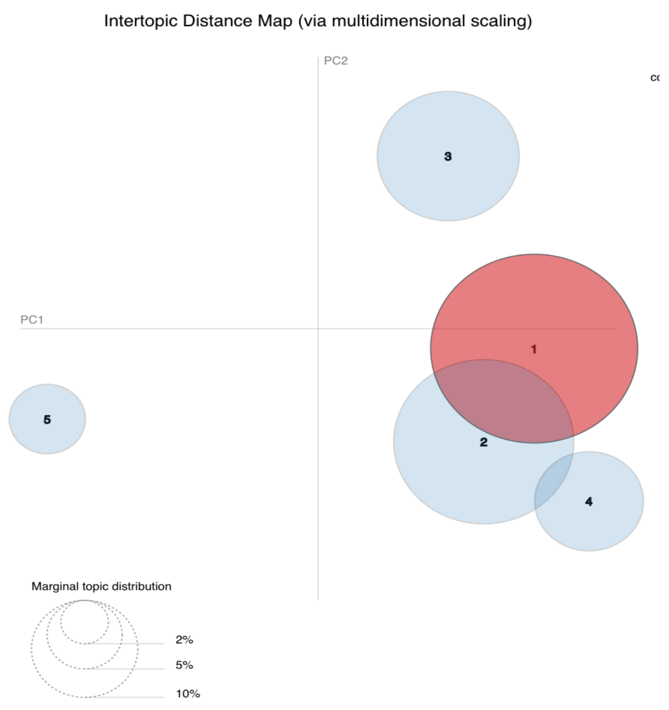
Hence, extraction of such keywords can help in identifying the components for software evolution and highlight key problems in the software. The topic clusters were also ranked by the count of documents present in each cluster, an indication of the importance of the issue in the cluster. Such ranking of clusters can be used to prioritise software tasks.

Thus, we see that our approach is sufficient for automatic extraction of key issues in software projects and is able to extract issues and rank them so as to serve as a helpful guide for software development.

## Android App Reviews Dataset

We scraped approximately 4000 negative app reviews from the android app store for various apps. We determine that a negative review is one with 2 or less stars given. We applied the same pre-processing procedure as above, and converted our textual data into BOW vectors from analysis.

Our hyperparameters for the automatic clustering algorithm was tuned as such: population size of genes was set to 30, fitness evaluation budget was set to 1000, mutation and crossover rates were set to 0.2. Afterwhich, we produced and LDA model for each gene from the population and the BOW vectors, and evaluated the goodness of the clustering using coherence topic score.
Our approach discovered that the optimal number of topics was 4 and the topic clusters produced was visualised in as such:
We observe that the clusters have minimal overlap in the visualisation, whose axes are the principal components of the clusters, which was sufficient for effective clustering. Then, we looked at the contents of each cluster by extracting the most representative words of each cluster. This was done through TF-IDF ranking, where TF-IDF is



representative of the importance of a word to a cluster. From the keywords, we were able to label each cluster. Thus, our topic clusters can be observed as such:

| Topic | Keywords |
|---|---|
| 0 | Level, Boring, Stupid, Screen, Love |
| 1 | Fix, Update, Ad, Working, Problem, Suck |
| 2 | Money, Waste, Bad, Keep, Download |
| 3 | Work, Version, Hate, Freeze, Refund |

We observe that due to the nature of app reviews, there were less technical words present as keywords as compared to with our React dataset. Instead, many of them were emotive words such as Boring, Stupid, Hate, etc. Hence, while our process was not as useful for discovering technical issues in the data, we can instead use our approach for semantic analysis of the data and gain a sense of how users are feeling towards our apps, and after identifying technical components within a topic cluster, associate it with a general emotion of that cluster. This can help in software development as users might feel differently to each part of the software and through this automatic process, we might be able to separate these emotions and use them to guide user experience design.

## Limitations

### LDA's high dependence on initial random topic allocation

LDA is greatly dependent on the initial seeds (initial topic allocation in our case). They are always randomly chosen due to the fact that the sampling from prior is always done randomly as well. Hence, we feel that exploring the way to somehow control this randomness worth the try to improve the results. One plausible way to do it is to minimise the variance of the data by running multiple incidents for each gene and obtain the average result although it incurs further computational cost.

### Qualitatively crafted issue titles

Currently, the issue titles are manually crafted with human efforts to come out with a meaningful summary of the top k topics ranked by their topic confidence level. Even if human supervision is the best method for this task, it is very expensive and sometimes, the human inspectors are also prone to making mistakes, making wrong judgement, especially when the domain of issues is not of the inspector's profession or the nature of issues is too complex to comprehend.

### Noticeable amount of redundant/irrelevant data

Currently, our web scripting collects every issue found in the link. But some of these issues contain redundant or irrelevant data for that particular issue. Filters could be applied to reduce noise data and extract only the relevant information to help with the LDA algorithm.

## Future Works

### Explore other SBSE algorithms

For further works, one of the things that can be done is to explore other SBSE algorithms such as hill-climbing and simulated annealing that might possibly suit topic modelling problem better than the genetic algorithm we used in our project.

### Explore other fitness function

We can also explore other metrics such as silhouette value that can possibly outperform topic coherence in measuring the fitness. Although we made an attempt to get along with silhouette value as metric, the computation cost was too high that topic coherence measure was implemented instead for its efficiency. Better similarity metric better represent the distance between two clusters which subsequently improve the algorithm's parameter tuning ability.

### Machine-learned topic summarizer

The problem of limitations on manually crafting issue title can be solved with machine-learned topic summarizer which will be able to summarize the topic words to form an issue title based on the knowledge it has learned from a vast amount of documents or issues in various fields.

## Conclusion

By tuning the hyperparameters of LDA algorithm, satisfactory results in extraction of representative topics of issues in the order of importance seem promising in a sense that this can be applied to the software engineering problems that require monitoring of the feedbacks from the users for various aspects such as bug fixing, recommendations and etc. This will greatly improve on the efficiency of the process of digging out the most urgent or important issues faced by the users, and as well as the designing and implementation of the solution to satisfy both the users and the engineers’ needs.