Mulesoft product documentation

Kliton Andrea, PhD

November 21, 2017

Introduction

MuleSoft technical documentation is published at https://docs.mulesoft.com. It starts with a high level overview of the platform and it is categorized by each of the products. In this report we analyze the version of documentation published at August, 2017. The goal is to find those documents that trigger customers to raise tickets with technical support regarding documentation problems.

1 Data retrieval and data preparation

Mulesoft technical documentation source is available at

https://github.com/mulesoft/mulesoft-docs.

The source is downloaded as a zip file in a local environment. Once downloaded we proceed with data cleaning.

1.1 Data cleaning

The unarchived documentation source has the following folder structure: $product/version/\{text \| images \| code\}$.

For each product there could be several release versions. We will keep only the latest version and delete the previous versions. Otherwise there would be a lot of duplicate documents which will skew the results of our analysis.

Each text representation of the document is written in AsciiDoc format. We will work with this format to clean the data and retrieve data features.

The images and code are discarded. We will work only with text.

We will use Pandas dataframes to create the working dataset.

```
def create_dataframe(dir_path):
    corpus = []
    for root, dirs, files in os.walk(dir_path):
        for file_name in files:
            file_n = file_name.split('.')[0]
            if '.' not in file_name:
                continue
            file_ext = file_name.split('.')[1]
            if file_ext != 'adoc' or file_n == '_toc':
                continue
            full_file_path = os.path.join(root, file_name)
            with open(full_file_path) as file:
                content = file.read()
                corpus.append({"file":full_file_path, "text":
    content})
    return pd.DataFrame(corpus)
```

The above function recursively iterates through all folders and creates a row for each document file.

Source documentation could not be used as is for text analysis. It should be processed. More concretely we will:

- Remove asciidoc format tags.
- Remove web links.
- Remove code snippets (mostly xml configurations, probably some java code).
- Remove spaces and new lines (`\n`).
- Treat all the tokens that include period marks . in a special way by replacing those to underscores i.e. `org.mulesoft.module` to `org_mulesoft_module`, `3.8.1` to `3_8_1`, etc. Those are meaningful tokens.

To achieve this goal we use regular expressions.

After tokenizing the text in documents it was established there are documents with a few words. Those documents have been removed when building a classification model.

1.2 Data wrangling

NLP requires the presence of a text corpus.First it is required to tokenize the text.

We will use SpaCy to:

- Remove stop words.
- Lemmatize the text.

And we use gensim for:

- Building phrases (word collocation).
- Create Bag of Words and tf-idf representation of documents.
- Topic analysis.
- Create text corpus.

2 Exploratory data analysis

The distribution of documents over the products is shown in (Figure 1). There is a big concentration of documents in *mule-user-guide* product. Which is actually the core product of the company. It is followed by *release-notes*. This section holds information about the changes in each version release of the product, known issues for each version, etc. For the purpose of this work we will omit this section. One more reason to exclude this section is the bias it creates.

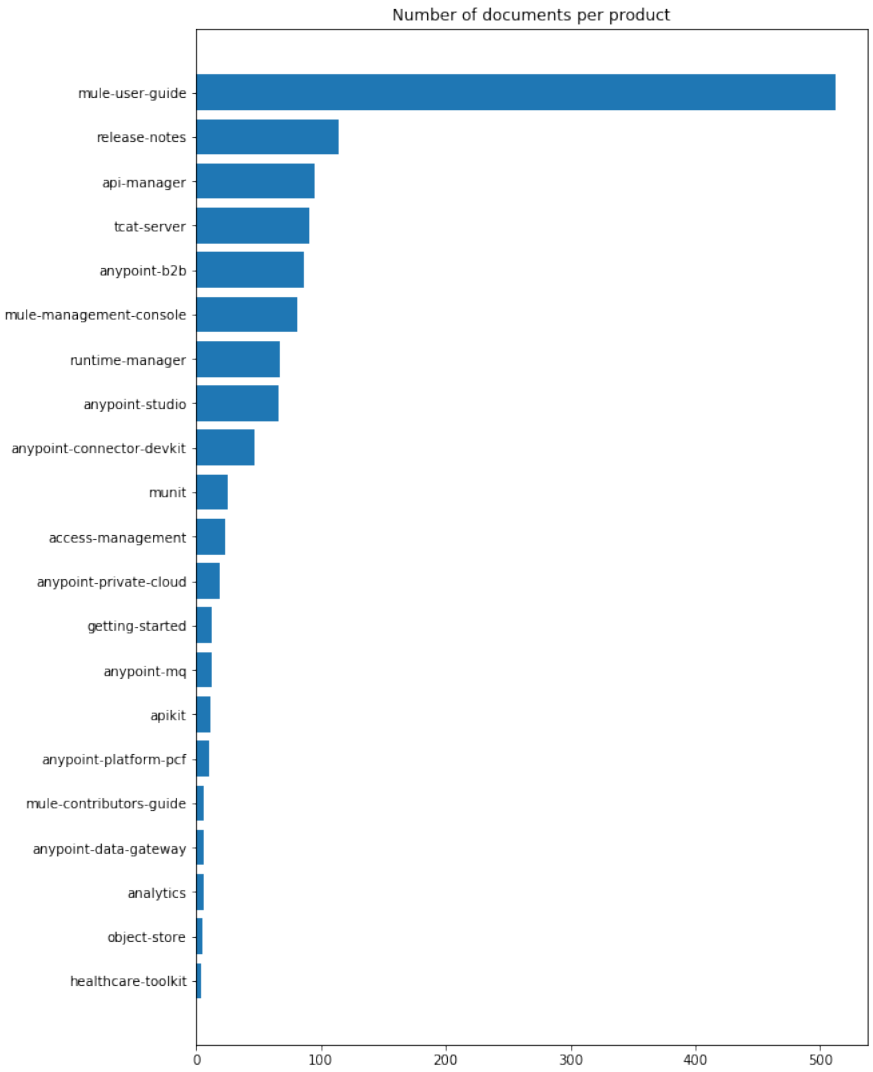

Figure 1: Number of documents per product.

Total number of documents is 1187 after removing *release-notes*.

Next, we create a bag-of-words representation of documents. Basically a bag-of-words is a 2-dimensional matrix. Each row is a document given by the index in dataframe. And each document is a collection of words, indexed in each of the columns. This representation of documents results in a sparse matrix. Each element of the matrix holds a value related to the count of the words in documents. To achieve this we use the following code to create a bag-of-words by counting the number of same words in each document.

```
1 from sklearn.feature_extraction.text import CountVectorizer
```

```
word_vectorizer = CountVectorizer( stop_words='english', min_df
    =2, max_df=0.95, max_features=1000)
bow_1gram = word_vectorizer.fit_transform(corpus_df.stripped_txt
    )
```

From bag-of-words we can calculate the proportion of words that can be found in x or less documents:

```
x, counts = np.unique((bow_1gram > 0).sum(0).tolist()[0],
    return_counts=True)
plt.plot(x, counts.cumsum()/counts.sum());
```

The result is given in (Figure 2).

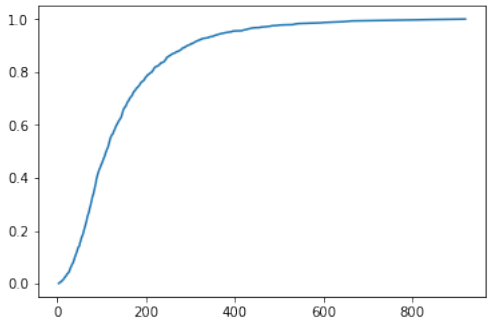

Figure 2: Word frequency proportion in documents.

A word cloud representation of a bag-of-word with CountVectorizer is shown in (Figure 3).

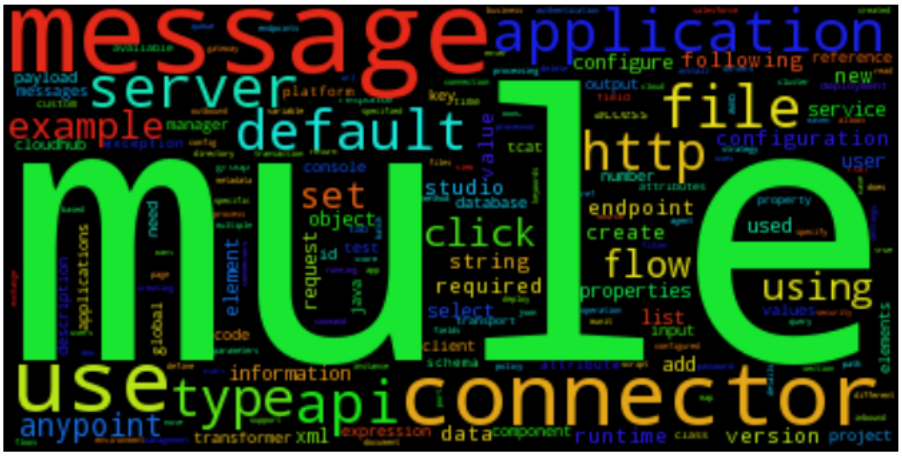

Figure 3: Word cloud with CountVectorizer.

Besides the straightforward count of words in a document, there is another more reliable estimate: term frequency - inverse document frequency (tf-idf) measure. Intuitively, this measure captures the relevance of a term in a given document. If the same term is very common and encountered in all documents, then its weight is decreased by using this measure. For a detailed explanation please refer to Tf-Idf.

When tf-idf is applied, a word cloud looks as in (Figure 4).

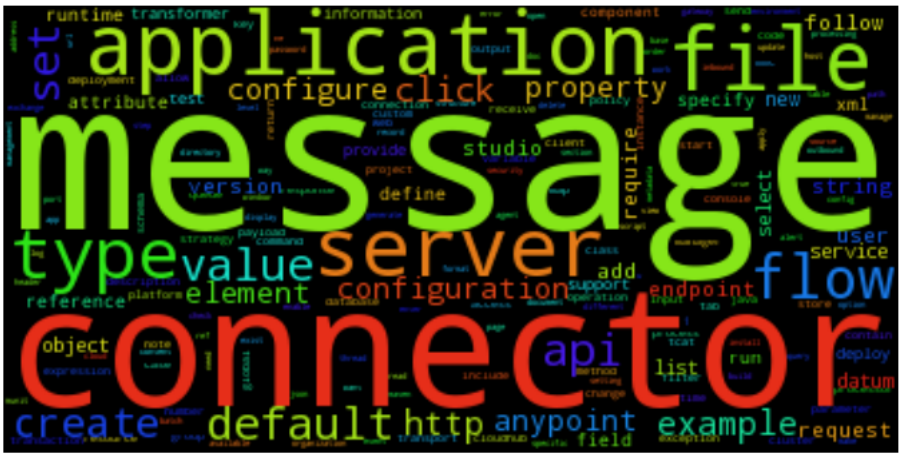

Figure 4: Word cloud with TfIdfVectorizer.

3 Predicting customer issues

4 Conclusions