Extracting and Analyzing JavaScript Snippets from Stack Overflow Data Dump

Oleksandra Liutova

***Abstract*—This project focuses on extracting JavaScript snippets from stackoverflow web platform and performing clustering and visualisation of this data. First, filtering out unmeaning snippets and code in different languages was performed. Then, snippets were vectorized using Bag-of-words and Tf-idf techniques. Clustering was done using K-Means algorithm. For visualisation, data was processed by truncated SVD and t-SNE sequentially. Obtained results were analysed.**

***Index Terms*—Data Preprocessing, Javascript Snippets, Clustering, Feature Extraction**

I. Introduction

Stack Overflow is a web platform, which enables asking and answering questions connected with computer programming. It contains over ten million questions and its data is publicly available[1]. This data has been used in this project. This work was done in close collaboration with PRL@PRG (Programming Research Laboratory, Prague). The main goal of this project is to perform clustering of JavaScript snippets present in posts. In order to do this, first of all it is necessary to preprocess data containing information about posts. That is, to extract JavaScript snippets from posts and filter them in order to get rid of senseless snippets or code in languages other than JavaScript. Than it is necessary to represent extracted snippets in some way and finally perform clustering on them.

II. Methodology

A. Extracting JavaScript snippets from text

The data available on stackexchange is in XML format and PRL@PRG team has processed it and stored data in a MySQL database. I have received access to this database and it is the resource I have been working with in this project. The table I worked with contains all the posts with tag "JavaScript" from the original dataset. Raw snippets of code were extracted from column "Body", which contains text of post. This was done by extracting text in tags <code>...</code>. Due to some encoding errors it was later revealed that characters '<', '>' and new line were substituted in extracted text by sequences 'lt;', 'gt;' and 'xA;' accordingly, so it was necessary to replace them with correct characters.

B. Filtering snippets

Extracted snippets were noisy, a lot of snippets consisted of just one or a few words, having no informational value. Some snippets were written in HTML, often used in combination with JavaScript. Here are examples of what some snippets looked like: *'1 to 10', 'check()', 'window.performance.timing', 'display', 'none' ,'e', 'do bar'*. This is why it was necessary to perform two kinds of filtering on extracted data: filtering out meaningless snippets and code written in other languages.

1) Filtering meaningless snippets: Of course, a decision whether a certain snippet is meaningless or not is very subjective. However, for this task a simple filtering algorithm was used. Filtering was performed based on size of snippet. Too small snippets were filtered out as meaningless. The threshold was set to 20 symbols. As a result 753323 out of 2438116 snippets (31%) were filtered out. Here is an example of some of filtered out snippets: *'OnClick', '<ul>', 'HttpWebRequest', 'PHP', '0', 'map()', 'com = ... '*.

2) Filtering snippets in language other than JavaScript: As part of this project I have investigated different methods of detecting programming languages of code. This turned out to be a very complicated problem itself. This particular problem is in addition complicated by the fact, that most of snippets are very small, and therefore performing statistical analysis on them (like counting keywords or calculating frequency of punctuation characters) would be unreliable. That is why I have decided to apply a straightforward filtering algorithm, that detects if a snippet begins or ends with characters '<' or '>'. This simple check filters out a lot of XML code, that is often used combined with JavaScript, and therefore is relatively frequent in posts with 'JavaScript' tag. As a result, 410857 out of 1684793 snippets were removed, that is 24.4% of all snippets obtained after filtering meaningless ones.

C. Snippets representation

Representing code is a pretty complex task. Ruffly speaking, code is text, so we can use text processing techniques to work with code. However, code is more than just text and has some deeper meaning. I used different feature extraction techniques on filtered snippets in this chapter.

1) Bag-of-words: Bag-of-words is a popular model used in text processing. It constructs a dictionary of all words used in the corpus and represents each document as a vector with numbers of occurrences of each word in given document. Here is an example of words from retrieved dictionary: *'class', 'console', 'time', 'div', 'log', 'submit', 'form', 'href', 'posts', 'query', 'customer', 'localhost', 'document', 'prototype', 'photo', 'font', 'and', 'function', 'hello', 'data', 'index', 'this', 'position', 'border', 'click', 'from', 'not', 'size', 'return', 'get', 'element'*. The result of this algorithm is a (1273936, 533222) matrix, where 1273936 is the number of snippets and 533222 is the number of features.

2) Tf-idf: Tf-idf (term frequency-inverse document frequency) is another text processing technique. It is somewhat more intelligent than bag-of-words and calculates term frequency weighted by term's general frequency in the whole corpus. The logic is such, that if the word is frequent in all documents (e.x. "the" or "she"), than it does not contribute much to particular document's topic. But it is a question if this behaviour is desired when processing code. In code we would expect keywords to be relatively frequent, while variables names should be more diversified. Some variables names (like "i" or "n") are very frequent and carry almost no information. Meanwhile variables like "number_of_iterations" or "meaningful_snippets" are

not frequent, but carry a lot of information. This is why this approach should give interesting and meaningful results while processing code. The result of this algorithm has the same structure as result of Bag-of-words.

D. Clustering

For clustering purposes I have used K-Means, a very popular and simple clustering algorithm. The user has to provide number of clusters as parameter to this algorithm, and I have performed clustering for 10, 20, 30 and 40 clusters.

E. Visualisation

The goal was to visualise clustered snippets in 2D space. For this purposes I used t-SNE (t-distributed stochastic neighbor embedding) algorithm. It's a nonlinear dimensionality reduction algorithm, often used for embedding high-dimensional data into a space of two or three dimensions for visualisation. It models each high-dimensional object by a two-dimensional point in such a way that similar objects are modeled by nearby points and dissimilar objects are modeled by distant points. t-SNE is a computationally demanding algorithm and in order to be able to perform it on my computer I had to reduce number of snippets to 20,000 and to apply another dimensionality reduction algorithm - truncated SVD - before t-SNE. Truncated SVD is very similar to PCA[2] and I chose to use it because it is more efficient while working with sparse matrices, it accepts scipy.sparse matrices without the need to densify them. Truncated SVD implementation in scikit-learn enables calculating explained variance ratios and choosing corresponding number of components. Here are explained variance ratios for first 20 components: *0.65010076 0.08743116 0.02622701 0.02058529 0.01442987 0.01238455 0.0085035 0.00689957 0.00661005 0.00487312 0.00482981 0.00430674 0.00384035 0.00348529 0.00337224 0.00301293 0.00283627 0.00246678 0.00233588 0.00225235*. The sum of explained variance ratios for first 20 components is 0.870783511297, which is sufficient. Let me emphasize that truncated SVD and t-SNE are used only for preparing data for visualisation, meanwhile clustering was performed on original not transformed features.

III. Results

A. Visualisation of clustered snippets

Here are visualisations for Bag-of-words feature extraction method and corresponding graphs of data distribution in clusters:

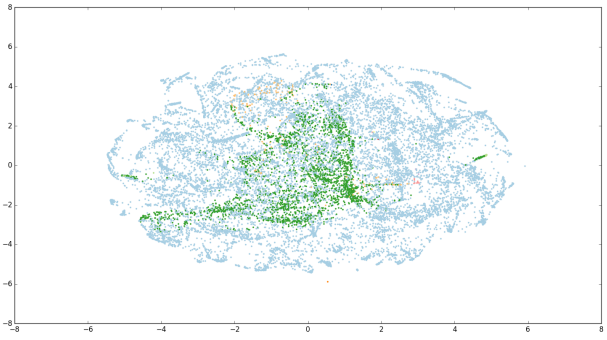

Fig. 1: Bag-of-words, 10 clusters

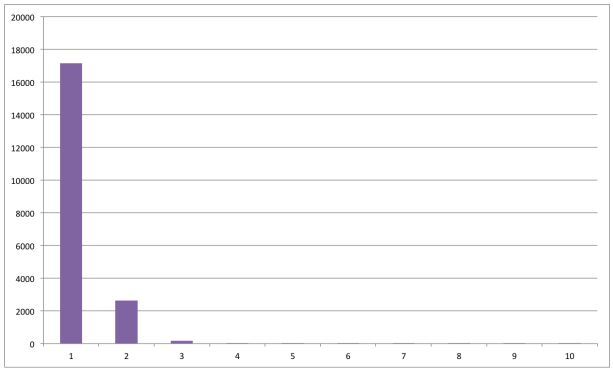

Fig. 2: Distribution of data in clusters, 10 clusters

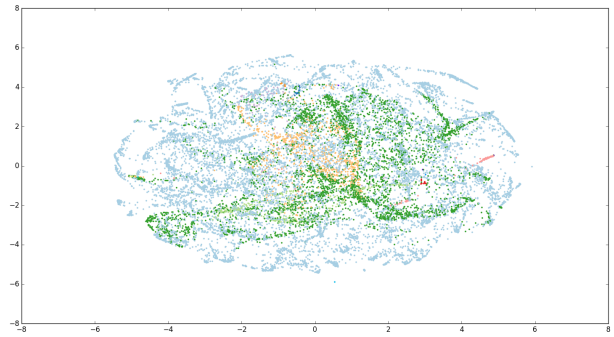

Fig. 3: Bag-of-words, 20 clusters

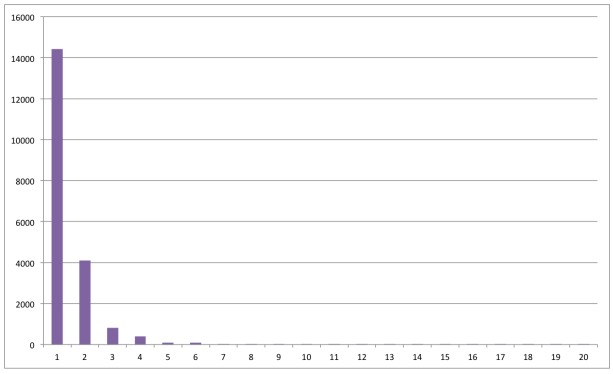

Fig. 4: Distribution of data in clusters, 20 clusters

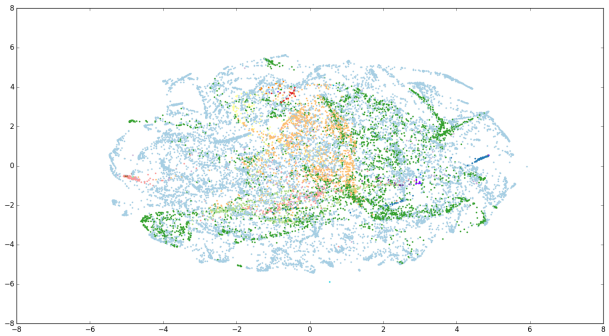

Fig. 5: Bag-of-words, 30 clusters

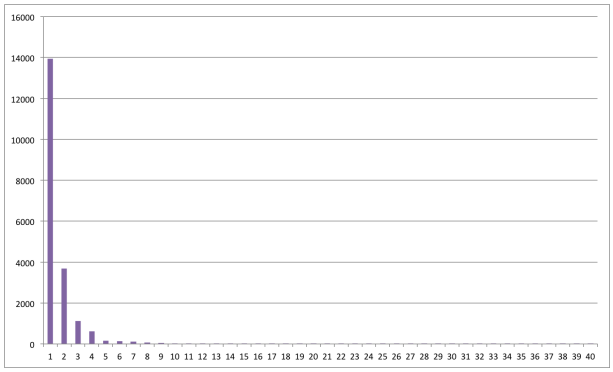

Fig. 8: Distribution of data in clusters, 40 clusters

Here are visualisations for Tf-idf feature extraction method and corresponding graphs of data distribution in clusters:

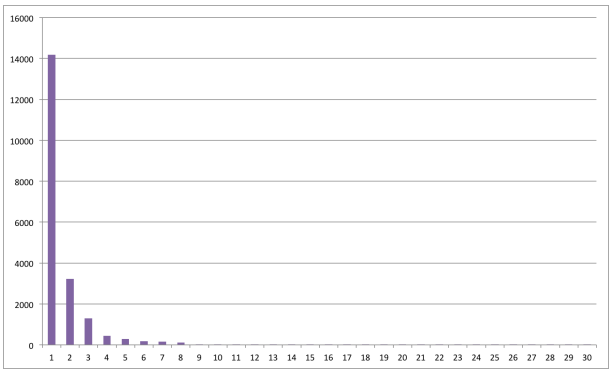

Fig. 6: Distribution of data in clusters, 30 clusters

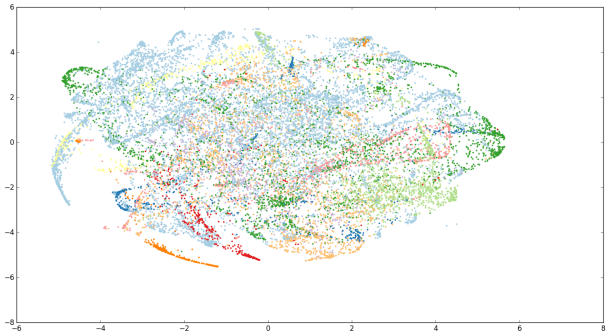

Fig. 9: Tf-idf, 10 clusters

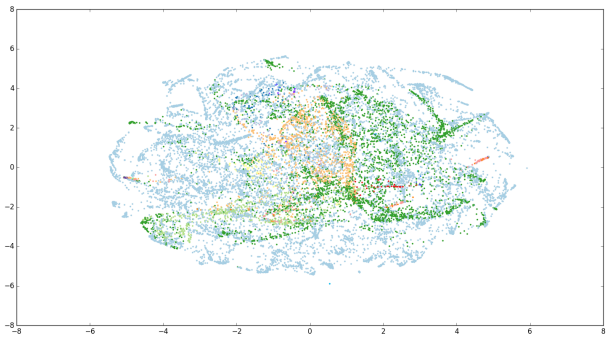

Fig. 7: Bag-of-words, 40 clusters

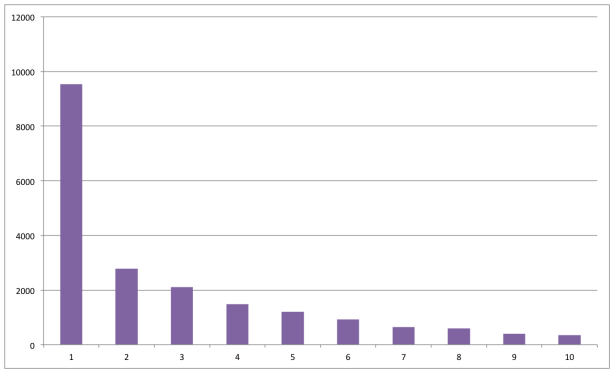

Fig. 10: Distribution of data in clusters, 10 clusters

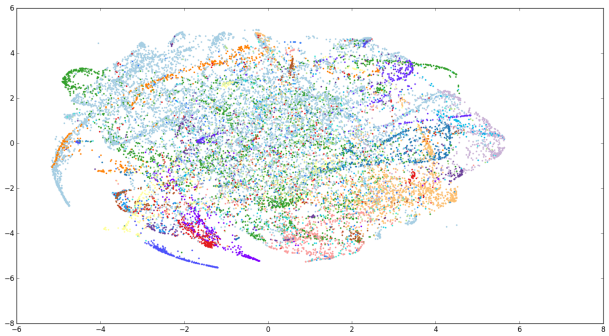

Fig. 11: Tf-idf, 20 clusters

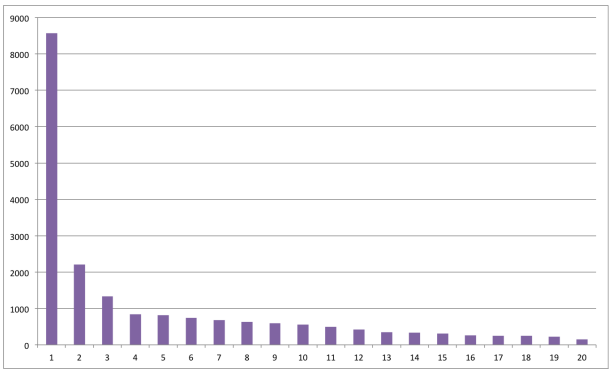

Fig. 12: Distribution of data in clusters, 20 clusters

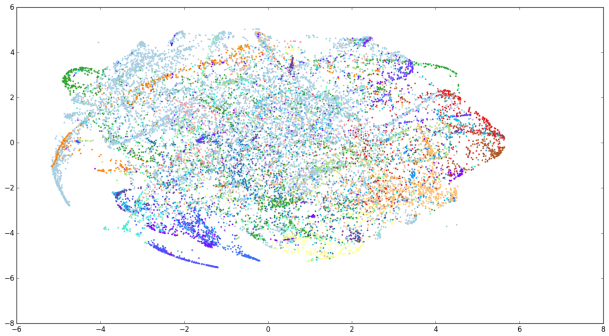

Fig. 13: Tf-idf, 30 clusters

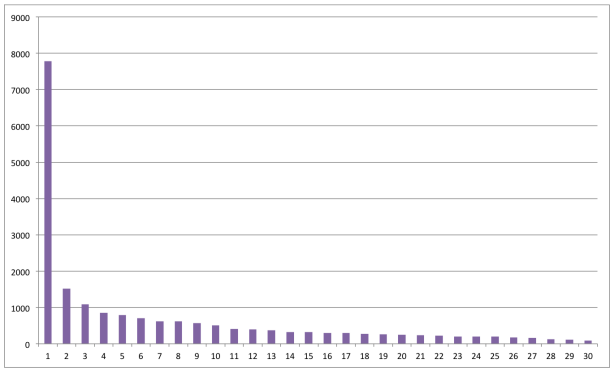

Fig. 14: Distribution of data in clusters, 30 clusters

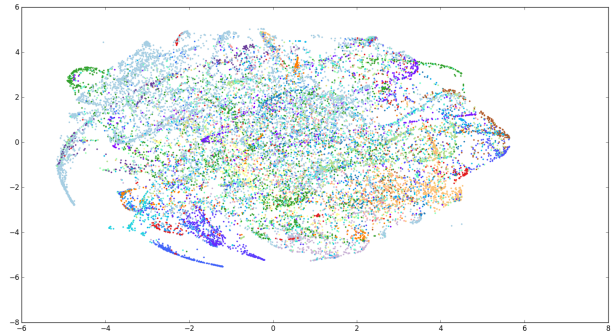

Fig. 15: Tf-idf, 40 clusters

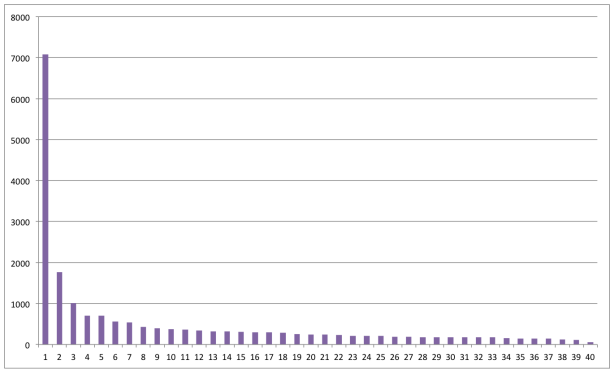

Fig. 16: Distribution of data in clusters, 40 clusters

B. Clusters' contents analysis

The graphs above show a difference between Bag-of-words and Tf-idf methods. Tf-idf features are clustered more smoothly, while for Bag-of-words most features are contained in the first few clusters and each cluster in the second half contains only 1 or 2 items. Those snippets, that were clustered into separate clusters share one common feature: they are all relatively long. Each of them contains 200 lines of code on average. They contain a lot of different words, and therefore they would be relatively far away from all other points in multi-dimensional space of Bag-of-words features. This is the reason for their being clustered into separate clusters. The situation is different for Tf-idf, because feature vectors are based on terms' frequencies instead of absolute counts.

It is hard to define what exact features distinguish snippets from one cluster from snippets from another one. But it's possible to notice some patterns. For example, the smallest cluster out of 40 for Tf-idf contains a lot of snippets that mostly operate with windows, pop-ups or dialogs. So those terms are frequent in those snippets.

C. CPU time measuring

In this chapter I will present brief analysis of computational time of algorithms used in this project. All computations were performed on OS X, 2.5 GHz i7, 16 GB RAM, 512 GB SSD.

Figure 17 shows computational time of feature extraction, applying truncated SVD and applying t-SNE for Bag-of-words and Tf-idf. T-SNE is the most computationally demanding algorithm used in this project, not only from time's perspective, but also memory's. It's important to note than feature extraction was performed on the full set of snippets (containing 1,273,936 snippets), while truncated SVD and t-SNE were performed on reduced set of 20,000 snippets.

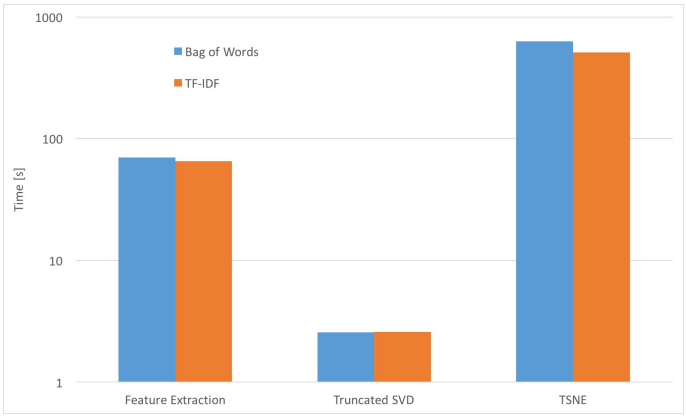

Fig. 17: Computational time of feature extraction, truncated SVD and t-SNE

Figure 18 shows computational time of K-Means for Bag-of-words, Tf-idf and different numbers of clusters. Features extracted using Tf-idf are generally processed longer than those from Bag-of-words. The bigger number of clusters, the longer algorithm runs.

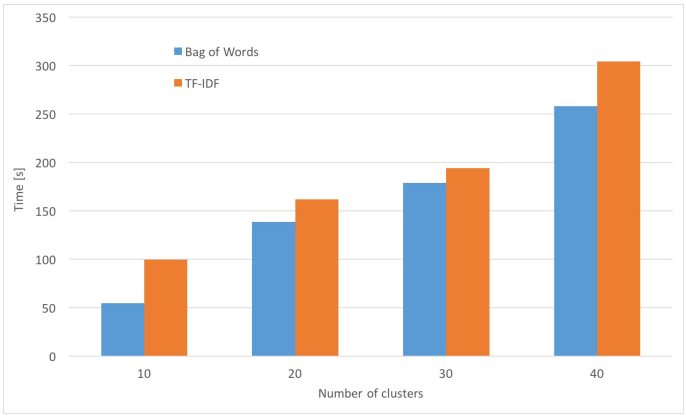

Fig. 18: Computational time of clustering

IV. Discussion

This project is the first step to understanding the structure of data on stackoverflow, particularly JavaScript snippets. There are a lot of improvements that can be done to this analysis and a lot of fields are left yet unexplored. First of all, it would be reasonable to run this analysis on the full set of million snippets. In this project I used feature extraction methods that are used in text processing and in the future more advanced methods for code representation can be explored. Also, methods used for detecting meaningful snippets and filtering JavaScript snippets can be improved as well.

V. Conclusion

In this project I have explored different methods of data preprocessing, performed data clustering, visualisation and results analysis. This project has achieved its goal and gives a general insight into structure of JavaScript snippets on stackoverflow platform.

References

[1] *Stack Exchange Data Dump* [online]. Stack Exchange Data Dump, 2016 [viewed. 2016-12-01]. Available at:https://archive.org/details/stackexchange

[2] *Truncated singular value decomposition and latent semantic analysis* [online]. scikit-learn developers (BSD License), 2010 - 2016 [viewed. 2017-01-20]. Available at:http://scikit-learn.org/stable/modules/decomposition.htmltruncated-singular-value-decomposition-and-latent-semantic-analysis