Mulesoft product documentation

Kliton Andrea, PhD

December 3, 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.

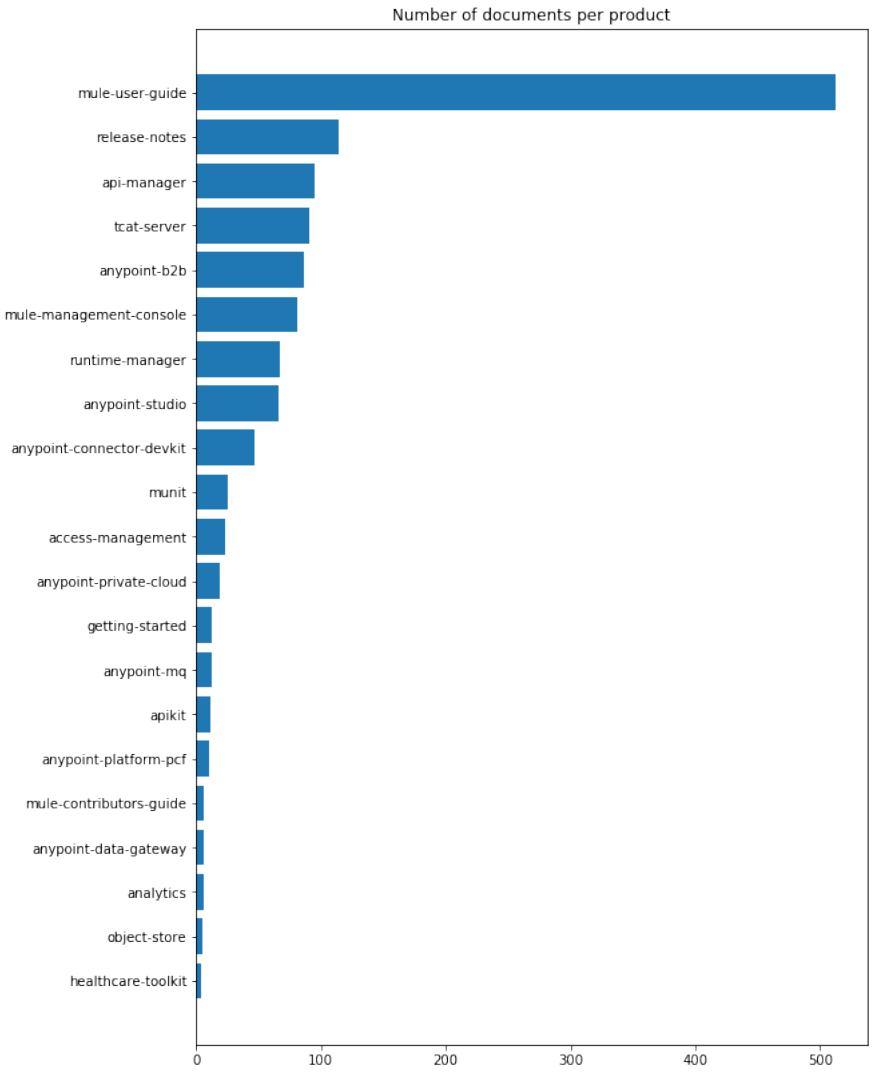

Figure 1: Number of documents per product.

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).

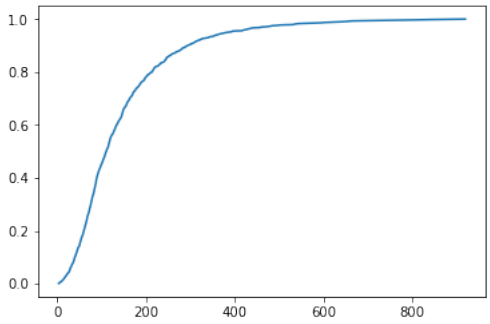

Figure 2: Word frequency proportion in documents.

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

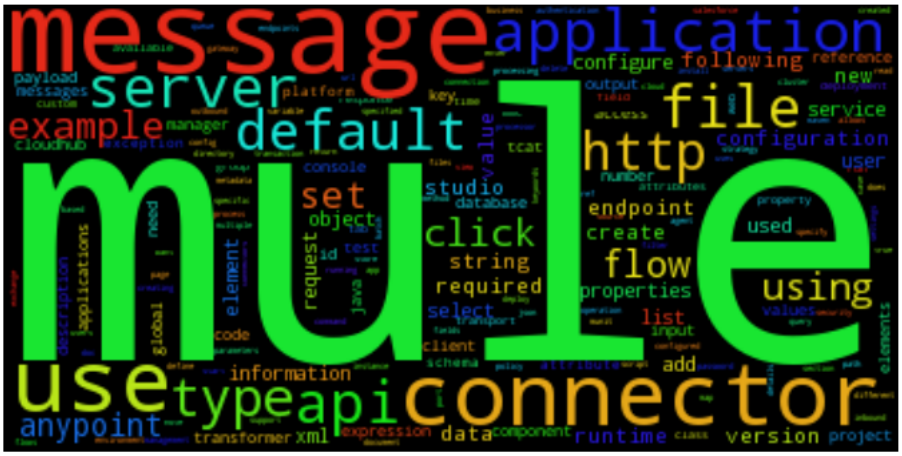

Figure 3: Word cloud with CountVectorizer.

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).

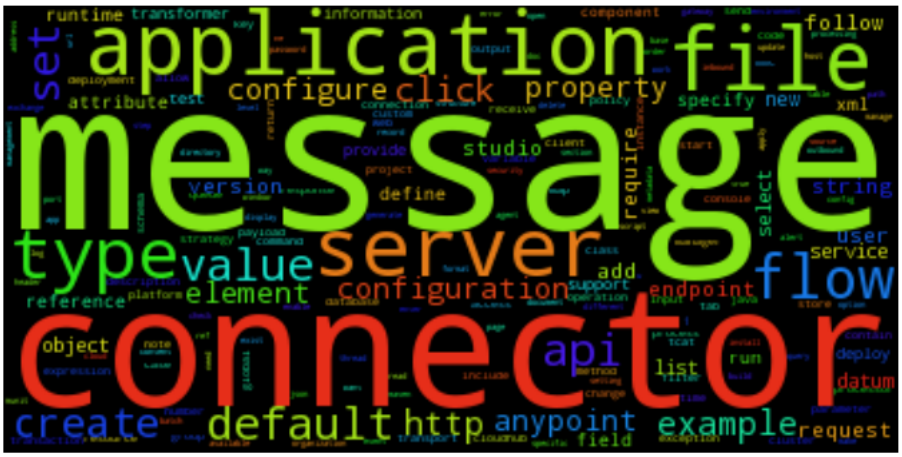

Figure 4: Word cloud with TfIdfVectorizer.

Both wordclouds share some significant words like: message, connector, file, server, flow, etc. These words are pretty common and important in MuleSoft documentation. The word emphasis on each of the wordclouds is different. Our priority will be on analysis performed on Tf-idf transformations.

As part of exploratory data analysis it is important to know the distribution of number of words especially if building prediction models. We will count the retained words after tokenization process is completed. The results are displayed at (Figure 5).

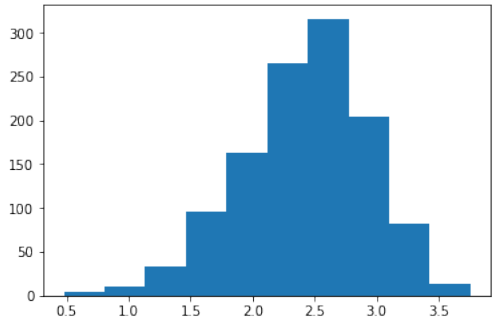

Figure 5: Document length distribution in log10 scale.

The mode of documents length histogram is at 2.5. Which means that most of the documents have approximately 200 tokenized words. There is a small amount of documents that is below 1. For our further investigation we will remove documents that have less than 10 words.

2.1 Reducing the number of features

There are 7372 words in 1176 documents. To reduce the number of components for building a predictive model we will apply PCA.

Cummulative sum of sorted eigenvalues is given in Figure 6.

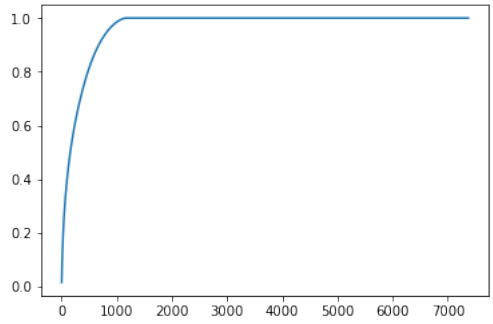

Figure 6: Cummulative variance of word components.

The number of components retaining 85% of variance is 562. Which means we can reduce the number of inputs to predictive model significantly from 7372 to 562.

2.2 Topic Modeling

The documentation is categorized by products. Earlier we mentioned the number of products is 20 (see Figure 1). Where the core product has 1/3 of documents. Assuming that the remaining number of documents is correctly classified, we would expect a total number of topics no less than 24.

For a better estimate we will try the following methods:

2.2.1 Latent Semantic Analysis

This method applies SVD to decompose Tf-idf corpus to document eigenvectors, singular values and term eigenvectors.

The intention is to find a number of topics based on the computed eigenvalues. We calculate the relative cummulative sum of eigenvalues and check visually.

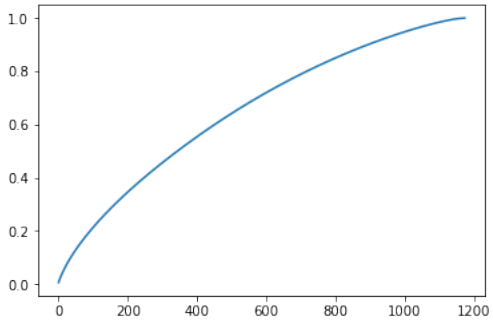

Figure 7: Cummulative sum of eigenvalues.

From Figure 7 there are no insignificant components. It is not possible to reduce the number of topics in this case. Will proceed with other alternatives.

2.2.2 Hierarchical Dirichlet Process

This method as the name states has a hierarchical structure. We can exploit this property to get a better idea defining the number of topics.

Gensim has an implementation of this algorithm which we apply to our data (tf-idf matrix). The result of this algorithm consists in the probabilities of α values for LDA.

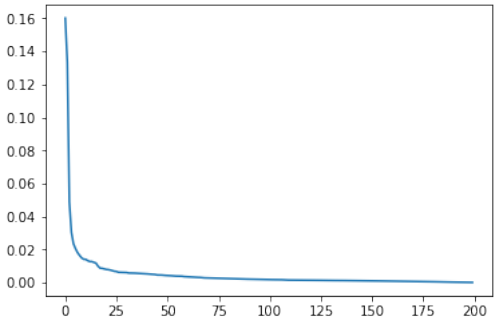

Figure 8: α probabilities calculated from HDP.

From Figure 8, we see that the values of suggested LDA model decrease rapidly until 25 and then they stay at a constant low value. Which means constant low probability.

Normalized cummulative sum of these probabilities achieves a mark of 80% at 58th value. We can consider this value to be the upper bound for the number of topics.

2.2.3 Latent Dirichlet Allocation

Based on the above results we calculate distribution of documnets into topics using Latent Dirichlet Allocation (LDA) implemnted in Gensim. The results of this algorithm are not deterministic. Each time this algorithm is applied the results might differ.

The algorithm was applied for a predefined number of topics of 28. The number of topics is not exact in any of NLP tasks. The number choosen here is arbitrary. To derive it we performed analysis in the above section and we have a lower bound of 24 and an upper bound of 58 for this number.

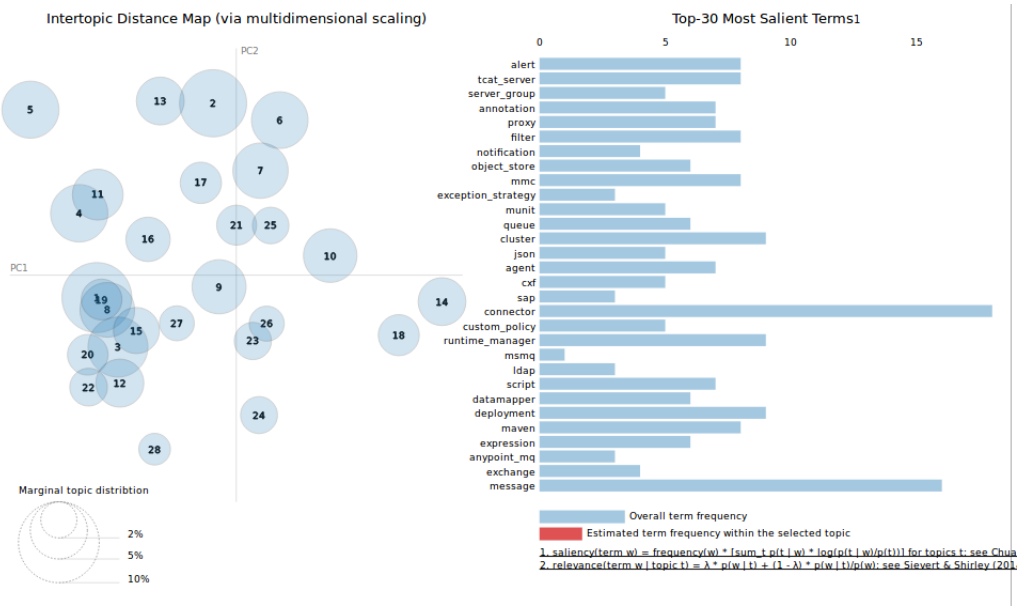

Figure 9: LDA topic modeling for 28 topics.

In previous section, HDP gave a suggested model for LDA. More specifically the α values. However, this vector is not going to be considered in LDA model. α values will be derived from scratch in LDA algorithm.

In Figure 9, topics are displayed in a 2 dimensional space using pyLDAvis.

Distances between The chart is interactive and the user can traverse all the topics to explore the terms for each topic. Topics with shorter distances are semantically related.

3 Predicting customer issues

The purpose of this work is to build a predictive model for a given document as input. The outcome is a binary number. If the outcome is 1 then the document will most likely serve as a trigger for the customers to raise a support a case, otherwise it is 0.

The number of features is reduced by a PCA transformation to 562. We have added the following input features:

- number of images
- number of references to other web pages
- number of code blocks in a document

From our data we have an imbalanced data set. The ratio of documents that have triggered a support case is 11%.

Due to a large number of input features we have decided to apply multinomial Naive Bayes and Support Vector Machine classifiers.

To validate the classifiers we are using the following estimates: area under curve (AUC) and F1-score.

3.1 Multinomial Naive Bayes and validation

This classifier gives train score AUC = 0.56 and F1 = 0.28. The validation on test dataset gives AUC = 0.6 and F1 = 0.33.

That is not a good result. On training datasets this classifier performs a little better than just generating the outcome randomly by flipping a coin.

Interesting enough, on validation set this classifier gives better scores.

3.1.1 Hyper parameter tuning

The classifier was trained by using default hyperparameters. There is only one hyper parameter for this classifier: additive (Laplace/Lidstone) smoothing parameter (α).

By default $\alpha = 1.0$. Tuning this parameter with a grid search algorithm we find out that for smoothing parameter $\alpha = 6$, we get AUC of 0.63 for training dataset. Which is much better than the results above. Validation on test data still holds the same result.

For this classifier this seems to be the limit with the data we have.

3.2 Support Vector Machines and validation

Training SVM with default values gives the following results. On training dataset AUC=0.51, F1=0.03 and testing data AUC=0.5 and F1 = 0. This classifier is not working at all. The next step is to tune the parameters and check the performance again.

3.2.1 Hyper parameter tuning

Γ and C are the parameters we are looking to tune in order to improve the performance. With a grid search we find the following values: $\Gamma = 10e - 07$ and $C = 10000000$. The respective scores are AUC = 0.76 and F1 = 0.51 for training dataset and AUC = 0.71 and F1 = 0.51 for test dataset.

This is a good result and much better than the default classifier.

3.2.2 Learning Curves

Scikit-learn package comes with out-of-the-box learning curves for evaluating the models. In this case we want to validate whether the model is too complex for the number of features.

Usually, when building learning curves one would focus on the error estimates. In this case we will utilize learning curves with AUC estimates.

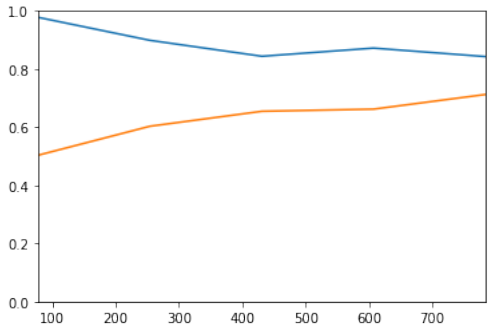

Figure 10: Learning curve for SVM model.

From Figure 10 the training scores (in blue) get a lower AUC scores with more samples being introduced into the training set and get stabilized at that point. Testing AUC scores (in red) get higher with a larger training set. The gap between the curves does gets the smallest value at approximately 10 percentage points.

The learning curve is also used as a tuning tool. The curves above are the result of tuning the parameters to $C = 100$ and $\Gamma = 10e - 04$. The number of features is decreased to 103.

Based on the learning curves, the hyperparameter values ($C = 100$ and $\Gamma = 10e - 04$) and the number of features (103) give a balanced outcome. No overfitting and not much bias either.

3.3 Random Forest

Applying a Random Forest algorithm results in overfitted models. The models fit very well the training data set, but poorly in validation set.

We have tested 3 scenarios.

A random forest with the default values gives AUC=0.95, F1= 0.95 on training data set and AUC=0.58, F1=0.27 on test data set.

We reduce the number of features to 103 and check again. The above values drop with one percentage point. Which does not make a big difference.

Because we have an imbalanced dataset, we will try to add more points. Using imblearn package we add more data points to our dataset. This dataset is used only for learning. We test the dataset with the same test set.

Training the model with more data, we get the following scores AUC=1.0, F1=1.0 for training dataset and AUC=0.63, F1=0.39 for test dataset.

4 Conclusions

In this work we cleaned unstructured data, built a text corpus with Spacy and Gensim packages and carried on exploratory analysis using LSI and LDA.

The topics in the documents are explored with an interactive visualization package like pyLDAvis.

The most important part of work is the predictive model. Investigated several models such as: naive bayes classifier, support vector machines and random forest.

Best classifier in terms of area under the curve and F1 scores is SVM.