Doc2Script: A product for mapping functionality scripts to the documentations by using Latent Dirichlet Allocation

Hardik Prabhu

July 2020

Abstract

We have been provided a corpus of functionality scripts. The scripts are provided in JSON format. We wish to understand the functionality tested by the scripts. We approach the task by first extracting different documents corresponding to the different functionality. We would then devise a method to map the scripts to the documents using Latent Dirichlet Allocation (LDA), which is a probabilistic topic modeling technique. In the end, we would develop a product for Temenos which would accept the scripts as input, HTML files containing the documentation, extract documents from them and then map the scripts to the documents.

1 Latent Dirichlet Allocation

1.1 Notion and Terminology

The objective of any topic modelling technique is of discovering the abstract "topics" that occur in a collection of documents.

LDA is a three-level hierarchical Bayesian model, in which each document of a collection is modeled as a finite mixture over an underlying set of topics. The documents are represented as random mixtures over latent topics, where each topic is characterized by a distribution over words. The model assumes that the topics are specified before any data is generated. LDA represents documents as mixtures of topics that spit out words with certain probabilities.

A document can be a part of multiple topics, kind of like in soft clustering in which each data point belongs to more than one cluster.

LDA exploits the sparsity of the structure of a document. That is, the documents are mostly made up of a select few topics which themselves have few words occurring with higher probabilities.

We can look at how the topics are distributed and see what the major themes are, both within documents and the topics. We can pick out documents with similar themes in the corpus.

1.2 The Generative Model

Imagine that you're trying to write an article, which has to be 30% about "science" and 70% about "sports". To generate each word in the document we need to :

- First pick a topic (according to the multinomial distribution), for example, you might pick the science topic with 3/10 probability and the sports topic with 7/10 probability.
- Use the topic to generate the word itself (according to the topic's multinomial distribution). For example, if we select the topic "sports", we might generate the word "match" with 30% probability, "stadium" with 15% probability, and so on.

Using such a generative model for a collection of documents, LDA tries to backtrack from the documents to find a set of topics that are likely to have generated the collection, which is done by using the EM algorithm.

- A document is a sequence of N words denoted by $\mathbf{w} = (w_1, w_2, \ldots w_N)$, where w_n is the nth word in the sequence.
- A corpus is a collection of M documents denoted by $D = \{\mathbf{w}_1, \mathbf{w}_2, \ldots \mathbf{w}_M\}$

LDA assumes the following generative process for each document $\mathbf{w}$ in a corpus D :

1. Choose $\theta \sim Dir(\alpha)$.
2. For each of the N words w_n:
   (a) Choose a topic $z_n \sim Multinomial(\theta)$.
   (b) Choose a word w_n from $p(w_n \,|z_n, \beta)$, a multinomial probability conditioned on the topic z_n.

β is a KXV matrix (number of topics X length of dictionary) where each row is the word distribution for a topic. Similar to the topic distribution for documents, we have another Dirichlet prior η associated with the word distribution for topics.

The probabilistic topic modelling methods are based on the "bag-of-words" assumption, that is, the order of words in a document can be neglected. In the language of probability theory, this is an assumption of exchangeability for the words in a document. A classic representation theorem due to de Finetti establishes that any collection of exchangeable random variables has a representation as a mixture distribution. This important theorem is the inspiration behind considering hierarchical Bayesian models.

1.3 A Note on Dirichlet Distribution

The pdf of Dirichlet distribution:

$$Dir(\alpha) = \frac{1}{\gamma(\alpha)} \prod_{i=1}^{K} \theta_i^{\alpha_{i-1}}$$

where $\gamma(\alpha) = \frac{\prod_{i=1}^{K} \Gamma(\alpha_i)}{\Gamma(\sum_{i=1}^{K} \alpha_i)}$, $\alpha = (\alpha_1, \alpha_2 ... \alpha_K)$, and $\sum_{i=1}^{K} \theta_i = 1$

The Dirichlet distribution is often used in Bayesian framework to set priors. Loosely speaking, its a distribution of discrete distributions(pmfs).

In the generative model we have two multinomial distributions. For a given document, to generate a word the first distribution($Dir(\alpha)$) is used for choosing a topic, and then, for that topic, to select a word from vocabulary, the second distribution($Dir(\eta)$) is used.

Let us look at an example for clarity. Consider a topic model with only 3 topics.

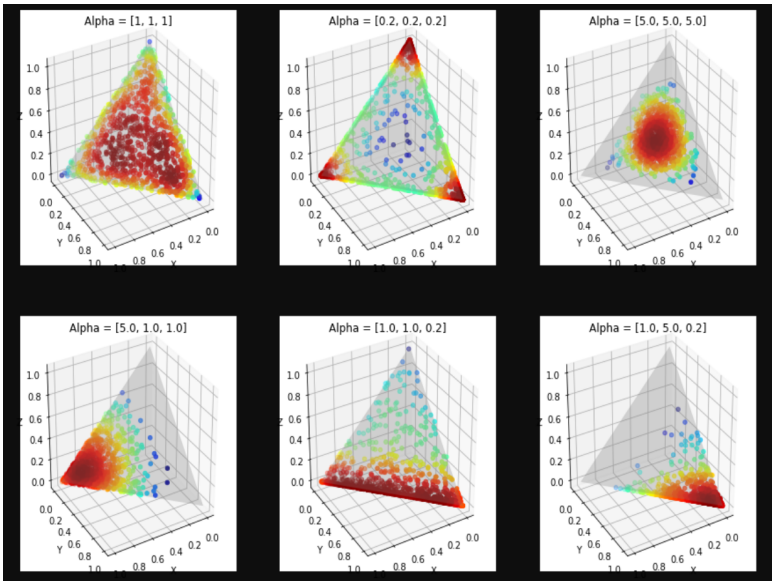
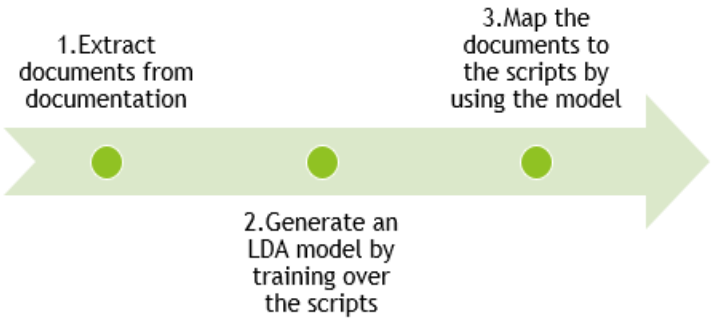


Figure 1: The heat map of probability density for different α values

For the 3 topics, we have the 2-dimensional simplex which contains all the points (x,y,z) such that x+y+z=1. Each of these points represents a multinomial distribution over topics. Each coordinate represents the probability of choosing a particular topic. To generate any document, the model will first select a multinomial distribution for topics and generate each word. For symmetric Dirichlet distributions, a low alpha value places more weight on having each document composed of only a few dominant topics (whereas a high value will

return many more relatively dominant topics). Similarly, a low eta value places more weight on having each topic composed of only a few dominant words.

2 Mapping scripts to documents using LDA



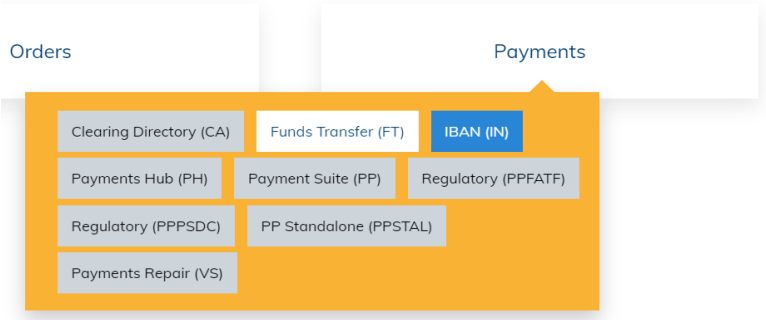
We map the scripts to the extracted documents based on similarity in their text. First, we preprocess the scripts, converting them from JSON to bag of words. After filtering out the noise, we train an LDA model on them to generate topics. Using this model we get the topic distribution for both the scripts and the documents. It is a reasonable assumption that similar documents and scripts have similar topic distribution. Hence, by having a similarity measure based on topic distribution, after setting a threshold value, the scripts and the documents which have qualifying similarity score will get mapped. It is better for the sake of clarity that we look at the entire mapping procedure through an example. In this section, we map the payment scripts to documents using LDA.
We perform the mappings on Python. The gensim module is used for generating the topic model.

2.1 Extracting Documents from Documentations

We will attempt at mapping documents under payment documentation to payment scripts using LDA. The documents corresponding to different functionality are extracted as follows:

1. Under the payments family, we select the options we have access to, which are namely, IBAN.

2. Under IBAN we have 6 documents regarding different functionality. For each of the documents, it has 3 sections containing texts under the headings “Introduction”, “Configuration”, “Working with”. All the texts from the 3 sections are appended together to form each document.



2.2 Training LDA model on scripts

Sticking to the language of text collections, we refer to the entities such as "documents", and "corpus", as:

- **Corpus**: PAYMENTS scripts.
- **Documents**: Scripts in the folder.

The LDA model is constructed in python using the gensim library.

2.2.1 Preprocessing

Reduction: Each Script is a JSON file. We reduce the scripts to only contain the descriptions, field names and applicationid which are present in 'sstObjectList'and"ssaObject".

Catching direct reference: There are scripts that directly refer to another in their description. For example:

'SCRPT170920200621': 'Scenario to upload a CSV file where the transaction has invalid beneficiary IBAN detail and goes to status Error in POA. The results for this scenario is captured in SCRPT170920200620. Creation of a new Customer. To be used as the Credit Account for the Scenario SCRPT171020200544. To be used as the Debit Account for the Scenario SCRPT171020200544. To Load funds to the account 10023137101. Script to upload CSV file for bulk payment'

These Scripts are dependent on each other. It is reasonable to combine these scripts to form a single document to represent them.

Bag of words format: The documents are then converted to a bag of words format, that is, a list containing words.

Removing Stop words: The words that occur commonly in English literature contributes to noise since these occur in abundance. The topics have a

multinomial distribution over the entire vocabulary. Therefore, its important to reduce the size of the vocabulary to useful words. The words "create", "script" and "scenario", also occur frequently across all the documents. These words were also removed.

Catching context using n-gram: The model works on the principle of exchangeability. The downside to this is that the occurrence of collection of words in order(phrases) is considered the same as the occurrence of individual words throughout the document. Some valuable contextual information is lost this way. For practicality, it is sufficient to consider bi-grams and tri-grams. Gensim provides a method to identify the phrases and automatically adds underscore between the words. For example: "New york" would be converted to "New_york", if it occurs frequently in the corpus.

Lemmatization: To reduce the vocabulary even further, we remove inflectional endings of a word to return the base form of the word.

2.2.2 Model inputs

The two main inputs to the LDA topic model are the dictionary(id2word) and the corpus.

- After the prepossessing, we get the corpus in a list of list of words format. Gensim provides a method to create a dictionary by passing the corpus as the input. It creates a unique id for each word in the corpus. Gensim allows us to filter extreme words out of the dictionary. We filtered the words which occurred in more than 60 percent of documents and, the words having a total count below 5.
- The model takes the corpus as the training input, but before feeding the corpus, the documents are to be converted to a vector form. For each word occurring in the document, a tuple of integers is associated. The tuple is obtained by denoting the first entry by the unique word id of the word, and the second entry as the count of total occurrence of the word in the given document. This way a document is converted into a vector containing tuples(word id, count). Gensim provides a method for the conversion.

2.2.3 Model Hyperparameters

To create a model, along with the inputs, we require the following hyperparameters.

- The number of topics. The model assumes the specified number of topics beforehand. (k)
- The hyperparameters concerning the EM algorithm.

- Parameters for Dirichlet distributions($Dir(\alpha)$ and $Dir(\eta)$).

To tune the hyperparameters, we require a measure to evaluate the model. By generating models for different combinations of hyperparameters values, we can select the parameters with a higher evaluation score. U_Mass metric Topic Coherence measures score a single topic by measuring the degree of semantic similarity between high scoring words in the topic. The U_Mass metric defines the score to be based on document co-occurrence:

$$score(v_i, v_j) = log \frac{D(v_i, v_j) + \epsilon}{D(v_j)}$$

where D(x, y) counts the number of documents containing words x and y and D(x) counts the number of documents containing x.

The Coherence score for a topic is the sum of coherence scores for combination of top words in the topic.

$$Coherence(T) = \sum_{(v_i, v_j) \in V} score(v_i, v_j)$$

The U_Mass score is then the average coherence score over all the topics. We evaluate the models based on U_Mass score and select the hyperparameter which results in higher score.

Tuning Model Hyperparameters
The parameters for Dirchlet priors can be set to auto. The model learns about α(document-topic) and η(topic-word) automatically while training.

The hyperparameters related to the EM algorithm are:

- Passes: Number of passes through the corpus during training.
- Iterations: puts a limit on how many times LDA will execute the E-Step for each document.
- Chunksize: Number of documents to be used in each training chunk.
- update_every: Number of chunks to process prior to moving onto the M step of EM.
- evaluate_every: For calculating perplexity after the number of updates. By setting it to "None" improves the runtime.

We will keep the default values of all of the parameters, except the number of passes and iterations. The number of passes and iterations should be high enough. The algorithm will converge after a point and higher values will increase the running time significantly.

We set number of iterations to 1000. By enabling logging, we found that the model converges for iteration=1000, around passes=120. After each pass, we get a topic diff score in the logs. The score measures the differences between each consecutive pass.

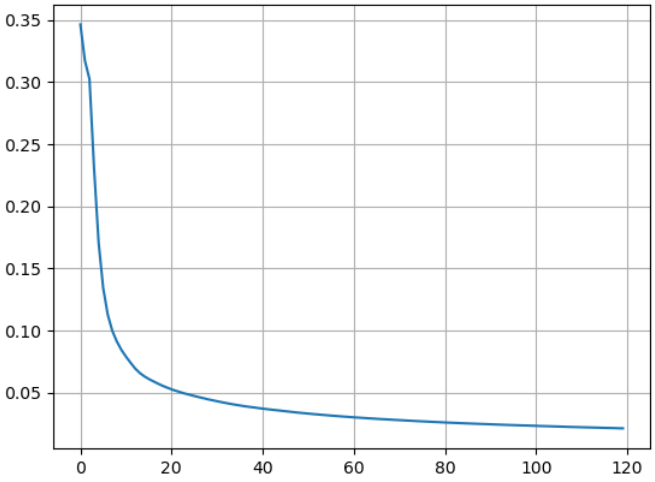


Figure 2: Plot for top diff with respect to number of passes.

For choosing the optimum number of passes, we will first generate a model for an arbitrary number of topics, say number of topics=20, and a high value for number of passes. we can then look into the log file to see the convergence of model.

To check whether, the topic model actually converged for passes =120 for different number of topics, we can use the log file to plot the topic diff over different values for number of topics.

The convergence is not affected significantly with different number of topics. So, we will stick to first generating a model for an arbitrary number of topics to get the value for number of passes and then we will tune the number of topics using the umass scores.

For choosing the optimum number of topics(k). We generated models for different values of k. The k for which we got the highest U_Mass score was selected. For practicality, we set lower-bound on k at 10.

2.3 Mapping the Documents to the scripts

We will attempt to map the documents which were extracted based on different functionality to the scripts by using the topic model which was trained on the scripts.

The scripts on which the model was trained, have a multinomial distribution over the latent topics. We assume that similar scripts have similar distributions. For each document, by using the trained model, we can get the multinomial distribution over the topics. The scripts which have higher similarity for distri-

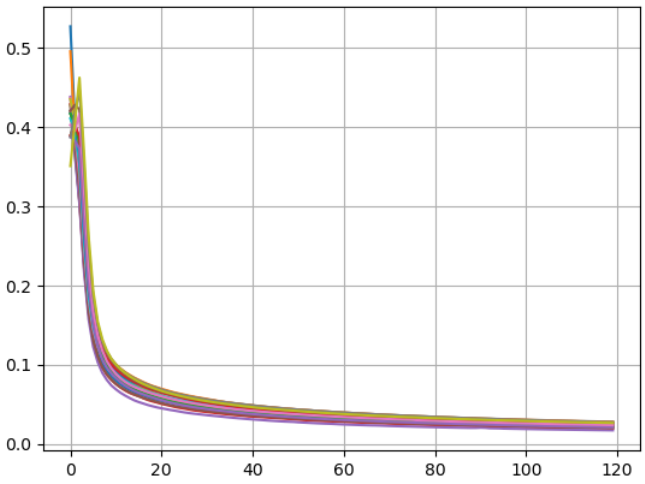


Figure 3: Plot for top diff with respect to number of passes for different number of passes.

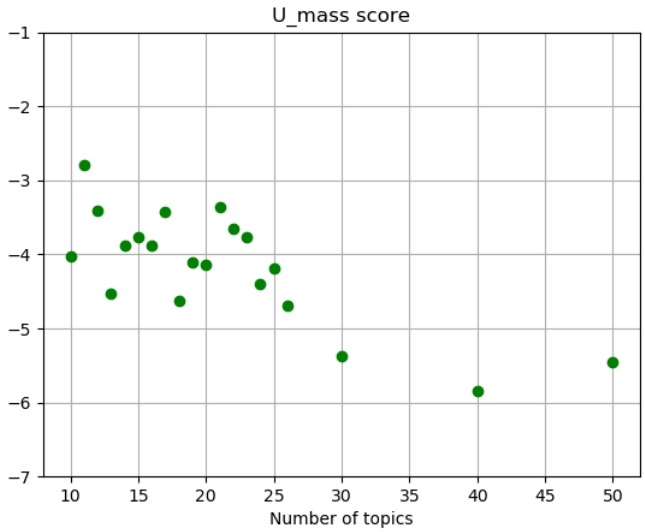


Figure 4: Highest score at k=11.

butions are mapped to the documents.

2.3.1 Similarity Measure:

Jensen–Shannon divergence is a method of measuring the similarity between two probability distributions. It is also known as information radius (IRad) or total divergence to the average. The Jensen–Shannon divergence (JSD) is a symmetrized and smoothed version of the Kullback–Leibler divergence $D(P \parallel Q)$.

$$JSD(PQ) = \frac{1}{2}D(P \parallel M) + \frac{1}{2}D(Q \parallel M)$$

where $M = \frac{1}{2}(P + Q)$. The JSD between two distribution is bounded by the interval [0,1]. Lower value implies higher similarity between the two distributions.

2.3.2 Implementation

1. For each document and script, by using the topic model, we obtain the multinomial distributions over the topics.
2. We calculate the JSD between the documents and the scripts.
3. We set a threshold value. The scripts and the documents which have JSD value below the threshold are mapped with each other.

3 Discussion

3.1 Training Data

The LDA model was trained on the corpus of scripts. Similarly, we could have approached the mapping the other way, by training the model on documents instead of the scripts. Arguments against this approach are,

1. The scripts are concise, it contains only useful information regarding its functionality in text. The documents are descriptive and contain a lot of noise. For example, a sentence like "Here is a screenshot", specifically word "screenshot" is not useful for our purpose of modeling. The keywords on the other hand are going to be present in both the scripts and the documents.
2. Size is also an issue. The number of different documents is always going to be lesser than the number of scripts. In our case, we could only extract 6 documents while the number of scripts was over 5000. By training on scripts we have more data to train on.

While it may be enticing to consider the documents as well as the scripts for training, after all we get the most data for training this way. However, we want to minimize the effect of the noise. In a sense, we want to form a k(number of topics) dimensional space where both the scripts and the documents lie and then based on how close both are we form a mapping. We don't want the noisy words which are not in the scripts to influence their position with respect to a document. We know that the scripts contain all the essential words required for mapping. If the documents aren't used for training, the 'noisy' words wont get included in the dictionary. Also, by excluding documents from the training of the model, we can focus on fewer documents at a time. we can even consider one document at a time and return the scripts which are mapped to them.

In the end, the mappings are data-dependent, so if, we get unsatisfactory mappings, we can still cluster the scripts together based on similarity. But let's not digress from our objective of mapping.

3.2 Cosine Similarity: Another approach at measuring similarity.

Cosine similarity is another approach for measuring similarity. For each document and script, we have distribution over the topics. If we consider the distributions as vectors with dimensions equal to number of topics. We can measure the cosine of the angles between two vectors. The closer the script is to the document, the cosine approaches 1.

Similarly, as we did with JSD, we can set a threshold value and the scripts are mapped to the documents for cosine value above the set threshold.

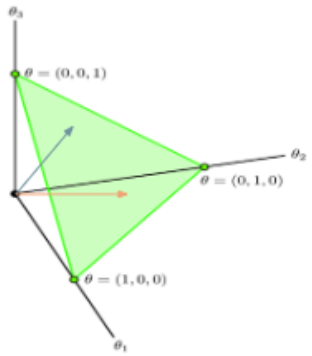


Figure 5: Vector representation for 3 topics model. All the scripts and the documents lie on the green simplex.

3.3 Top-words for Documents

For each of the documents, we would associate a list of words, to represent the mapping of scripts to it. In order to do that, we would consider the topic distribution of the document itself, as for the scripts, having a distribution similar to it is what makes the mapping possible.

By using the topic distribution of the document, we can then get the probability for a word to be generated by the document.

For this, we assume documents to be of length 1 word, and then by using the generative model, we get the probability of the word given the topic distribution. We then create a list of words in decreasing order of probability to represent the mapping corresponding to each document.

Given the parameters α and β, the joint distribution of a topic mixture θ, a set of N topics z, and a set of N words w is given by:

$$p(\theta, z, w|\alpha, \beta) = p(\theta|\alpha) \prod_{n=1}^{N} p(z_n|\theta) p(w_n|z_n, \beta)$$

For N=1,

$$p(\theta, z, w|\alpha, \beta) = p(\theta|\alpha) p(z_1|\theta) p(w_1|z_1, \beta)$$

where $w = \{w_1\}$ and $z = \{z_1\}$.

$$\Rightarrow p(\theta, w|\alpha, \beta) = p(\theta|\alpha) \sum_{z_1} p(z_1|\theta) p(w_1|z_1, \beta)$$

$$\Rightarrow p(w|\theta, \alpha, \beta) = \sum_{z_1} p(z_1|\theta) p(w_1|z_1, \beta)$$

Under our assumption of each document being of length 1 word, $p(w|\theta, \alpha, \beta)$ is the probability of generating that word w_1 for θ topic distribution. By looking at the equation, this probability is the weighted sum of probability of that word for the given topic summed over all topics. The weights are the probability of the topic given the document(θ).

If we calculate the probabilities for all the words in the vocabulary, and arrange them in decreasing order, by selecting first n words, we get the list of n top-words.

4 Pitching the Product

The process of mapping scripts to documents can be divided into 4 steps.

1. Preprocessing of scripts.
2. Tuning model parameters and training.
3. Extracting documents from documentation.

4. Mapping Scripts to documents.

Instead of going through each step manually, we provide an interface which accepts as inputs, the scripts, HTML files containing the documentation, extract documents from them and then map the scripts to the documents and return all the mappings and top-words for each document.

By abstracting the technical stuff away, the users are provided with few configuration options. These options are provided with default values as well. With these options, the users can aid the final mapping with their intuition.

4.1 Application Architecture

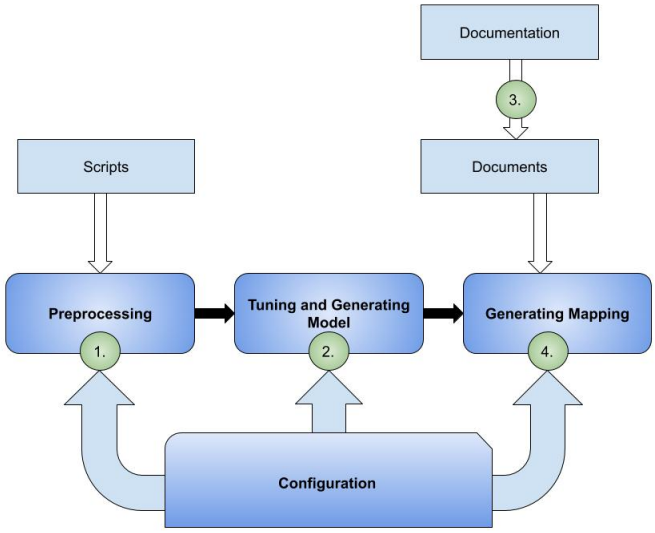


Figure 6: Application Architecture

4.2 Configuration

The users are provided with few configuration options which are stated below.

Reduction of scripts from JSON to txt: The scripts are converted from the JSON file to text. By default, the descriptions, field names, and applicationid which are present in 'sstObjectList' and "ssaObject" are selected. Users can specify if anything else should be added.

Filtering Dictionary extremes: The model uses a dictionary for training/mapping. The user can filter out the words from the dictionary that occur in total less than m times and in more than n percent of scripts.

Tendency to catch phrases: The model will catch more/fewer phrases based on the value given for this parameter.

Adding Stop-words: The user can add to the list of stop-words if they feel like certain words are not significant for mapping. They can do this by creating a text file containing stop-words(space separated)

Dirichlet Parameters: The default values are set to auto ("alpha"=auto,"eta" =auto). The model will automatically learn asymmetrical "alpha" and "eta". The user can also set symmetrical "alpha", "eta".

Similarity measure used for mapping: We have two measures of similarity in distributions of scripts and documents over the topics. i) Cosine similarity ii) JSD.

The user will be able to set threshold value (in the interval [0,1]). The scripts will get mapped to the documents for a qualifying similarity score.

4.3 Outputs

As outputs, the user will get 2 text files containing:

1. Mappings: For each document, the user will get a list of mapped scripts(which are labelled with scenario-id).

2. Top-words: For each documents, the user will also get a list of words/phrases which contributed in the mapping.

5 Getting Started

5.1 The user interface.

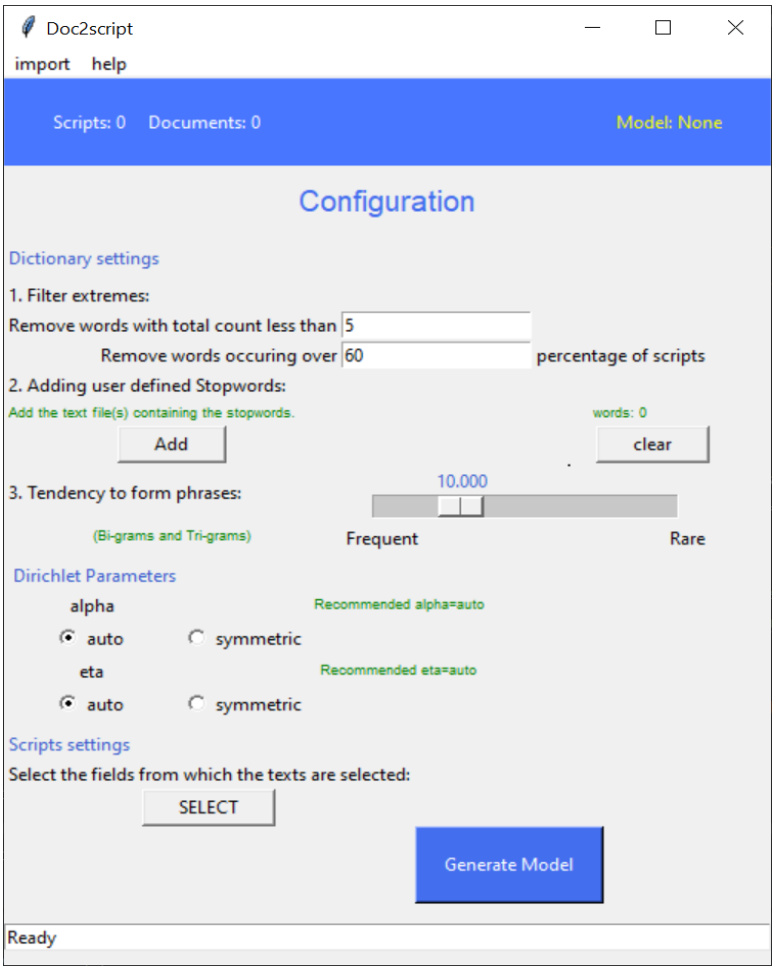


Figure 7: The user interface with default values.

After generating the model, another frame appears which allows users to select similarity measure and set a threshold to generate the mapping.

5.2 Extracting Documents.

Each document extracted from a folder containing HTML files. These folders themselves belong inside a particular folder. The path of that folder is to be specified. For importing documents, select the import option under the menu

bar, specify the folder path and then the documents are automatically extracted. If the documents belonging to different folders are to be extracted, repeat the procedure again, and specify the other folder path.

Example: For extracting payment documents under IBAN, only the path of the IBAN folder is to be specified. Inside IBAN we have several folders containing HTML files "working with", "introduction" and "Configuration". All the texts from these files are appended together to form each document. We can repeat the process and also add the documents under other folders such as "Funds Transfer". Note that the documents don't contribute to the training of the model, they can be added after generating the model as well.

The documents are not required for generating the model. Documents can be added after generating the model. Since model generation is time-consuming, while mapping, the users are provided with the option of clearing the previously added documents so that they can perform the mappings again on a new set of documents using the same model.

5.3 Tuning Parameters

The generation of the model can be time-consuming depending upon the size of the grid(list for different values for number of topics), top diff threshold, and upper limit for the number of passes. All these variables can be managed by opening the LDA.py file. To further decrease the time, the user can set the variable "optimize"(on line 16) to "False" and then set the number of passes and topics manually. The script LDA.py contains comments for further guidance.

6 References

- **David M. Blei, Andrew Y. Ng, Michael I. Jordan**, Latent Dirichlet Allocation, Journal of Machine Learning Research 3 (2003) 993-1022
- **Keith Stevens, Philip Kegelmeyer, David Andrzejewski, David Buttler**, Exploring Topic Coherence over many models and many topics, proceedings of Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning, Jeju Island, Korea, July 2012.
- **Wikipedia contributors**. "Dirichlet distribution." Wikipedia, The Free Encyclopedia. Wikipedia, The Free Encyclopedia, 29 Jun. 2020. Web. 30 Jun. 2020.
- **Wikipedia contributors**. "Jensen–Shannon divergence." Wikipedia, The Free Encyclopedia. Wikipedia, The Free Encyclopedia, 4 Jun. 2020. Web. 30 Jun. 2020.