Guidelines for Calibrating Nucleus APIs

SumUp Analytics, Proprietary & Confidential

The purpose of this document is to facilitate the calibration of Nucleus APIs' input arguments by end-users.

Each section is dedicated to an API, and within each section, a subsection is dedicated to a problem an end-user is trying to solve.

0) Dataset Creation and Upload

a. I have a set of documents (txt, docx, csv, pdf, URL) that I want to upload to create a dataset off of them. How do I do that?

For txt, docx, csv, pdf: use the API Upload with input args {file, dataset}.

If the documents you work with are either txt, docx or pdf then you will need to proceed with a single document per API call. By specifying the same dataset name in the append_file_to_dataset API, those documents will get inserted into the same dataset.

If the documents you work with are csv/xls then a csv file will be uploaded as a collection of documents, where each row of the csv/xls file is considered to be a distinct document.

```
api_instance = nucleus_client.DatasetsApi(nucleus_client.ApiClient(configuration))
file = 'quarles20181109a.pdf' # file |
dataset = 'dataset_from_file' # str | Destination dataset where the file will be inserted.
api_instance.post_upload_file(file, dataset)
```

For URLs: use the API Upload with input args {file_url, dataset, filename, metadata}. The input arguments 'filename' and 'metadata' are optional.

This URL-based upload notably enables users to upload from an AWS S3 bucket.

```
api_instance = nucleus_client.DatasetsApi(nucleus_client.ApiClient(configuration))
dataset = 'dataset_from_url'
file_url = 'https://www.federalreserve.gov/newsevents/speech/files/quarles20181109a.pdf'
payload = nucleus_client.UploadURLModel(dataset=dataset, file_url=file_url) # UploadURLModel |
api_response = api_instance.post_upload_url(payload)
```

b. I have a database table containing lots of documents. I want to upload it to create a dataset. How do I do that?

Every database has a different schema, and for security reasons, most enterprise databases are only accessible internally. Therefore, in the current release of Nucleus APIs, database tables are not yet supported through a dedicated upload API. Users who wish to connect such data source to the APIs need to write a custom script leveraging the Nucleus library function post_append_json_to_dataset.

The db table content would need to be read line by line. Each line is then inserted in the dataset as per the below example.

```
for row in reader:
    document['time'] = row['time']
    document['content'] = row['content']
    document['title'] = row['title']
    payload = swagger_client.Appendjsonparams(dataset='db_test', language='english', document=document)
    try:
        response=api_instance.post_append_json_to_dataset(payload, async=True)
    except ApiException as e:
        print("Exception when calling DatasetsApi->post_append_json_to_dataset: %s\n" % e)
```

1) Dataset Deletion

a. I want to delete a dataset. How do I do that?

Use the DeleteDataset API. Please note that there is no dataset backup API currently available, so the deletion of a dataset can only be reverted by a SumUp System Admin. You would need to re-upload all the documents from this dataset to reinstate it on your end.

```
api_instance = nucleus_client.DatasetsApi(nucleus_client.ApiClient(configuration))
dataset = 'dataset_from_json'
payload = nucleus_client.Deletedatasetmodel(dataset=dataset) # Deletedatasetmodel |
 api_response = api_instance.post_delete_dataset(payload)
```

b. I want to delete specific documents from a dataset. How do I do that?

Use the DeleteDocument API. Please note that there is no dataset backup API currently available, so the deletion of a document can only be reverted by a SumUp System Admin. You would need to re-upload that document in the dataset to reinstate it on your end.

```
api_instance = nucleus_client.DatasetsApi(nucleus_client.ApiClient(configuration))
dataset = 'dataset_from_json'
targetDocid = 'integer_id_of_doc_to_delete'
payload = nucleus_client.Deletedocumentmodel(dataset=dataset, docid=targetDocid) #
Deletedocumentmodel |
api_response = api_instance.post_delete_document(payload)
```

2) **Topic Modeling**

   a. **The topics I get are too high-level, or they are well-known for someone with domain expertise in my field**

   Increase the number of topics 'num_topics' identified and extracted by the algorithm and look at the higher-order topics.

```
api_instance = nucleus_client.TopicsApi(nucleus_client.ApiClient(configuration))
dataset = 'dataset_example' # str | Dataset name.
num_topics = 20 # int | Number of topics to be extracted from dataset. (optional) (default to 8)
api_response = api_instance.get_topic_api(dataset, num_topics=num_topics)
```

   Are the higher-order topics more specific and relevant? Great.

   Are all these topics still too high-level? Add their respective keywords to the input argument 'custom_stop_words' and rerun the Topic Model API.

```
api_instance = nucleus_client.TopicsApi(nucleus_client.ApiClient(configuration))
dataset = 'dataset_example' # str | Dataset name.
custom_stop_words = ' "word1", "word2" '
num_topics = 20 # int | Number of topics to be extracted from dataset. (optional) (default to 8)
api_response = api_instance.get_topic_api(dataset, num_topics=num_topics,
custom_stop_words=custom_stop_words)
```

   Do you still find some topics are relevant to you, but they are a bit too broad? In this case, go to the subtopics level.

   - Run a first-pass topic model
   - Select the topics you identify as relevant
   - Extract the keywords from those topics, and run a second-pass topic model that contains these keywords as user-query in the input argument 'query'

```
api_instance = nucleus_client.TopicsApi(nucleus_client.ApiClient(configuration))
dataset = 'dataset_example' # str | Dataset name.
num_topics = 20 # int | Number of topics to be extracted from dataset. (optional) (default to 8)
api_response = api_instance.get_topic_api(dataset, num_topics=num_topics)
pprint(api_response) # Retrieve keywords from topics of interest

num_topics = 6 # int | Number of subtopics to be extracted from dataset. (optional) (default to 8)
query = ' ("word1" OR "word2" OR … OR "wordN") ' | Fulltext query
api_response = api_instance.get_topic_api(dataset, query=query, num_topics=num_topics)
```

   b. **The topics I get are too narrowly defined**

   Increase the number of keywords 'num_keywords' per topic that is identified and extracted. Rerun the Topic Model API.

```
api_instance = nucleus_client.TopicsApi(nucleus_client.ApiClient(configuration))
dataset = 'dataset_example' # str | Dataset name.
```

```
num_keywords = 20 # int | Number of keywords per topic that is extracted from the dataset. (optional) (default to 8)
api_response = api_instance.get_topic_api(dataset, num_ keywords =num_ keywords)
```

c. Several of the topics I get contain some generic words for someone with domain expertise in my field

Add these words as stopwords in the input argument 'custom_stop_words' and rerun the Topic Model API.

```
api_instance = nucleus_client.TopicsApi(nucleus_client.ApiClient(configuration))
dataset = 'dataset_example' # str | Dataset name.
custom_stop_words = ' "word1", "word2" '
api_response = api_instance.get_topic_api(dataset, custom_stop_words=custom_stop_words)
```

d. I want to exclude certain documents in my dataset from the topic modeling analysis

If you know the document ID of these documents, you can pass them in the input argument 'excluded_docs' of the Topic Model API and rerun.

If you know the titles of these documents, first use the Doc Info API to retrieve the corresponding document IDs, then pass them in the input argument 'excluded_docs' of the Topic Model API and rerun.

```
api_instance = nucleus_client.TopicsApi(nucleus_client.ApiClient(configuration))
dataset = 'dataset_example' # str | Dataset name.
excluded_docs = ' "docid1", "docid2" ' | ID of documents to be excluded
api_response = api_instance.get_topic_api(dataset, excluded_docs =excluded_docs)
```

3) Topic Summarization

a. The summary I get is too dry, I would like a bit more contextual understanding

Increase the value of the input argument 'context_amount' to the Topic Summary API and rerun. Every increment gives you an extra contextual sentence.

```
api_instance = nucleus_client.TopicsApi(nucleus_client.ApiClient(configuration))
dataset = 'dataset_example' # str | Dataset name.
context_amount = 1 # int | The number of sentences surrounding key summary sentences in the documents that they come from. (optional) (default to 0)
api_response = api_instance.get_topic_summary_api(dataset, context_amount=context_amount)
```

b. The summary I get covers too much of an overview content of the document. I want it to be about more niche elements, or more contrasted

Identify words characteristic of the overview content:

- Either by reading the sentences currently produced in your summary and deciding

- Or by grouping those sentences from several documents that you are working on into a new dataset and running this dataset through the Topic Model API. In this case, the topics that you identify and extract can be used as the characteristic words of the overview content

Add these words as stopwords through the input argument 'custom_stop_words' to the Topic Summary API and rerun the API.

```
api_instance = nucleus_client.TopicsApi(nucleus_client.ApiClient(configuration))
dataset = 'dataset_example' # str | Dataset name.
api_response = api_instance.get_topic_summary_api(dataset)
pprint(api_response)
num_topics = 20 # int | Number of topics to be extracted from dataset. (optional) (default to 8)
api_response = api_instance. get_topic_summary_api (dataset, num_topics=num_topics)
pprint(api_response) # Retrieve keywords from topics of interest

# If you want to determine keywords using a topic modeling approach, pull out every sentence from the above pprint response using the API DocDisplay and write a short script with the function post_append_json_to_dataset

custom_stop_words = ' "word1", "word2" '
api_response = api_instance. get_topic_summary_api (dataset,
custom_stop_words=custom_stop_words)
```

c. I want to exclude certain documents in my dataset from the topic summarization

If you know the document ID of these documents, you can pass them in the input argument 'excluded_docs' of the Topic Summary API and rerun.

If you know the titles of these documents, first use the Doc Info API to retrieve the corresponding document IDs, then pass them in the input argument 'excluded_docs' of the Topic Summary API and rerun.

```
api_instance = nucleus_client.TopicsApi(nucleus_client.ApiClient(configuration))
dataset = 'dataset_example' # str | Dataset name.
excluded_docs = ' "docid1", "docid2" ' | ID of documents to be excluded
api_response = api_instance.get_topic_summary_api(dataset, excluded_docs = excluded_docs)
```

4) Topic Sentiment Analysis

a. I am surprised by the sentiment on one of my topics, it doesn't look right

The Topic Sentiment API provides a few supplemental outputs in the form of document-level sentiment and weights, for the documents in your corpus.

Take a look at those outputs, to detect whether some specific documents have surprisingly large sentiment (whether positive or negative) and/or weight.

If there are such documents: does their sentiment or weight make sense upon reading the content? If no, you can exclude them from the analysis and rerun your API call.

```
api_instance = nucleus_client.TopicsApi(nucleus_client.ApiClient(configuration))
dataset = 'dataset_example' # str | Dataset name.
api_response = api_instance. get_topic_sentiment_api(dataset)
pprint(api_response)
```

b. I want to exclude certain documents in my dataset from the topic sentiment analysis

If you know the document ID of these documents, you can pass them in the input argument 'excluded_docs' of the Topic Sentiment API and rerun.

If you know the titles of these documents, first use the Doc Info API to retrieve the corresponding document IDs, then pass them in the input argument 'excluded_docs' of the Topic Sentiment API and rerun.

```
api_instance = nucleus_client.TopicsApi(nucleus_client.ApiClient(configuration))
dataset = 'dataset_example' # str | Dataset name.
excluded_docs = ' "docid1", "docid2" ' | ID of documents to be excluded
api_response = api_instance. get_topic_sentiment_api(dataset, excluded_docs = excluded_docs)
```

5) Topic Consensus Analysis

a. I am surprised by the consensus on one of my topics, it doesn't look right

The Topic Sentiment API provides a few supplemental outputs in the form of document-level sentiment and weights, for the documents in your corpus.

Take a look at those outputs, to detect whether some specific documents have surprisingly large sentiment (whether positive or negative) and/or weight.

If there are such documents: does their sentiment or weight make sense based on reading the content? If no, you can exclude them from the analysis and rerun your API call.

```
api_instance = nucleus_client.TopicsApi(nucleus_client.ApiClient(configuration))
dataset = 'dataset_example' # str | Dataset name.
api_response = api_instance. get_topic_sentiment_api(dataset)
pprint(api_response)
```

b. I want to exclude certain documents in my dataset from the topic consensus analysis

If you know the document ID of these documents, you can pass them in the input argument 'excluded_docs' of the Topic Consensus API and rerun.

If you know the titles of these documents, first use the Doc Info API to retrieve the corresponding document IDs, then pass them in the input argument 'excluded_docs' of the Topic Consensus API and rerun.

```
api_instance = nucleus_client.TopicsApi(nucleus_client.ApiClient(configuration))
dataset = 'dataset_example' # str | Dataset name.
excluded_docs = ' "docid1", "docid2" ' | ID of documents to be excluded
api_response = api_instance. get_topic_consensus_api(dataset, excluded_docs = excluded_docs)
```

6) Document Summarization

a. The summary I get is too dry, I would like a bit more contextual understanding

Increase the value of the input argument 'context_amount' to the Doc Summary API and rerun. Every increment gives you an extra contextual sentence.

```
api_instance = nucleus_client.DocumentsApi(nucleus_client.ApiClient(configuration))
dataset = 'dataset_example' # str | Dataset name.
doc_title = 'doc_title_example' # str | The title of the document to be summarized.
context_amount = 1 # int | The number of sentences surrounding key summary sentences in the documents that they come from. (optional) (default to 0)
api_response = api_instance.get_doc_summary_api(dataset, doc_title,
context_amount=context_amount)
```

b. The summary I get contains generic content from the document

Identify words characteristic of this generic content:

- Either by reading the generic sentences and deciding
- Or by grouping generic sentences from several documents that you are working on into a new dataset and running this dataset through the Topic Model API. In this case, the topics that you identify and extract can be used as the characteristic words of your generic content

Add these words as stopwords through the parameter 'custom_stop_words' in input to the Document Summary API and rerun the API.

```
api_instance = nucleus_client.DocumentsApi(nucleus_client.ApiClient(configuration))
dataset = 'dataset_example' # str | Dataset name.
doc_title = 'doc_title_example' # str | The title of the document to be summarized.
api_response = api_instance.get_doc_summary_api(dataset, doc_title)
pprint(api_response)

# If you want to determine keywords using a topic modeling approach, pull out every sentence from
#the above pprint response using the API DocDisplay and write a short script with the function
#post_append_json_to_dataset

custom_stop_words = ' "word1", "word2" '
api_response = api_instance. get_doc_summary_api (dataset, doc_title,
custom_stop_words=custom_stop_words)
```

c. **My document summary is not specific enough for someone with domain expertise in my field. I want a summary that focuses more on the niche aspects and the contrasting aspects of this document**

Identify words characteristic of the overview content:

- Either by reading the sentences currently produced in your summary and deciding
- Or by grouping those sentences into a new dataset and running this dataset through the Topic Model API. In this case, the topics that you identify and extract can be used as the characteristic words of the overview content

Add these words as stopwords through the input argument 'custom_stop_words' to the Document Summary API and rerun the API.

```
api_instance = nucleus_client.DocumentsApi(nucleus_client.ApiClient(configuration))
dataset = 'dataset_example' # str | Dataset name.
doc_title = 'doc_title_example' # str | The title of the document to be summarized.
api_response = api_instance.get_doc_summary_api(dataset, doc_title)
pprint(api_response)

# If you want to determine keywords using a topic modeling approach, pull out every sentence from
#the above pprint response using the API DocDisplay and write a short script with the function
#post_append_json_to_dataset

custom_stop_words = ' "word1", "word2" '
api_response = api_instance. get_doc_summary_api (dataset, doc_title,
custom_stop_words=custom_stop_words)
```

7) Content Recommendations

a. **I want to get more recommended documents on my list of topics**

Increase the value of the input argument 'doc_num' and rerun the Doc Recommendation API.

```
api_instance = nucleus_client.DocumentsApi(nucleus_client.ApiClient(configuration))
dataset = 'dataset_example' # str | Dataset name.
num_docs = 30 # int | Number of desired recommended docs per topic
api_response = api_instance.get_doc_recommend_api(dataset, num_docs=num_docs)
```

b. **I want to exclude certain documents in my dataset from the content recommendation**

If you know the document ID of these documents, you can pass them in the input argument 'excluded_docs' of the Doc Recommendation API and rerun.

If you know the titles of these documents, first use the Doc Info API to retrieve the corresponding document IDs, then pass them in the input argument 'excluded_docs' of the Doc Recommendation API and rerun.

```
api_instance = nucleus_client.DocumentsApi(nucleus_client.ApiClient(configuration))
dataset = 'dataset_example' # str | Dataset name.
excluded_docs = ' "docid1", "docid2" ' | ID of documents to be excluded
api_response = api_instance.get_doc_recommend_api(dataset, excluded_docs=excluded_docs)
```

8) **Topic Historical Analysis**

   a. **The charts of topic strength, sentiment and consensus I get are too granular. How can I get them with less frequent updates?**

   Adjust the input argument 'update_period' from being a daily frequency ('d') to a lower frequency such as weekly ('w') or monthly ('m').

   If you want an adjustment that is a bit less radical, you can use the input argument 'inc_step', which reflects how many units of the update_period are taken in between two historical analyses. E.g.: inc_step = 1 with update_period = 'd' means that the historical analysis will run every day in the time period of your dataset. Increase the value of the input argument 'inc_step'.

   b. **I want to adjust the topic modeling itself before conducting a historical analysis on topics. What is the best way?**

   Refer to section 2) of this document: Topic Modeling. That API will run faster so you can refine the topics the way you need.

9) **Author Connectivity**

   a. **I know the name of an author I am interested in analyzing. How do I use the Author Connectivity API?**

```
dataset = dataset # str | Dataset name.
target_author = 'D_Trump16' # str | Name of the author to be analyzed.
api_response = api_instance_topic.get_author_connectivity_api(dataset, target_author)
```