Identifying keywords, topics and summaries using text data from Stack Exchange sites

Sayan Das

1. Introduction and background

Stack Exchange and its associated websites provide a massive platform for sharing technical knowledge and collaboration on projects. Encompassing actively used sites such as *Stack Overflow*, *Super Users* and *Ask Ubuntu*, the Stack Exchange environment is a network for questions and answers across a wide range of topics. These forums serve as hubs for basic as well as complex queries providing a rich collection of technical text data.

Such information often covers important topics and the latest trends in various industries. A way to summarize this text as well as identify topics in the data could help search engines sort user queries and provide quicker and more accurate results. This would not only help the companies hosting the platform optimize their results but also help other similar forums categorize future questions in an automated manner where users can more easily find solutions to their queries. In this project, we propose topic modeling techniques to analyze this text data and identify keywords related to each question. As part of the analysis, different algorithms such as *Latent Dirichlet Allocation (LDA)* as well as *lda2vec* using word embedding are evaluated in order to optimize the categorization.

Stack Exchange as well as similar platforms such as *Quora*, *Reddit*, *LinkedIn* groups, *Facebook* groups etc, could use this model or even adapt the model on their respective platforms to categorize text data. Since text data is platform agnostic, this model and analysis can be extended to other sites and search engines such as Google and Bing. The analysis can also be applied to any sort of text data such as comments, reviews, item descriptions in platforms such as Yelp, Amazon, Twitter etc.

2. Data Description

The dataset is acquired from Kaggle and consists of questions from Stack Exchange users, spanning **420,668 rows**. Each instance contains a column that has the question title and another column containing the body of the question which could consist of text as well as code. The text in the body is in HTML format which was processed to extract only the meaningful text. A snippet of the dataset is shown in *Fig 1*.

	Title	Body
0	How to check if an uploaded file is an image w...	I'd like to check if an uploaded file is an...
1	How can I prevent firefox from closing when I ...	In my favorite editor (vim), I regularly us...
2	R Error Invalid type (list) for variable	I am import matlab file and construct a dat...
3	How do I replace special characters in a URL?	This is probably very simple, but I simply ...
4	How to modify whois contact details?	<pre><code>function modify(.....)\n{\n $mco...

Fig 1. Snippet of dataset showing the *Title* and *Body* columns

There was an additional column which contained the topics related to each document, serving as the target variable. The original kaggle competition was structured to predict these topics/labels. However, for our application, we ignore the target variable and conduct our own analysis on simply the title and the question treating the data as unstructured.

3. Data Cleaning

a. Handling missing values

An initial look at the data using the revealed 420,556 non-null values out of the 420,666 total instances indicating 110 missing/unknown rows. Since this was a relatively amount, we were able to simply remove these instances without affecting the overall quality of the dataset.

b. Removing HTML tags

HTML tags are special characters within the source code of websites that encompass different types of data about the features on the page. All such tags generally come in pairs of complimentary tags that indicate the start and end of sequences. The start tag is embedded within the **<>** notation whereas the end notation is **</>** emphasize by a forward slash. For example, the * * and * * notations indicate a **paragraph** while the *<b>* and *<b/>* tags specify **bold text**.

All the HTML tags in the dataset were identified and extracted using the *re* regular expressions library in Python. The *Body* variable contained a total of **6,231,982 tags** out of which **91 were unique** tags. One thing to note would be that there could be more than one type of tag in an instance. In *Fig 2*, we can see the 25 highest occurring tags in the *Body* column. One interesting thing to note from the plot is that all the tag compliments have the same frequencies as expected. However, there are rare cases where a tag could have a different frequency than its compliment. For example, **<Code>** has 3 occurrences while **</Code>** has 2 occurrences.

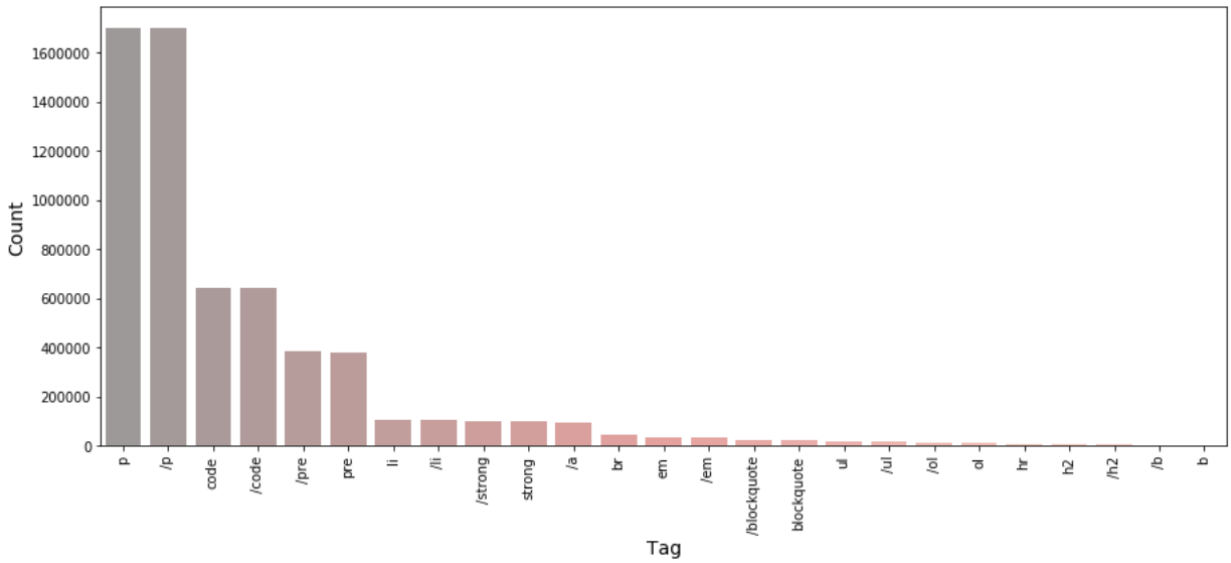

Fig 2. Distribution of top 25 tags in the *Body* column.

To somewhat visualize the occurrence of these tags in each instance, we can plot the distribution for each tag stacked on top of another as shown in *Fig 3*. From this plot, it is evident that the frequency of **<p>** in general is higher. Also, we can see that some tags have unusually high frequencies in certain documents.

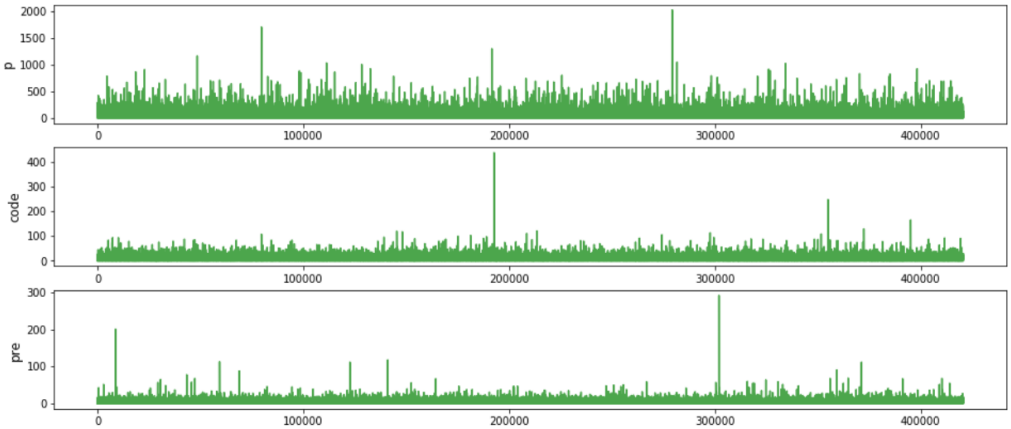

Fig 3. Occurrences of tags across each instance in the *Body* documents.

The distribution of tag frequencies for *Title*, as shown in *Fig 4*, are far more uneven than the *Body* case. There are no complimentary tags and some of these words are not even actual HTML tags such as **script, img,** and **object**. Most of these words could have simply had **<>** and **<>** around them because the user wanted to emphasize them. Considering that, most of these words don't seem to be keywords either. Therefore, we get rid of these words completely.

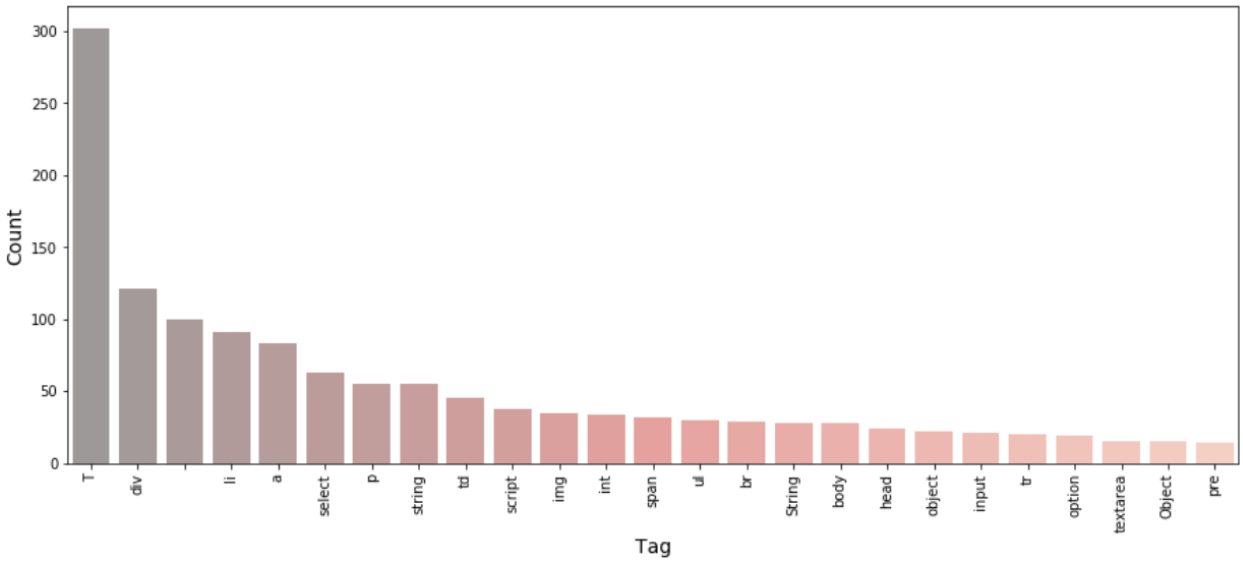

Fig 4. Distribution of top 25 tags in the *Body* column.

c. Removing newline characters

Newline characters (/n) offer no real value for topic modeling, hence they were removed from the dataset.

4. Pre-processing

a. Feature selection

It turns out that tokenizing over 450,000 text documents for both **Title** and **Body** takes up too much computational resources. Further processing can cause the kernel to crash. From some initial analysis done on the frequently occurring words using just 500 instances from the dataset, it was discovered that the title in fact contains more crucial keywords than the body. Therefore, for our analysis, we are simply going to use the *Title* variable.

Fig 5 shows the 50 most frequent words in the title documents while *Fig 6* shows the same for the body. From the title distribution, we can see that words such as **create, jquery, server** and **java** are some of the highest occurring words. These words could serve as crucial keywords. The highest occurring words in the body column are **http, new, like, try** and **work**. Most of the

frequent words in this case are more conversational with a lot of verbs as compared to the words in the title column. This makes sense as the body contains more detailed descriptions while the title tends to contain more keywords.

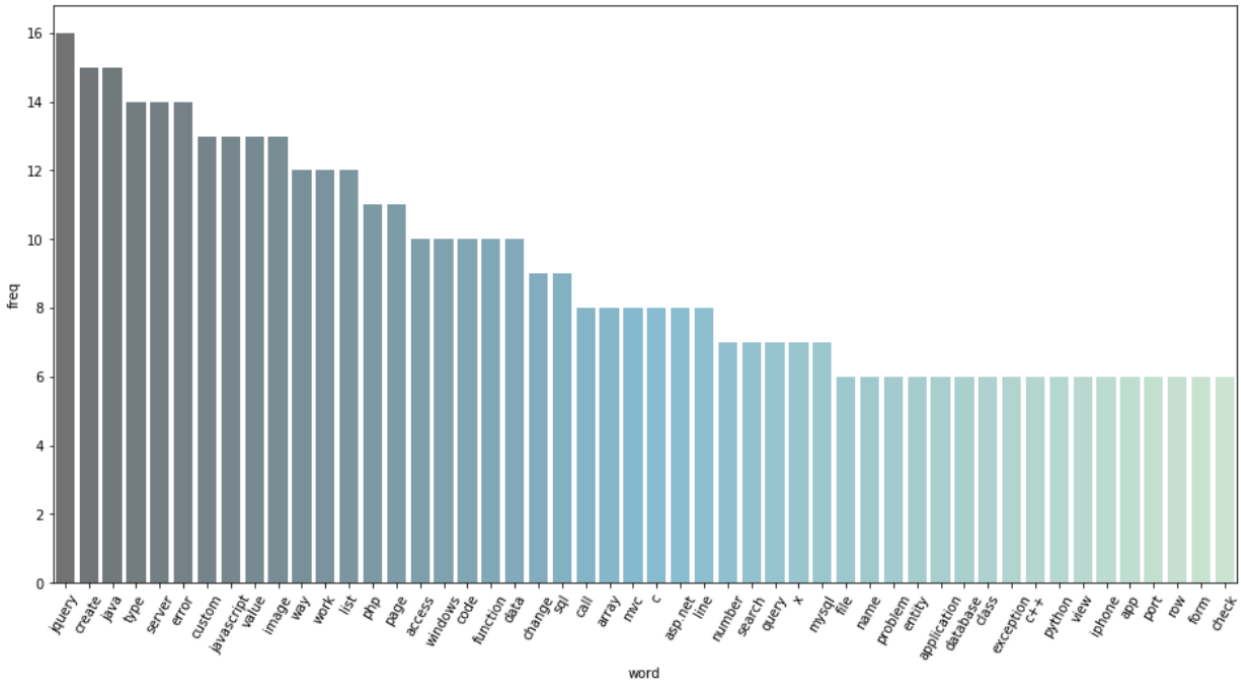

Fig 5. Distribution of top 50 highest occurring words in the title.

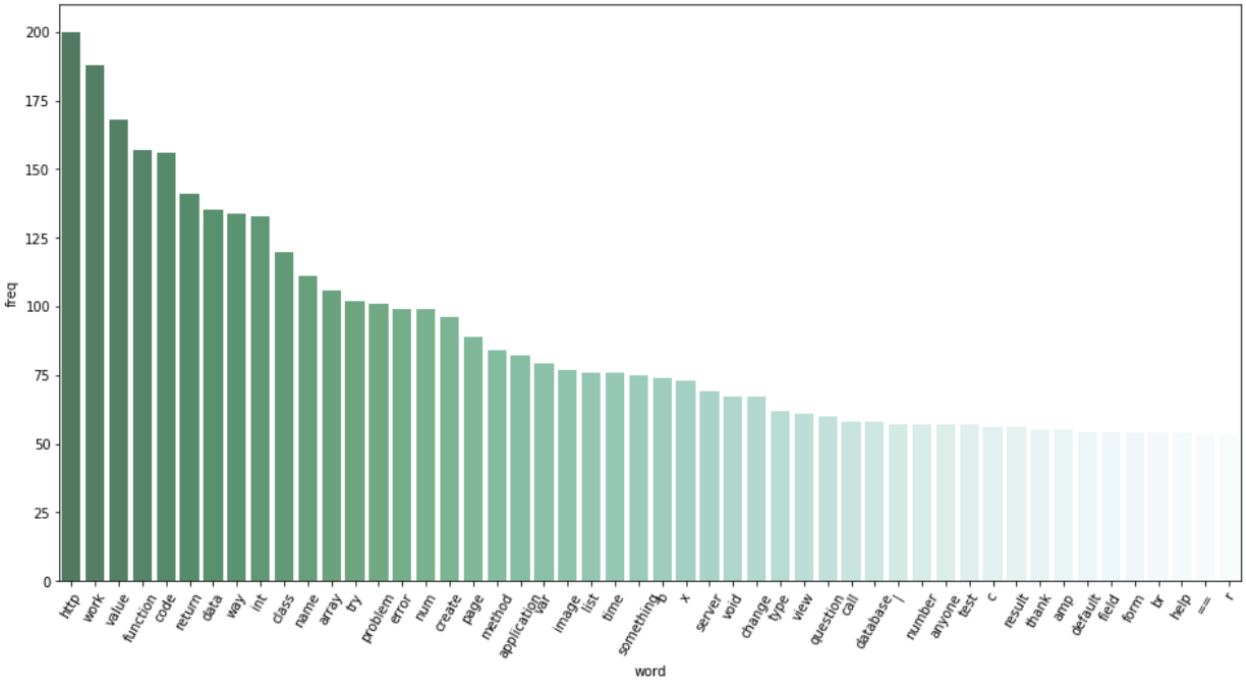

Fig 6. Distribution of top 50 highest occurring words in the title.

b. Word tokenization

Tokenization splits a sentence or text document into tokens which could be words, special characters, punctuations etc. Hence, it's more effective than simply using the ***.split()*** function. This technique is applied to the entire dataset using the *word_tokenize()* function of Python's *nltk* library. A snippet of the tokenized documents is shown in *Fig 7*.

	Title
0	[How, to, check, if, an, uploaded, file, is, a...
1	[How, can, I, prevent, firefox, from, closing,...
2	[R, Error, Invalid, type, (, list,), for, var...
3	[How, do, I, replace, special, characters, in,...
4	[How, to, modify, whois, contact, details, ?]

Fig 7. Snippet of tokenized words for each document in the title.

c. Changing words to lowercase

All words in the dataset are changed to lowercase so that the topic model can recognize two words such as 'Java' and 'java' as the same for practical purposes.

d. Handling stopwords, special characters and symbols

The stopwords were removed in two stages. First we extract a list of all common stopwords using the *stopwords.words('English')* function from the *nltk* library. However, there were still words such as **I, The** and **'d** which also behave as potential stopwords. For these special cases, we iteratively collected as many such strings by checking the highest token frequencies in the dataset.

Tokens such as **(, $, ", +** etc. would show up as some of the highest frequency tokens but offer little in terms of topic modeling. Therefore, such tokens manually added to a list which would then be used to filter out these stopwords from the dataset in a single step. This process was repeated two to three times to get rid of the special characters, symbols and other stopwords that the *nltk* function missed.

e. Lemmatization

Lemmatization was used to reduce different variations of a word to a single word. For example, **runs, ran** and **running** were all changed to the root word **run**. However, we needed to

provide a context for the change. In our case, the context or part of speech was set to a **verb** which allowed all the variations of a word to be changed to the simplest verb form.

The verb context was selected since a lot of the questions are about the user wanting to do something or something that they have already tried. This allowed us to group all the variations of a word together for topic modeling. By default, *lemmatizer* uses the noun form. Later, we get rid of verbs in the dataset as nouns are better keywords for topic modeling. When it comes to more meaningful keywords as such as JQuery or Java, lemmatizer will keep them in the default form as the verbs for these words don't exist. This is another reason why choosing the verb POS for lemmatization is more effective as it would detect all the verbs which would then be filtered out properly.

We have not opted for the *PorterStemmer* method as that tends to reduce words to root words that may not be in the standard English language.

f. Part of Speech (POS) tagging

First we explored all types of POS tags that exist in the dataset for both title and body.

Common POS tags*:*

- *Nouns*: NN
- *Plural nouns*: NNS
- *Proper nouns*: NNP, NNPS
- *Verbs*: VBP, VBN and VBD
- *Adjectives*: JJ, JJR, JJS
- *Adverbs*: RB, RBR (comparative), RBS (superlative)

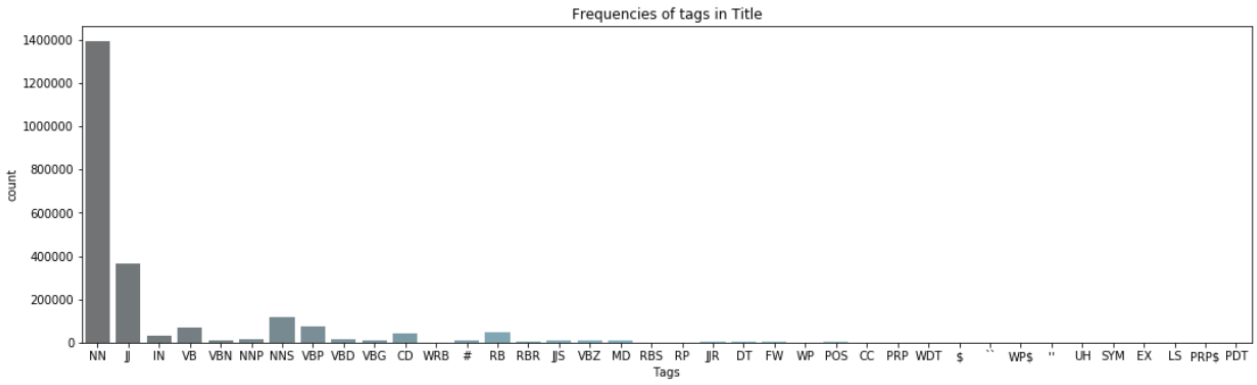

Fig 8. Distribution of POS tags in the title documents.

Fig 8 shows the distribution of the different POS tags in the dataset while *Fig 9* shows the exact frequencies in a table format. From the distributions, we can see that **nouns are by far the most frequent (1,394,514)** with **adjectives coming in at second (367719)**. **Verbs (VB, VBN, VBP, VBD etc) come up to around 19,000 to 20,000** in total while adverbs (RB, RBS, RBR) account for less than 50,000 of the instances.

NN	1394514
JJ	367719
NNS	118973
VBP	76103
VB	70280
RB	47175
CD	44420
IN	29867
NNP	17146
VBD	14201
VBZ	12238
VBG	10940
VBN	10064

Fig 9. Table showing exact POS tag frequencies for the top 13 tags.

The frequencies of the 40 highest occurring verbs are shown in *Fig 10*. We can see that most of the words do not add much value for topic modeling. However, there are certain words that should really be in the **noun** category such as **windows** and **android**. On the other hand, certain words could fall under both verb and noun depending on the meaning and context. Such words are **object** and **string**. The POS tagger is not going to be able to handle these intricacies. Although, there is a good chance that some of these words also show up as nouns so we don't have to delete them.

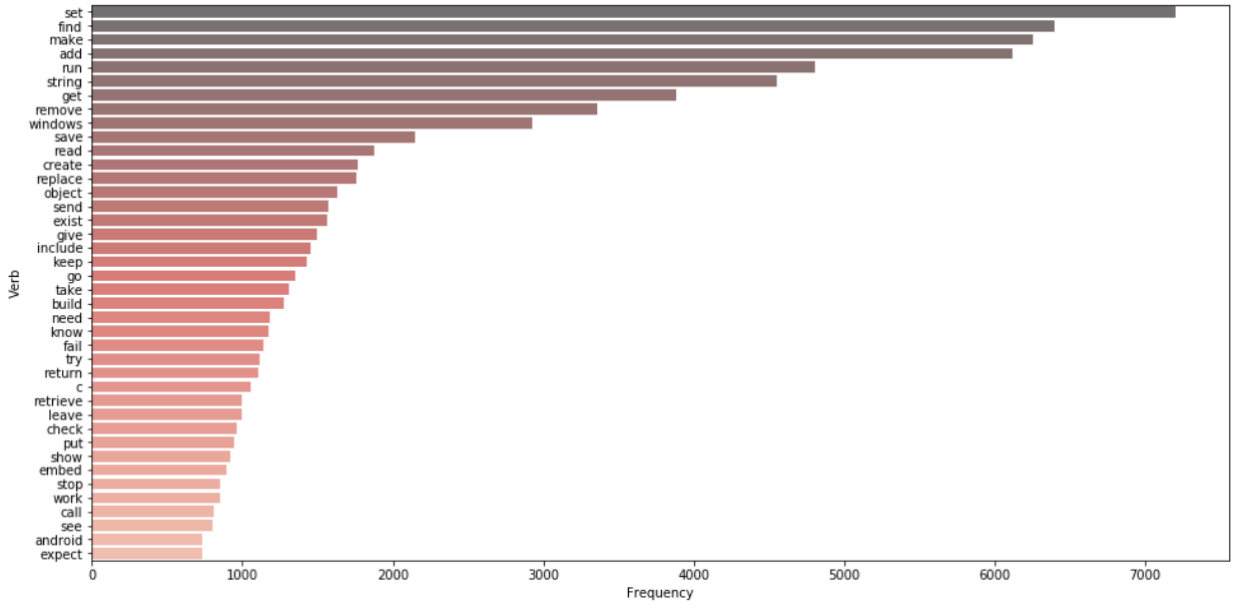

Fig 10. Distribution of 40 highest occurring verbs.

When it comes to adjectives the highest occurring words, shown in *Fig 11*, are accurate to a good extent but there are some keywords such as **android, library, c, url and php** that need to be preserved as they could serve as potential keywords. Overall, the adjectives seem to be more useful for topic modeling than verbs. Hence, we may end up retaining this category.

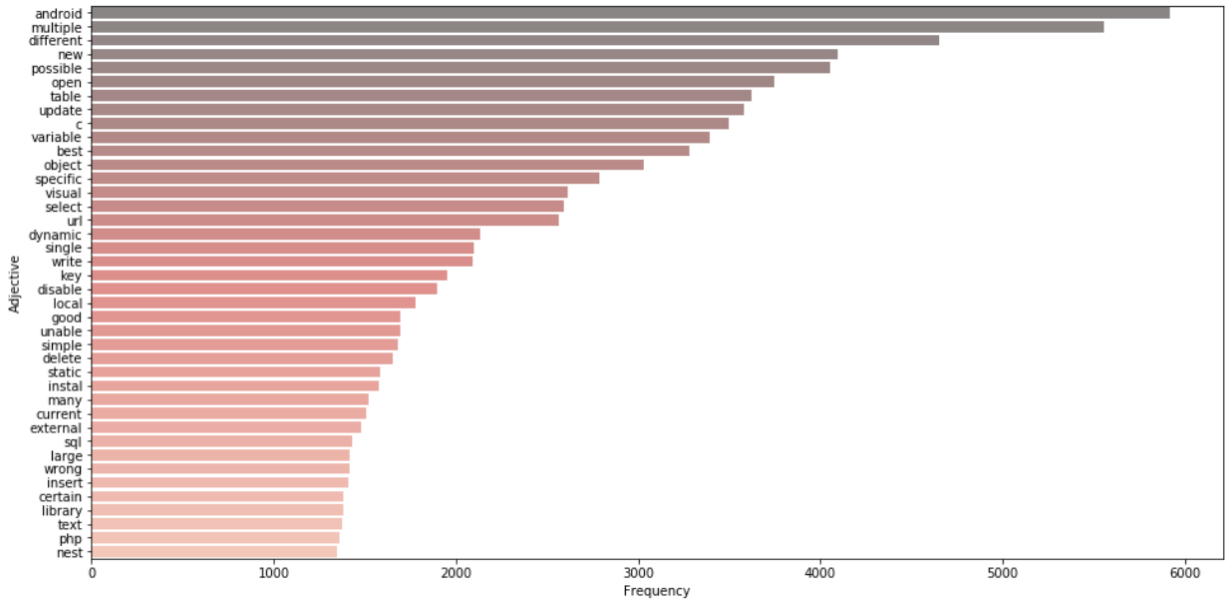

Fig 11. Distribution of 40 highest occurring adjectives.

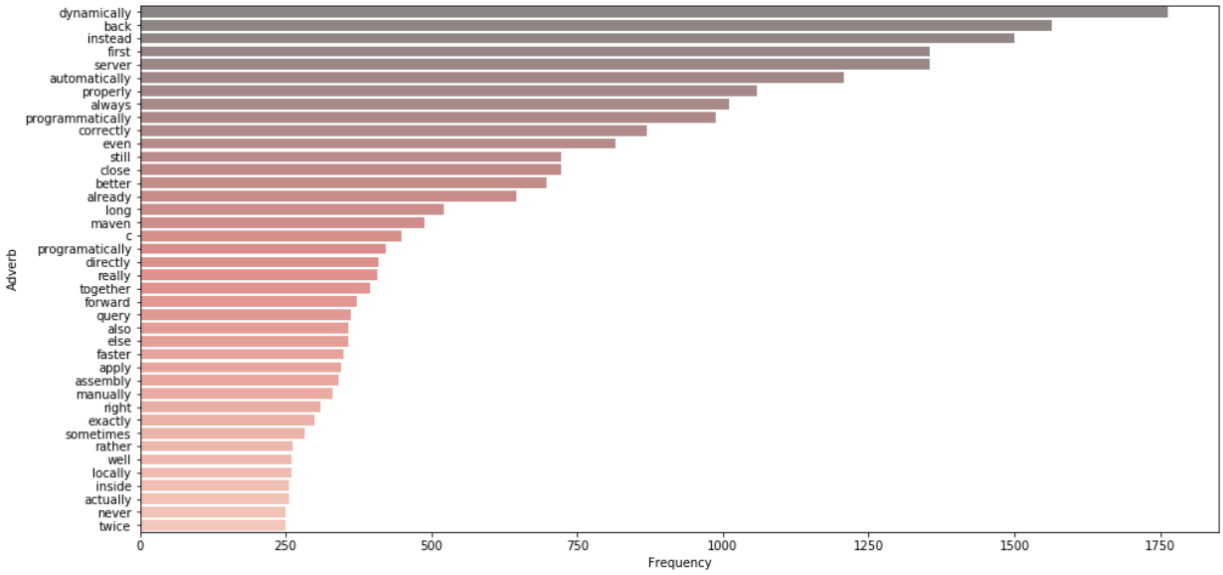

Fig 12. Distribution of 40 highest occurring adverbs.

The distribution of adverbs are shown in *Fig 12*. Since adverbs describe an action (or verb), a lot of these words would be similar to verbs themselves. Hence, similar to verbs, adverbs don't really provide a lot of keywords for our analysis. We can possibly get rid of these.

Fig 13 shows the top 40 nouns in the dataset and we can immediately see that most of these words could be crucial words for topic modeling. Words like **jquery, image, array, java** etc. are technical terms and would provide more meaning to our model than action words.

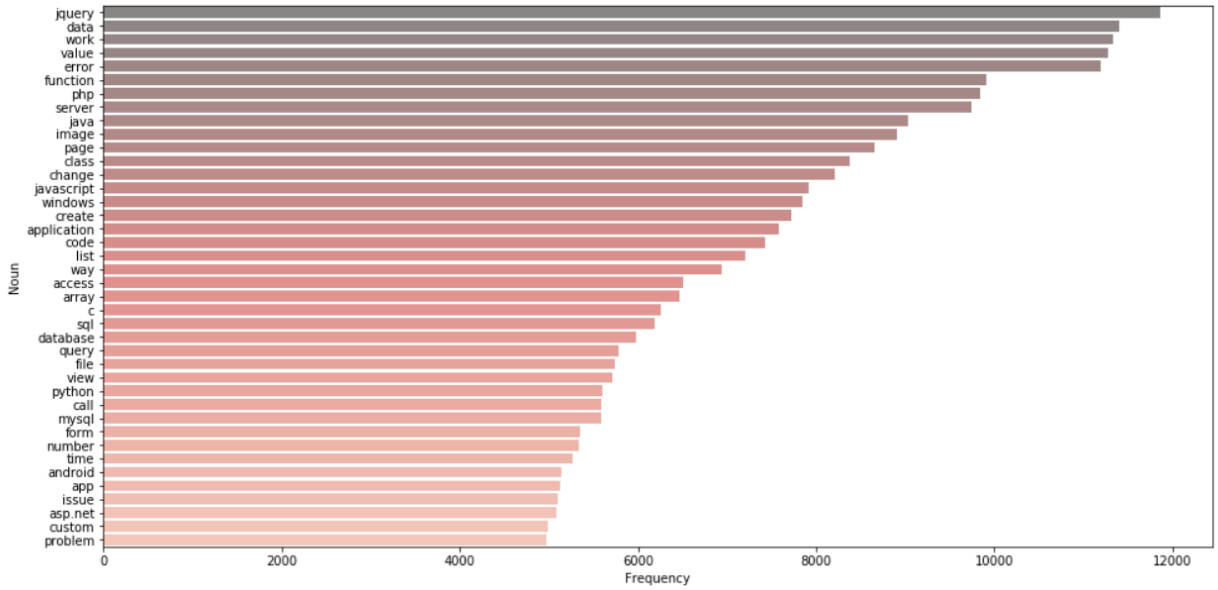

Fig 13. Distribution of 40 highest occurring nouns.

For our analysis, we only kept the nouns and got rid of other tags such as verbs, adverbs and adjectives. We did this by retaining all words with the POS tags *NN, NNS, NNP* and *NNPS*.

5. Exploratory Data Analysis

From the distribution in *Fig 14*, we can see that words such as **create, jquery, server** and **java** are some of the highest occurring words in the title column and they match our earlier visualization that contains just nouns. This makes sense as we only have nouns in our dataset.

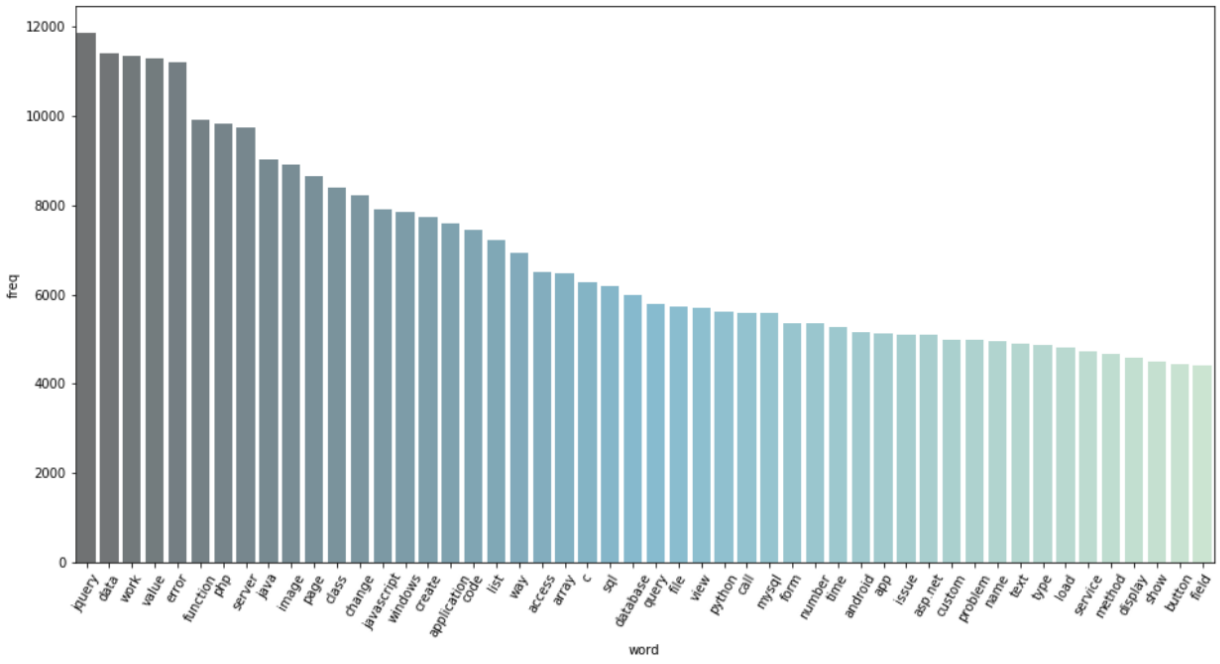

Fig 14. Distribution of highest occurring words in the preprocessed dataset.

6. Topic Modeling Using Latent Dirichlet Allocation (LDA)

To find the best topic model for our data using LDA, two common word representations are compared and evaluated:

- Bag of Words (BOW)
- Term Frequency Inverse Document Frequency (TFIDF)

In this analysis, only the first 100,000 of the 450,000 plus instances were chosen as the computation power as well as time needed to model the entire dataset proved to be substantial making it not feasible for repeated model generation during evaluation and optimization.

a. Preprocessing

The cleaned dataset containing the vocabulary was prepared for topic modeling by first converting the documents from string format to tokenized format. The dataset needed to be converted into a *corpus*. Before that, a *dictionary* of words had to be created which assigns a unique identifier to each word in the dataset, done using the *copora.Dictionary()* function of the *genism* Python library. This serves as a lookup table for all unique words, shown in *Fig 15*.

```
0 check
1 image
2 mime
3 type
4 ctrl-w
5 firefox
6 press
7 prevent
8 error
9 list
10 r
```

Fig 15. Dictionary of words displaying the first 11 entries.

This dictionary of unique words is used to convert the dataset containing the documents to the *bag of words (BOW)* representation using the *doc2bow()* function, whereby providing a corpus of words. The corpus contains IDs representing each token and the frequency of that token in each document. In *Fig 16*, we can see the first 20 instances of the corpus. These 20 instances represent the first 20 documents in the BOW format.

```
[[(0, 1), (1, 1), (2, 1), (3, 1)],
 [(4, 1), (5, 1), (6, 1), (7, 1)],
 [(3, 1), (8, 1), (9, 1), (10, 1)],
 [(11, 1), (12, 1)],
 [(13, 1), (14, 1)],
 [(15, 1), (16, 1)],
 [(17, 1), (18, 1), (19, 1), (20, 1)],
 [(21, 1), (22, 1), (23, 1), (24, 1)],
 [(25, 1), (26, 1)],
 [(27, 1), (28, 1), (29, 1), (30, 1), (31, 1), (32, 1)],
 [(33, 1), (34, 1), (35, 1), (36, 1), (37, 1)],
 [(38, 1), (39, 1), (40, 1)],
 [(41, 1), (42, 1)],
 [(43, 1)],
 [(44, 1), (45, 2)],
 [(46, 1), (47, 1), (48, 1)],
 [(10, 1), (49, 1), (50, 1), (51, 1)],
 [(52, 1), (53, 1), (54, 1)],
 [(26, 1), (47, 1), (55, 1), (56, 1), (57, 1), (58, 1)],
 [(50, 1), (59, 1), (60, 1), (61, 1), (62, 1), (63, 1)]]
```

Fig 16. Corpus or BOW showing the frequency of words in each document.

Each document in the BOW format consists of a list of tuples where each tuple is of size two. The first number in the tuple is the ID of the word in the dictionary while the second number is the frequency of that word in the particular document. Therefore, the same ID can be present in more than one document.

The TFIDF representation, on the other hand, is similar to BOW with the difference being that the word frequencies in each document are not whole numbers but weighted decimal values between the range of 0 to 1, as shown in *Fig 17*.

```
[(0, 0.4324362285170671), (1, 0.3400147007278977), (2, 0.7352512076370518), (3,
[(4, 0.7307555258428255), (5, 0.38586296208285703), (6, 0.42741204360605), (7,
[(3, 0.4910368453884094), (8, 0.39654357171903315), (9, 0.4472752142445557), (1
[(11, 0.6511999331492017), (12, 0.758906217570047)]
[(13, 0.703350259697448), (14, 0.7108434512489599)]
[(15, 0.6496168765377784), (16, 0.7602617402692975)]
[(17, 0.5965129297338804), (18, 0.6390460225330046), (19, 0.4152995160670111),
[(21, 0.3811539407504564), (22, 0.5980675041399135), (23, 0.5576527062334482),
[(25, 0.7956451687365886), (26, 0.605762961451198)]
[(27, 0.2930988459474298), (28, 0.58056770893597), (29, 0.49047127193549894), (
[(33, 0.24956987590763943), (34, 0.28044339911957505), (35, 0.5343913508627418)
```

Fig 17. TFIDF representation of words in the corpus

This representation of the corpus assigns higher values to words that are more unique in the collection of documents. For example, if a word occurs in many documents, it's frequency in TFIDF will be relatively low in each document. On the other hand, if a word occurs very few times, its frequency will be higher in each document.

b. Baseline model

The baseline LDA model was created using the following model parameters:

- *no_below (10)*: Ignores words that occur in less than 10 documents. Since these words are so rare, they would probably not provide much information for the mode. This is equivalent to removing features that contain too few non-null values when it comes to classification problems.
- *no_above (0.5)*: Ignores words that occur in more than 50% of documents. The motivation behind this is that words that occur too often don't provide much useful information either.
- *keep_n (1000)*: Only considers the top 1000 most frequent words in the vocabulary for modeling.
- *num_topics (20)*: Specifies the number of topics to generate.
- *passes (10)*: The number of iterations of LDA to execute. With 10 passes, the algorithm will create word distributions for each topic by going through all documents 10 times.
- *corpus*: TFIDF

In order to come up with suitable model optimization strategies, the results of the baseline model was explored to build an initial understanding of how LDA treats our data.

Below are some of the key metrics that were taken into consideration while evaluating the baseline model as well as subsequent models:

- Topic interpretability
- Word distribution in each topic
- Word distribution in other topics
- Overall word distribution
- Coherence scores (*c_v* and *c_umass*)

As most of these metrics are subjective and need to be evaluated manually, the performance of the LDA model largely depends on the distribution of words within each model and if they provide enough context to the model. Regardless of measurable metrics such as coherence, the evaluation eventually comes down to the manual interpretation of each topic.

The topics in the baseline model were evaluated by visualizing them using the *pyLDAvis* library which displays many of the aforementioned metrics all at once as shown in *Fig 18*.

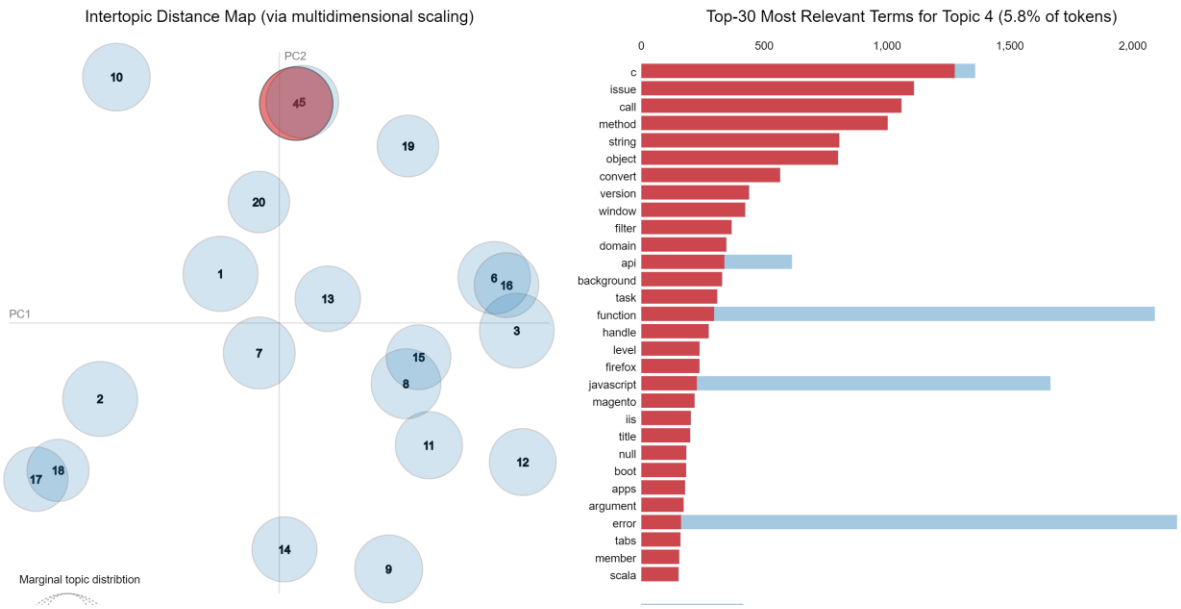

Fig 18. pyLDAvis visualization showing word distributions for topic 4.

In order to analyze the *pyLDAvis* visualization for our baseline model, the following chart characteristics had to be taken into consideration:

- The circles represent the different topics where the proximity of each circle from another determines how similar the two topics are. The sizes of the circles indicate the importance of the topics. The number on the circles follows this notation as well

where **topic 1** has the largest circle. Despite the sizes, the credibility of each topic is still mostly dependent on human judgement.

- The TFIDF corpus (multi-dimensional sparse matrix) has been reduced to a two-dimensional space using dimensionality reduction. In this case, *Principal Coordinate Analysis (PCoA)* is used to do the reduction. Unfortunately, the results of dimensionality reduction are not too interpretable in terms of concrete numbers.
- The bar plots on the right show the distribution of words in the corpus (blue) as well as in the selected topics (red).

Topic 4

We can see from the graph in *Fig 19* is that the term *c* in **topic 4** is denoted as the most important term. This somewhat makes sense as we can see from the distribution plots on the right that this term's frequency is the highest in the topic. Also, it interesting to note that it's distribution in the corpus is only slightly higher than its frequency within the topic. This means that this term is highly unique to this particular topic and hence could define the overall nature of the topic.

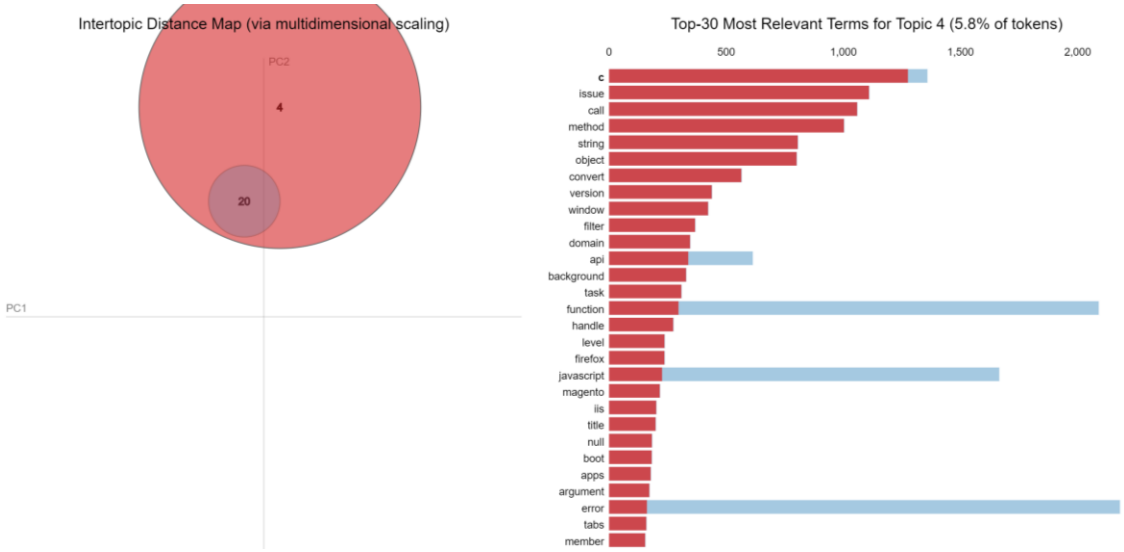

Fig 19. pyLDAvis plot showing the distribution of the word *c* in different topics.

As we can see from *Fig 19*, the only other topic in which this term exists is 20. For the other top words in this topic such as **issue, call, method and string**, all the occurrences of these words in the corpus is only within this topic. Hence, these words are completely unique to this topic. No other topics contain these words. This can potentially help in isolating one topic from another.

Topic 1

Key words in this topic include **image, content, css, page, link, access, load etc**, as shown in *Fig 20*. This would indicate that this topic has to do with things like **web page formatting, blogging, web page templates** and things of that nature.

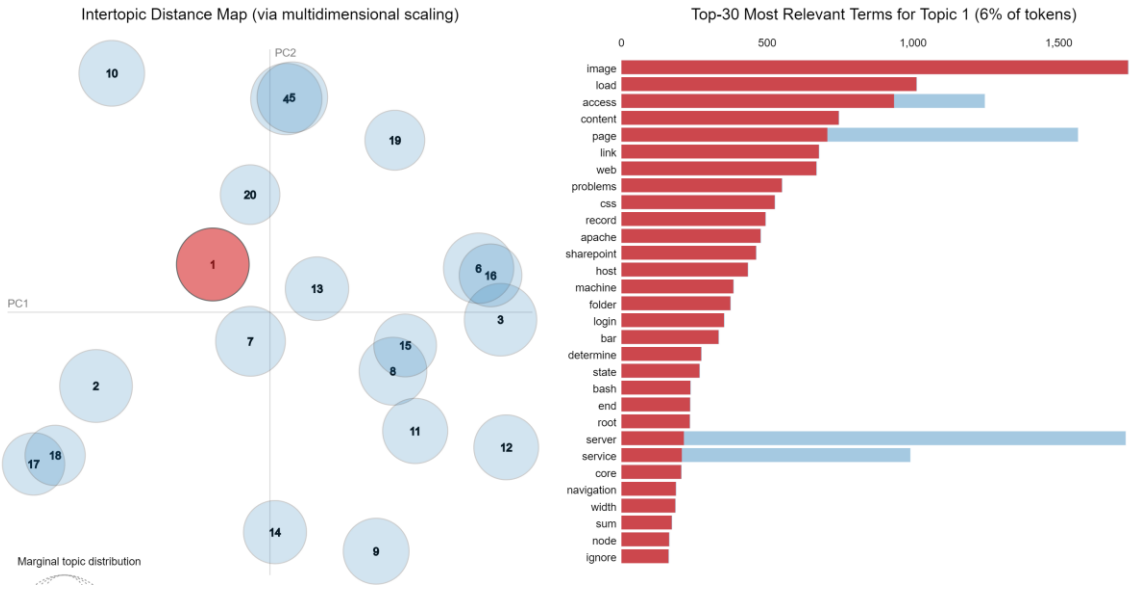

Fig 20. pyLDA visualization for topic 1.

Topic 2

This topic is slightly trickier to interpret. However, we do see some common themes appear if we combine some high-level topics such as **databases, servers, web services, web pages and frameworks** together. The pyLDAvis graph can be seen in *Fig 21.*

For example, *ajax* provides client-side web technologies allowing web applications to retrieve data from online servers. On the other hand, *rails* is a server-side web application framework that provides structures for databases, web pages and web applications. When we compare these two terms, the similarity becomes apparent. Other terms that makes sense within this topic's context are **server, sql, system, file, type, generate, reference etc.**

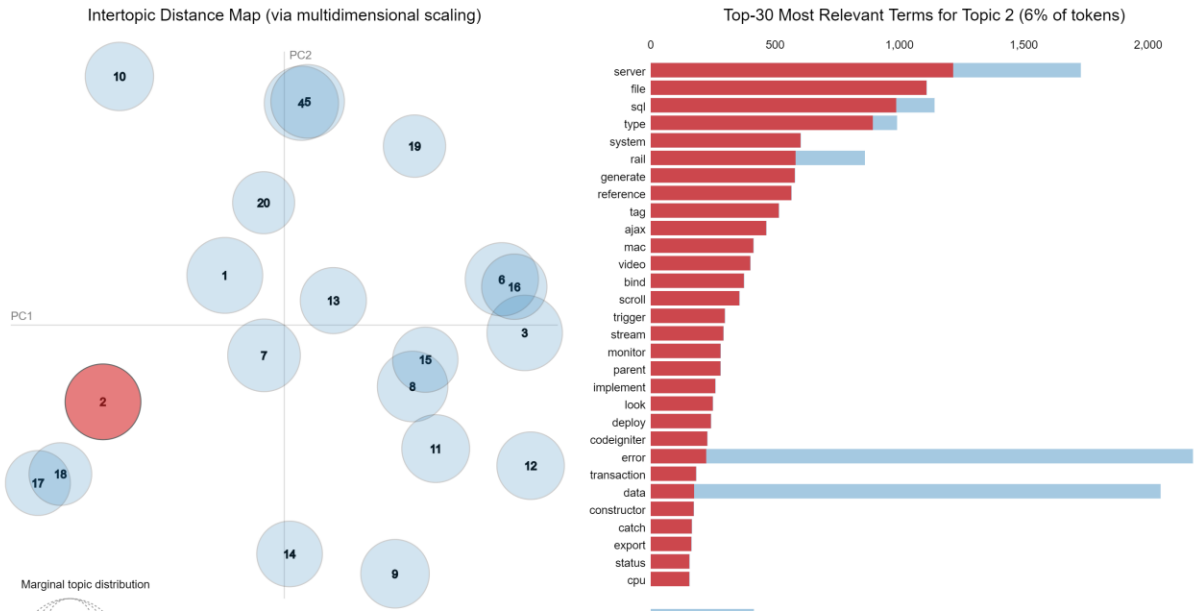

Fig 21. pyLDAvis visualization for topic 2.

Topic 3

This topic seems to cover themes such as **programming, data types and API**.

Topic 5

The themes appear to be **web applications, user interface** and **web development.**

Based on these results, the baseline model provided fairly interpretable topics. We have a **C_v coherence score of 0.4289** and a **C_umass score of -11.9389**. Just by themselves, these numbers don't provide much meaning. More models will need to be evaluated to determine if these results are good or not.

c. Model Optimization

i. Parameter tuning

For the optimization process, the number of passes was kept to 10 in order to quicken the model training. The goal was to change this value to 50 or 100 once the model was fine tuned in order to get more consistent results. Other parameters such as *no_below* and *no_above* are also kept the same at 10 and 0.5 respectively as these seemed to provide decent results.

Below is a summary of how the various parameters were used during the optimization process:

- **Parameters kept constant:**
    - *n_below*: 10
    - *n_above*: 0.5
    - *passes*: 10
- **Tuned parameters:**
    - TFIDF vs BOW

TFIDF appears to provide slightly clearer topics. Much of this interpretation is subjective and depends on manual analysis of word distributions in each topic.

- Number of topics (*num_topics*)

The baseline model is first optimized by varying the number of topics in the LDA model for both TFIDF and BOW representations. Unfortunately, evaluation for these parameters needed to be manual as it was discovered that the number of topics strongly affects the coherence scores regardless of the practical contextual power of the topics for each model.

About 20 to 50 topics seemed to be the optimal number of topics. As the number of topics increases beyond 50, the number of important words within each topic became decreased and after a point, the topic was not interpretable anymore. On the other hand, if the number of topics was less than 20, the topics become too general and would consist of many topics.

- Number of top most common words to keep in the model *(keep_n)*

In order to further optimize the model, the number of most frequent words to include in the model was also changed which is in fact an argument to the dictionary and not specifically for the LDA model. When this number was increased from 1000 to 10000, there were two noticeable changes:

1. The topics on the PoCA graph were more evenly spread out. Before, a lot of the topics would be close to each other or concentrated in one area. Now, due to the higher number of words being considered, the semantic separation between topics appeared to be clearer.
2. Secondly, the number of important words within each topic increased, which was mostly due to the higher number of total words in the model. As a result, the word distribution bar chart was now less right skewed and had a more even distribution. Due to this, there were more words available to describe the different topics.

The differences between *keep_n* values of 1000 and 10000 is captured by the plots in *Fig 22 and Fig 23* respectively. In the first graph with, the topics can be seen to be very closely spaced from one another in the two-dimensional PoCA space. However, the second graph shows the topics more spread out.

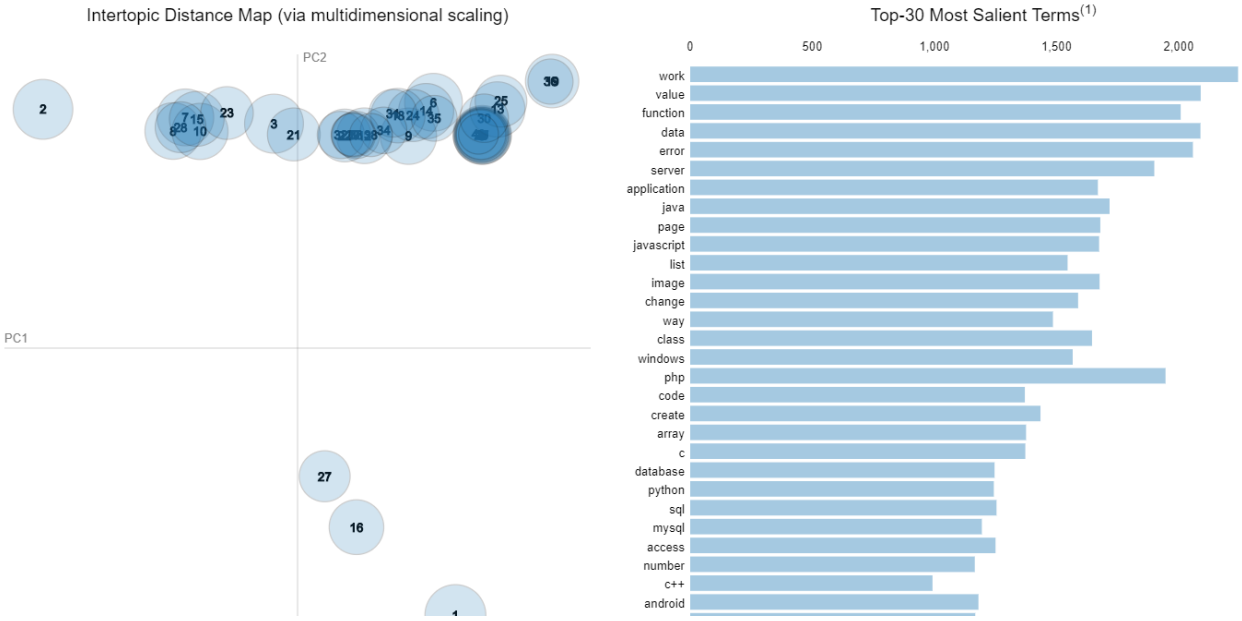

Fig 22. 40 topic model with keep_n value of 1000

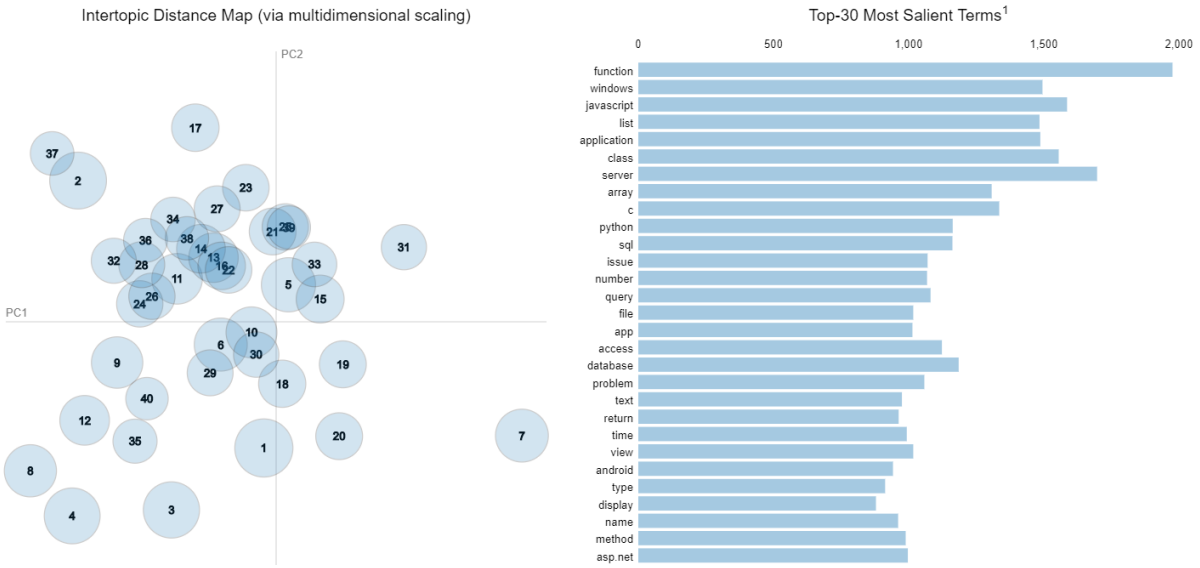

Fig 23. 40 topic model with keep_n value of 10000

ii. Evaluation metrics

Coherence score is one of the metrics to evaluate the topics. It measures the relative distance between words in a topic where the overall coherence score of a topic is the average of the distances between words. There are two major types c_v, typically between 0 and 1 and umass between -14 and 14. The plot in *Fig 24* shows that the c_v coherence score increases, almost in linear fashion, with increasing number of topics. Also, there is a slight indication of a flattening of the curve as we reach 150 topics.

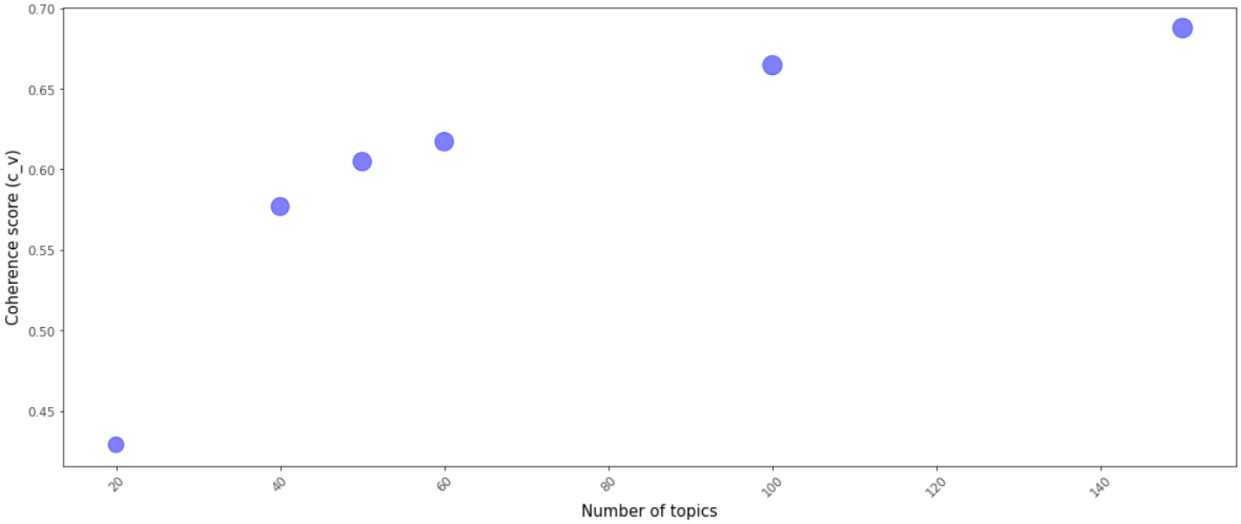

Fig 24. Plot showing the trend of c_v coherence score against number of topics.

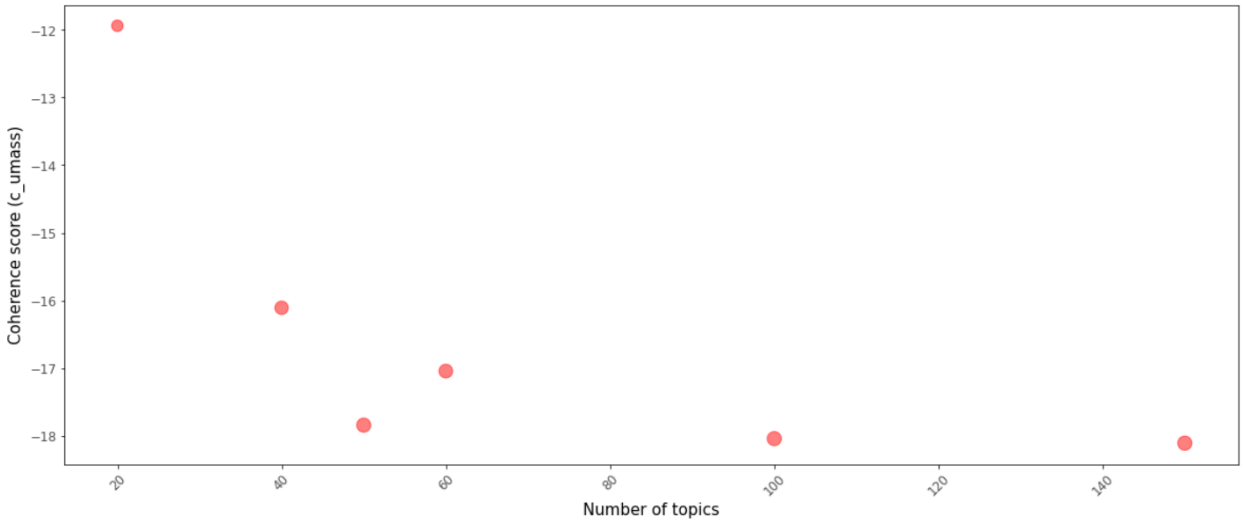

Fig 25. Plot showing the trend of c_umass coherence score against number of topics.

In *Fig 25* we can see the relation between the umass score and the number topics. In contrary to the c_v score, the umass score actually decreases with increasing number of topics. However, it is to be noted that the rate of decrease flattens beyond a 100 topics.

num_topics	keep_n	c_v	c_umass
20	1000	0.4289	-11.9389
20	10000	0.4431	-17.8066
30	10000	0.5192	-17.9372
40	1000	0.5769	-16.1082
40	10000	0.5192	-17.9372
50	1000	0.6048	-17.8427
50	10000	0.5919	-18.1935
60	1000	0.6172	-17.0439
100	1000	0.6647	-18.0425
150	1000	0.6878	-18.1091

Fig 26. Table showing the distribution of coherence scores across number of topics and keep_n

After evaluating 10 models as shown in *Fig 26*, the best model appeared to be model the one which uses TFIDF, 40 topics and a *keep_n* value of 10,000. As discussed earlier, the following metrics were used to pick this model:

- *Topic interpretability*: The five most important topics were used for evaluating each model. For our selected model, the topics were interpretable to a good extent with each topic actually comprising more than one theme.
- *Word distribution in each topic*: A keep_n value of 10,000 with 40 topics provided a less skewed word distribution for each topic.
- *Word distribution in other topics*: Mostly words that were unique to each topic were considered when defining each topic.
- *Overall word distribution*: This relates to the previous point.
- *Coherence scores (c_v and c_umass)*: The coherence scores were not the most optimal for the selected model but as discussed earlier, topic interpretability in this case is more important.

Also, for each model, only the top five topics are evaluated to make the process quicker. Any further optimization was done using this model.

i. Alpha and Beta hyperparameters

Looking at the C_v scores in *Fig 27*, we can see that the best score is **0.6107** corresponding to an **alpha value of 0.1** and **beta value of also 0.1**. Overall, the plot shows a general downward trend as beta is increased. The case is similar for alpha values but the changes are not as significant. *Fig 28* contains a table showing more detailed results.

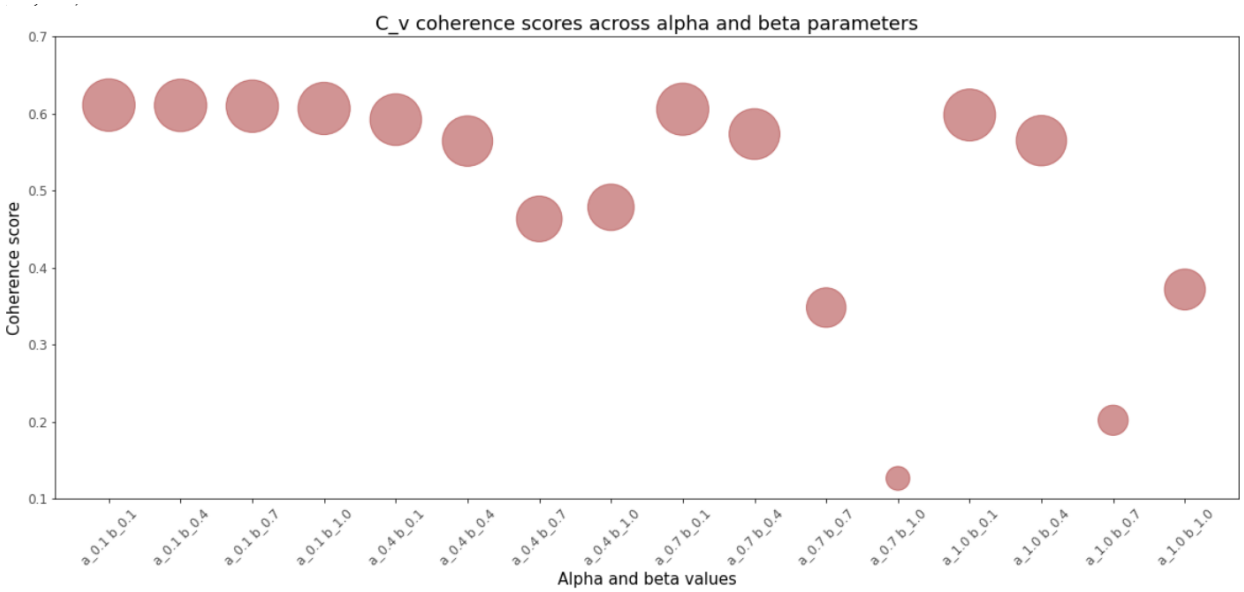

Fig 27. Plot showing the trend of c_v coherence score against alpha and beta hyperparameters.

	alpha	beta	cv	umass
0	0.1	0.1	0.610757	-17.142843
1	0.1	0.4	0.610381	-17.133532
2	0.1	0.7	0.609356	-17.092674
3	0.1	1.0	0.606382	-17.024916
8	0.7	0.1	0.605378	-17.118142
12	1.0	0.1	0.597923	-16.540367
4	0.4	0.1	0.591901	-16.920964
9	0.7	0.4	0.573125	-16.232641

Fig 28. Table showing coherence scores across alpha and beta hyperparameters.

For C_umass, the general trend for the alpha parameter is slightly ascending as shown in *Fig 29*. However, the trend for beta is clearly upward trending with values such as 0.7 and 1.0 showing significantly higher values. C_umass scores closer to 0 indicate a higher cohesion

between words and other topics. Our analysis has suggested that as the number of topics goes up, the C_umass score decreases. This means that as the number of topics increase, each word becomes more less semantically related with other topics. Since a lower score means lesser word correlation to other topics, it seems like the more desirable outcome.

Provided these findings, a higher C_v score and lower C_umass score seems like the best combination. Therefore, we can pick a value of 0.1 for both alpha and beta parameters.

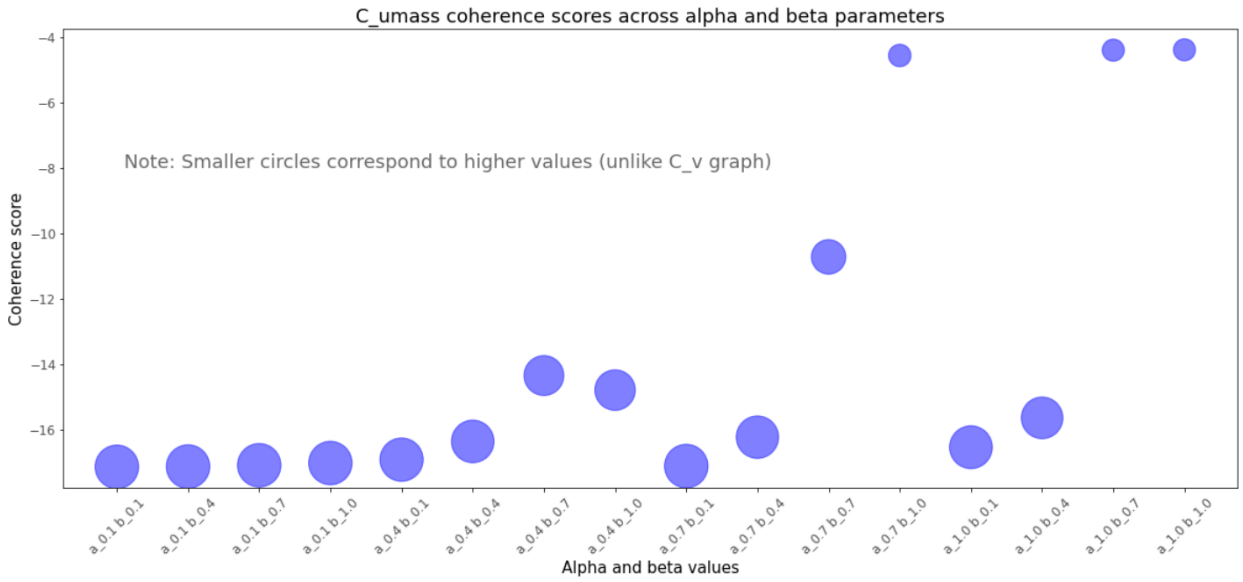

Fig 29. Plot showing the trend of c_v coherence score against alpha and beta hyperparameters.

Why settle for a C_v score of 0.6107 with only 40 topics even though model with 50,100 and 150 topics yielded higher scores?

Higher number of topics means lesser important words per topic. For these models, the interpretability of each topic was getting increasingly harder as there were very few words in each topic to form a general meaningful theme.

Using an alpha value of 50/T

A common practice for LDA is to use a value of 50/T where T is the number of topics. In our case, the number of topics is 40. Hence, another model was fitted with an alpha parameter of 50/40 or 1.25 while keeping beta at 0.1.

An alpha value of 1.25 yielded a better separation between topics, hence this would be a more ideal model. Also, during hyperparameter optimization we saw that as alpha increased from 0 to 1, the coherence score was still around the 0.6 range and did not deviate much. However, the

PoCA plot showed that as we increase the alpha score, the topics are more evenly spread out in 2-dimensional space.

ii. Optimizing number of passes

For this step, we compared the hyperparameter optimized models across 10 and 50 passes. From the visualization of the 50 passes model in *Fig 30*, we can straight away see that a lot of the topics are now clustered together in the top-right quadrant. Our initial impression with the 10 passes model was that the topics would have been spaced out evenly across the two-dimensional plane.

However, despite the topics being largely clustered in the PoCA plane, the model is actually better tuned if we examine the topic distributions on the right. For example, if we look at **topic #3** for this model in *Fig 30*, we can see that all the word distributions within the topic are close to the overall word distribution in the dataset. This indicates that this **topic is completely unique**.

In slight contrast, the corresponding topic (#6) for the model with 10 passes in *Fig 31* shows that distributions for words like **jquery** and **array** are significantly less than their overall distributions. Therefore, these words are not completely unique to this topic. The other topics follow the same trend. Hence, the 100 passes model's topic #3 gets rid of the **array** term completely while making the **jquery** term the most important word in the topic. Hence, through this tradeoff the model was able to more accurately assemble this topic.

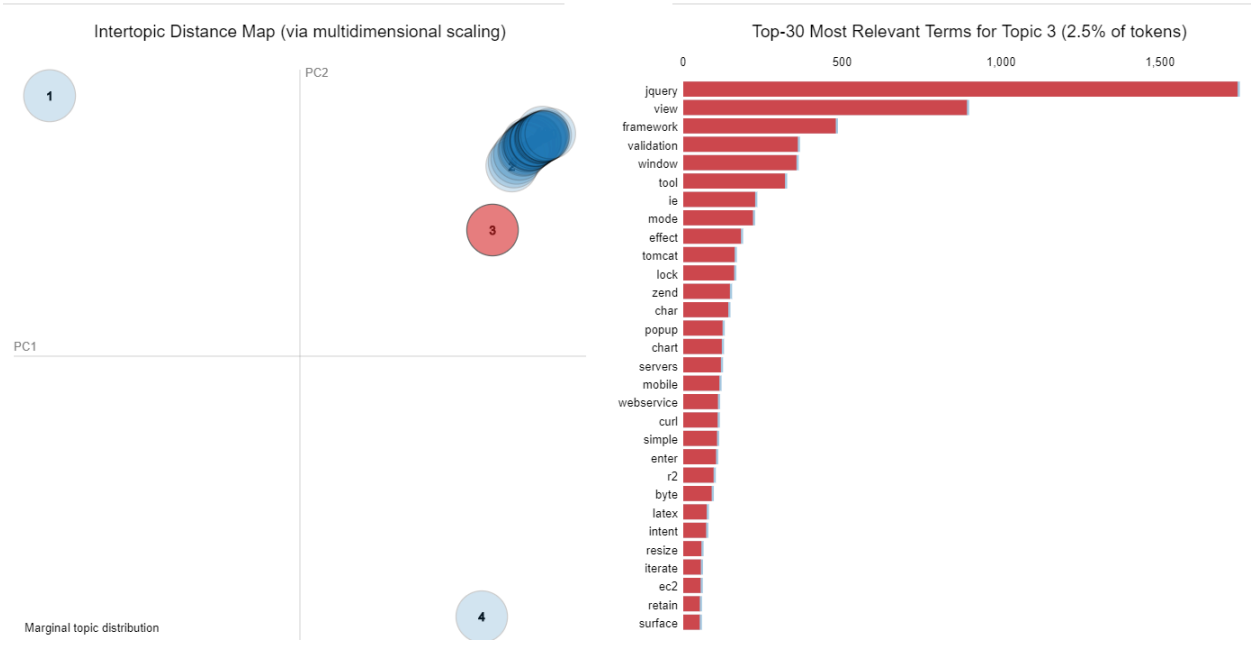

Fig 30. PyLDAVis visualization for optimized model with 50 passes.

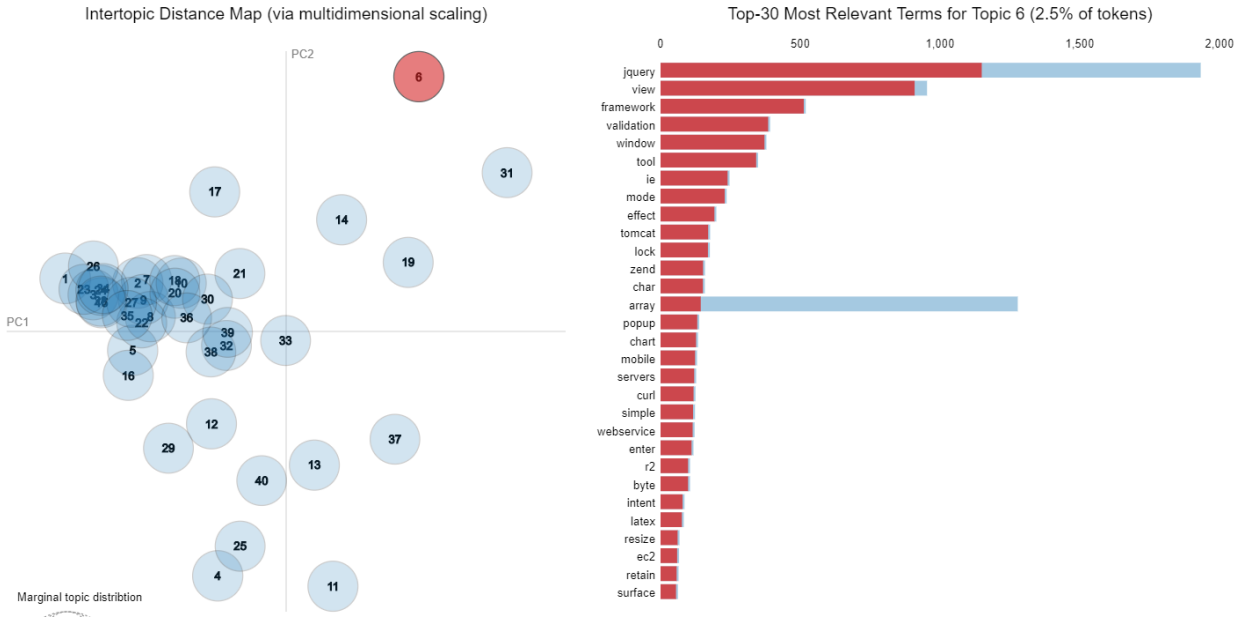

Fig 31. PyLDAVis visualization for optimized model with 10 passes.

d. Topic generation

Each of the 40 topics were manually assigned based on the word distributions for each topic. Most of the topics contained more than one keyword to describe the prevalent themes contained within the topic. Word distributions for **topic 3 and topic 13** are shown in *Fig 32* as examples. Topic 3 was assigned keywords such as **data structures, websites and syntax** while topic 6 was assigned the themes **web design, web layout and internet**. The PyLDAVis graphs provided a more extensive list of word distributions which were mainly used to determine the keywords in the topics.

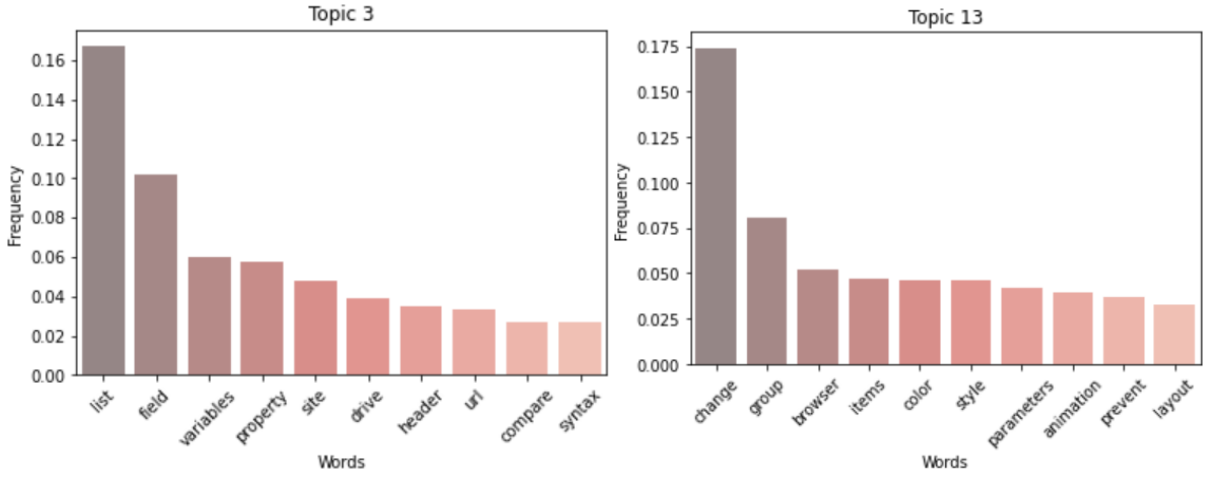

Fig 32. Word distributions for topic 3 and 13.

7. Predicting documents used for training

Shown below (*Fig 33*) is a snippet of the topics predicted for the documents used to train the model. The following important columns are displayed in the table:

- *Document*: This is the variable containing the documents to be predicted.
- *Dominant topic:* Most relevant topic for the document.
- *Topic distribution*: Distribution of the dominant topic within the document.
- *Keywords*: Main themes describing the topic.

	documents	top_topic	topic_dist	keywords
0	[check, image, mime, type]	15	0.041667	[functions, general programming, memory, space, web frameworks, c, object oriented programming]
1	[prevent, firefox, press, ctrl-w]	32	0.041667	[error, store, loop, directory, web layout, web design, internet, software developement]
2	[r, error, type, list]	3	0.041667	[data structures, website, syntax, functions, general programming, memory, space, blogs, posts, wordpress, domain]
3	[character, url]	3	0.043269	[data structures, website, syntax, web frameworks, databases, functions, general programming, memory, space]
4	[contact, detail]	32	0.043269	[error, store, loop, directory, functions, scripting, cloud platforms, programming, scripting, databases]
5	[directory, environment]	20	0.043269	[query, databases, html, web developement, web page, internet, api, website, url, authentication]
6	[draw, barplot, way, coreplot]	11	0.041667	[servers, authentication, web frameworks, databases, data, information, security]
7	[fetch, xml, fee, asp.net]	31	0.041667	[javascript, oracle, event, functions, scripting, data format, login shells, visuals]
8	[generate, javascript]	3	0.043269	[data structures, website, syntax, page, c++, cloud platforms, programming, scripting, databases]
9	[sql, server, procedure, call, inline, concatenation]	9	0.058036	[database servers, web layout developement, blogs, posts, wordpress, domain, apps, django, web framework]

Fig 33. Table showing information about the dominant topic for each predicted document.

From the predictions we can make a couple of observations:

- The predictions are fairly specific due to the respective keywords covering a broad set of categories which could indicate some overfitting. Another reason is that the keywords are from the top three topics predicted for the documents instead of a single category.
- In most cases, the keywords describe the respective documents. However, as mentioned above, some of the additional keywords might lead to overfitting.

For the document **['r','error','type','list']**, the keyword **data structures** describes **list** while **syntax and programming** describes **error, type and r**. However, the additional keywords such as space, blogs and post do not represent any of the document words.

Fig 34 shows the overall frequency of each topic where it can be noticed that some of the dominant keywords are related to programming, scripting and web development.

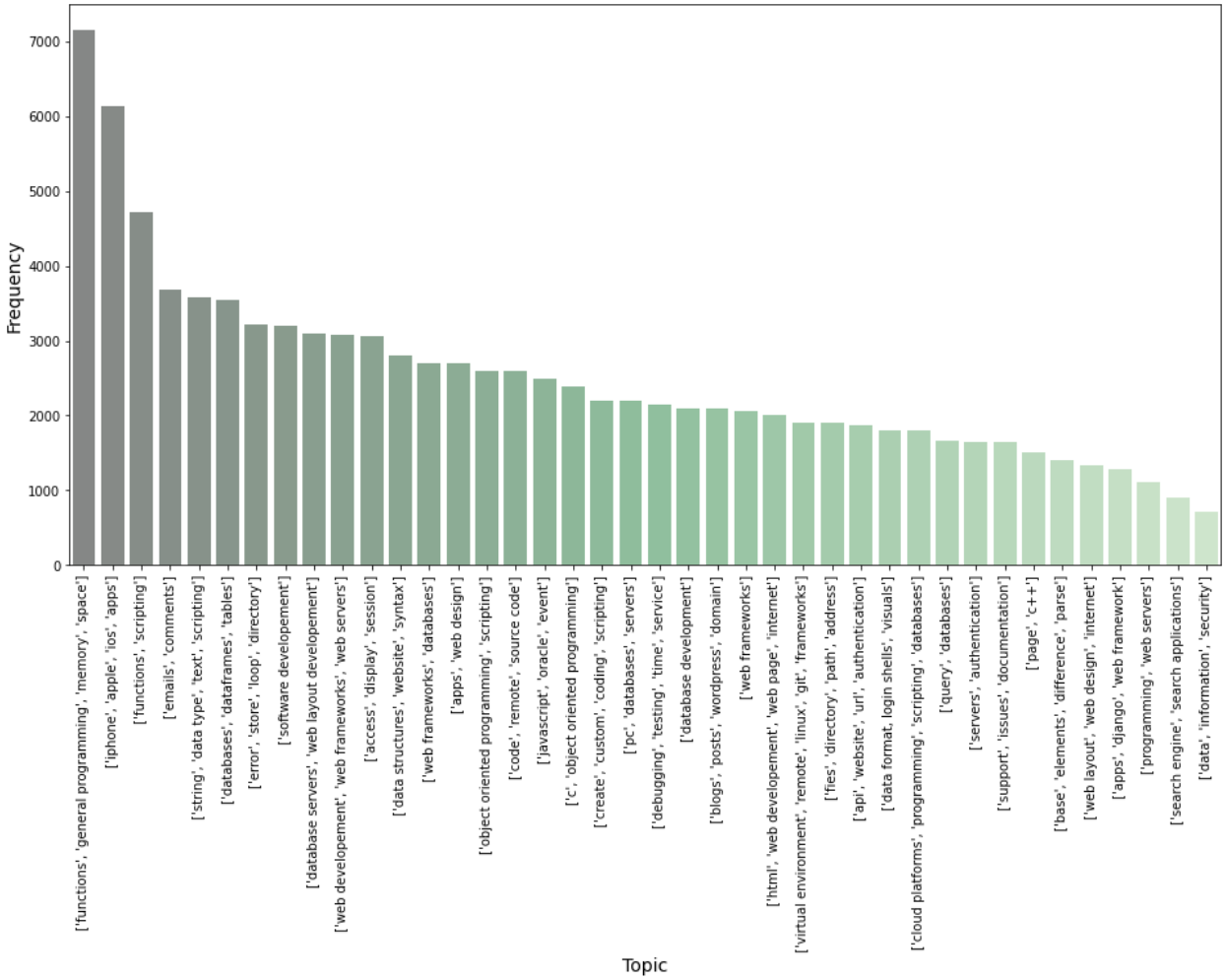

Fig 34. Distribution of topics following the prediction of the documents used for training.

In order to further explore the keywords, the frequency of each individual keyword across all topics was determined, shown in *Fig 35*. As expected, keywords such as **scripting, programming, databases and web frameworks** are the top themes. Fig 36, on the other hand, shows the least frequent keywords such as **string, text, ios, iphone, apple, database developement, html, web page, search applications** and **search engine**. Overall these keywords are mainly related to **text, iphone, html and search engines**. Hence, we can safely say that these are some of the least queried topics on Stack Exchange.

Some of these actually make sense. For example:

- People generally don't go to Stack Exchange for debug or troubleshooting questions with regards to the iphone. Other places like *CNET* or *Gear Patrol* might be more useful since it's a consumer product.
- Questions regarding text or string data are not as prevalent as topics such as syntax, numbers or data structures.

- While HTML is the foundation of web page development, there are other languages such as *JavaScript*, *CSS* and *PHP* that have become more popular over time.
- Not many people need help with search engines unless they are creating or simulating one on their own. Even then, creating a search engine is a very niche technical area.

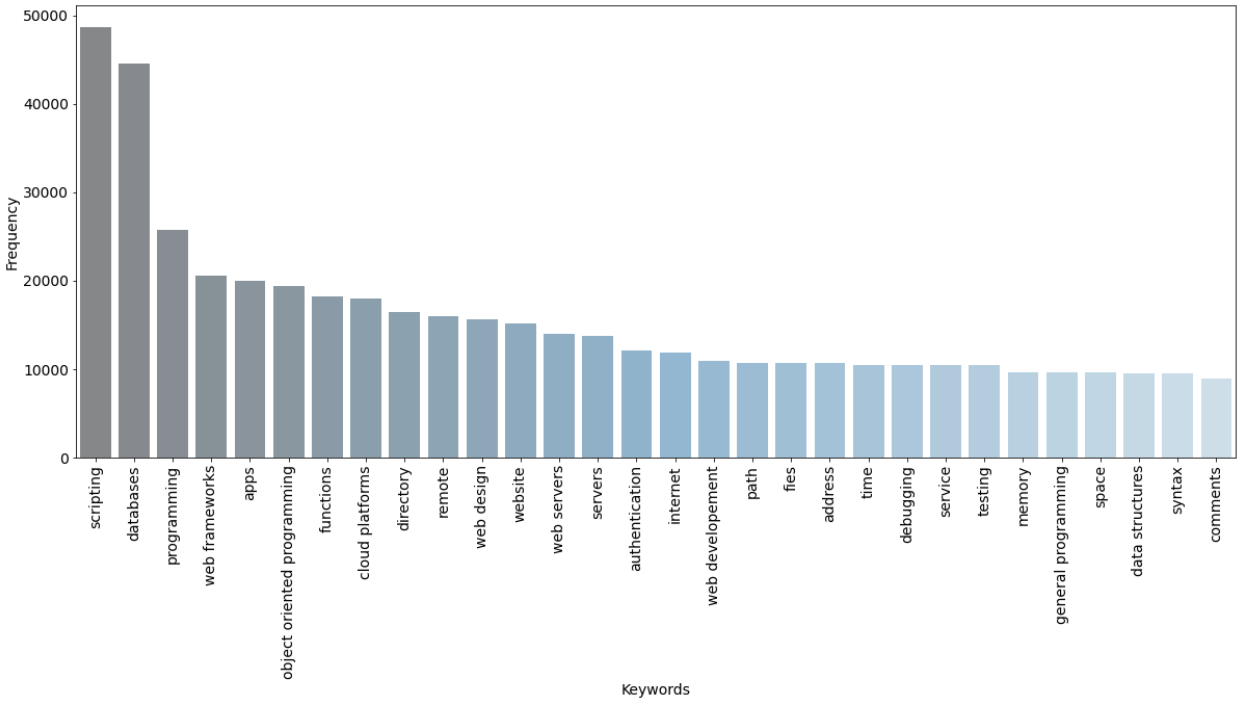

Fig 35. Distribution of top keywords following prediction of training documents.

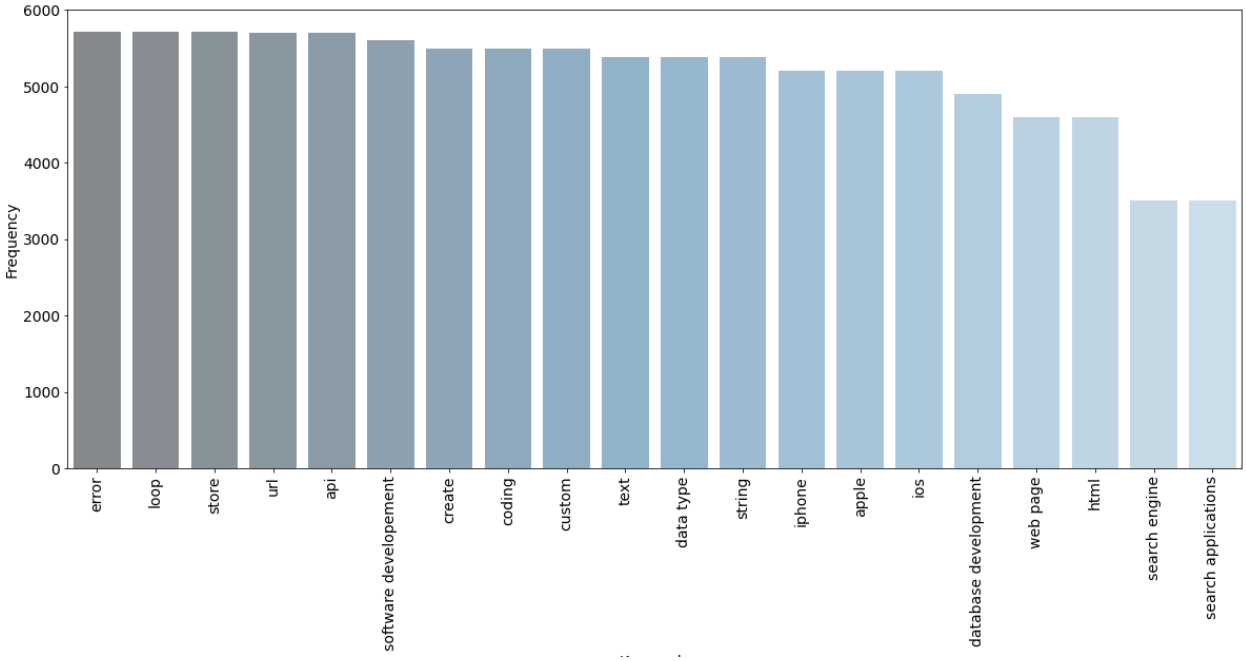

Fig 36. Distribution of lowest occurring keywords following prediction of training documents.

Fig 37 shows the distribution of documents across each topic. This basically tells us what percentage of documents each topic appeared in. Topics 24, 2, 3 and 15 are the highest occurring topics. *Fig 38* shows the table sorted in descending order of document distribution which also shows the keywords contained within the aforementioned topics.

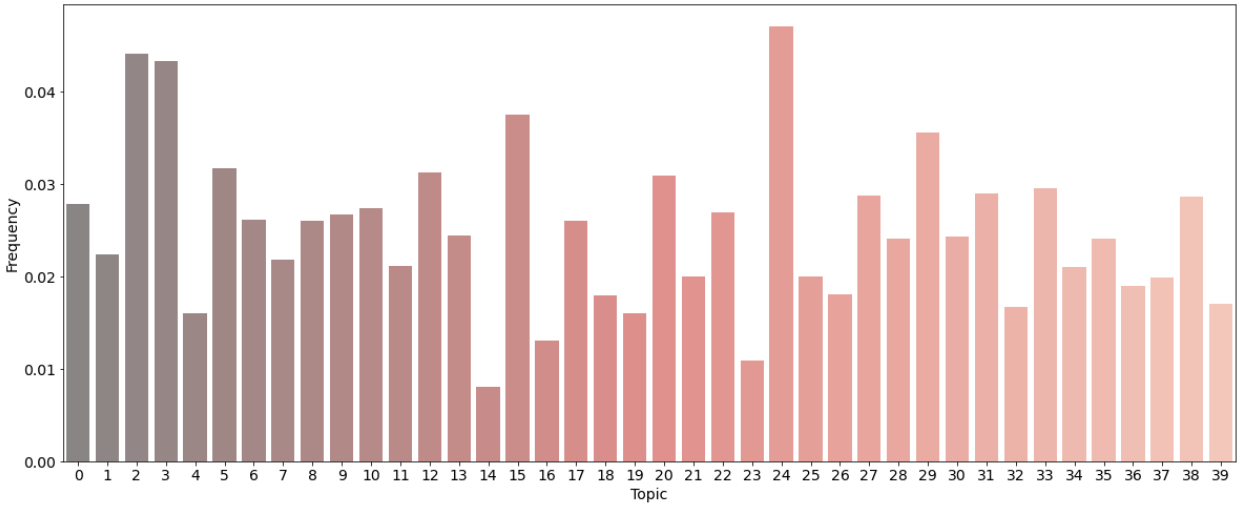

Fig 37. Distribution of documents across each topic.

Dominant_Topic	Topic_Keywords	Num_Documents	Perc_Documents
24	['debugging', 'testing', 'time', 'service', 'data format, login shells', 'visuals', 'cloud platforms', 'programming', 'scripting', 'databases']	4708	0.0471
2	['fies', 'directory', 'path', 'address', 'html', 'web developement', 'web page', 'internet', 'object oriented programming', 'scripting']	4412	0.0441
3	['data structures', 'website', 'syntax', 'query', 'databases', 'emails', 'comments']	4326	0.0433
15	['functions', 'general programming', 'memory', 'space', 'data', 'information', 'security', 'databases', 'dataframes', 'tables']	3747	0.0375
29	['c', 'object oriented programming', 'web developement', 'web frameworks', 'web servers', 'error', 'store', 'loop', 'directory']	3556	0.0356
5	['functions', 'scripting', 'apps', 'web design', 'functions', 'general programming', 'memory', 'space']	3168	0.0317
12	['web frameworks', 'functions', 'scripting', 'cloud platforms', 'programming', 'scripting', 'databases']	3133	0.0313
20	['query', 'databases', 'functions', 'scripting', 'string', 'data type', 'text', 'scripting']	3090	0.0309

Fig 38. Table showing the distribution of documents across topics sorted in descending order of document frequency.

8. Predicting unseen documents

Prediction using unseen data was done by selecting 100,000 other documents from the preprocessed dataset that we already had. Interestingly, the most of the top overall keywords resulting from the prediction are similar to the ones seen for the prediction done on the training data. In both cases, the eight highest occurring keywords are:

1. *Scripting*
2. *Databases*

3. *Programming*
4. *Web frameworks*
5. *Apps*
6. *Object oriented programming*
7. *Functions*
8. *Cloud platforms*

The table in *Fig 39* shows a comparison of the top keywords between train and test results. Notice that from the ninth keyword onward, the distribution starts to differ between the two sets. This indicates that the model generalizes particularly well when it comes to the top keywords and then sees some variation as less frequent keywords are encountered.

However, in order to get a sense of weather prediction on unseen data is actually effective, the frequencies were explored in further detail. *Fig 40* shows the relative frequencies between train and test predicted data across all keywords. From the figure, we can see that for most of the high frequency keywords, the **test numbers are higher**. The sum of the frequencies of the top eight most frequent keywords for the test data (233,215) is approximately 30,000 higher than that of the train data (214,994). Considering that the total sum for all keywords are 885,299 and 884,673 for test and train data respectively, the difference between the top eight keywords seems significant, where the distance between the totals is about 600.

This shows that the prediction on the unseen documents is somewhat effective as the frequencies of the top topics are consistently higher than that of the test data.

	word_trian	freq_train	word_test	freq_test
0	scripting	48700	scripting	56800
1	databases	44486	databases	44746
2	programming	25681	programming	24857
3	web frameworks	20500	web frameworks	23000
4	apps	19969	apps	22640
5	object oriented programming	19414	object oriented programming	21672
6	functions	18244	functions	21300
7	cloud platforms	18000	cloud platforms	18200
8	directory	16400	remote	17143
9	remote	15931	web servers	15957
10	web design	15668	web developement	15100
11	website	15200	directory	13900
12	web servers	13981	web design	12905
13	servers	13787	website	12360
14	authentication	12100	space	12000
15	internet	11814	general programming	12000
16	web developement	10900	memory	12000
17	path	10681	servers	10900
18	address	10681	internet	10700
19	files	10681	ios	10135

Fig 39. Table comparing the top 20 keywords from train and test prediction results.

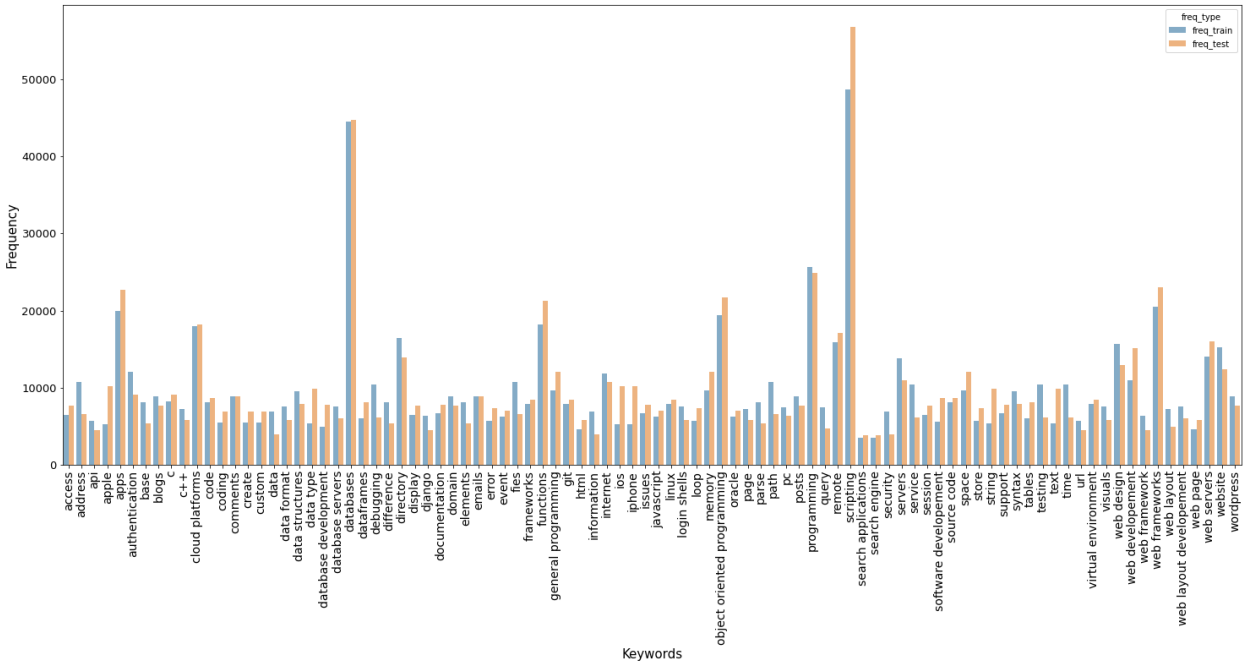

Fig 40. Graph showing the frequency of each predicted keyword for the train and test data.

Fig 41 shows the number of documents each topic appears in for both the train and test set. A key thing to notice is that the most frequently occurring topics contain some of the same keywords which are the most frequent words in the predicted results. For example, the highest document frequency for the test predictions occurs for the topic **[‘functions’,’general programming’,’memory’,’space’]** where words such as **functions** and **programming** are within the top 8 most frequent keywords. Another high document frequency topic is **[‘functions’,’scripting’]** where both words are high frequency keywords.

These topics also have a high document frequency for the train set. This indicates that in general, the high frequency keywords are predicting the most documents which makes sense.

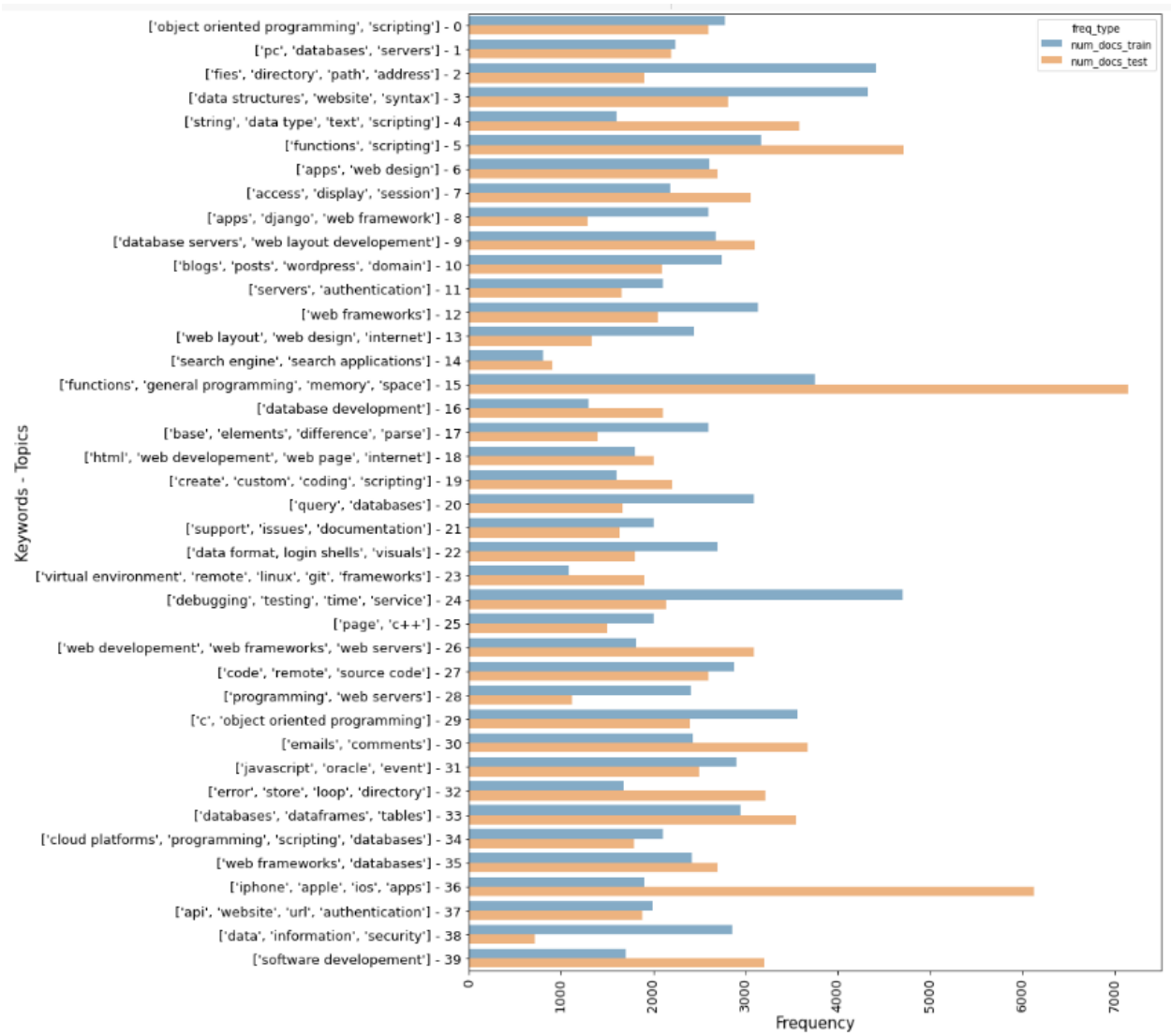

Fig 41. Graph showing the document frequency for each topic for the train and test data.

9. Conclusion

The final LDA model does a reasonable job in predicting unseen documents. However, it could sometimes over-predict a document. This is because in some cases, the number of keywords within a topic could exceed the number of words in the document. Also, since a topic could be a combination of two or more themes, it would use extra words to explain the document. Hence, it would over-explain certain documents.

In terms of predicting unseen data, the model provides similar results as the train data where both sets see the same eight most frequently occurring keywords such as **scripting**, **databases**, and **programming**. For lesser frequent words, the similarities between the sets

reduce. When looking at the topics which cover the highest number of documents, both train and test sets have a few similar topics that provide high document frequencies.

Findings such as the most frequently occurring topics can be used by Stack Exchange or other forums or search engines to quickly filter search results due to the topic being so prevalent. Less frequent topics, on the other hand, could be promoted in order to find answers to those questions which might not be readily available.

Overall, the model can be applied to other applications such as reviews, articles, social media comments etc., to provide a quick summary providing users with relevant information. Additionally, the model can be used by search engines to display topics relevant to the typed word or words.