

**Technical Design Approach**

**Text Analytics Tool for Data Cataloguing and Classification**

**Version 1.0 , Draft**

**10th September 2018**

**Document History**

|  |  |  |
| --- | --- | --- |
| **Version No** | **Description** | **Author** |
| 1.0 | Draft Document , Data Catalogue Design Technical Design | Lakshmi Krishnamurthy |
|  |  |  |

Contents

[1. Introduction 4](#_Toc524420385)

[2. Scope of Work 4](#_Toc524420386)

[3. Development Platform 5](#_Toc524420387)

[ Option A : KNIME Analytics Server 5](#_Toc524420388)

[ Option B : Open Source (R/ Python Platform) 5](#_Toc524420389)

[4. Design Steps 5](#_Toc524420390)

[4.1 Creating the Master Data Dictionary of Product Master Classes (Nouns) 5](#_Toc524420391)

[4.2 Basic feature extraction using text data 5](#_Toc524420392)

[4.3 Basic Data pre-Processing 5](#_Toc524420393)

[4.4 Basic Text Pre-processing of text data 5](#_Toc524420394)

[4.5 Advanced Text Processing 6](#_Toc524420395)

[4.6 Finding the terms that are the most correlated with each of the products 6](#_Toc524420396)

[4.7 Supervised Classification Modeling - Multi-Class Classifier: Features and Design 6](#_Toc524420397)

[4.8 Model Tuning and Performance 6](#_Toc524420398)

[5. Code Framework (Python) 6](#_Toc524420399)

[5.1 Creating the Master Data Dictionary of MRO Data Catalogue Master (Noun , Modifier and Character ) 7](#_Toc524420400)

[5.2 Basic Data pre-Processing 7](#_Toc524420401)

[6. Advanced Text Processing 9](#_Toc524420402)

[7. Train supervised classifiers /Predict the Noun /Modifier /Character values 10](#_Toc524420403)

[8. Sample UI Output 10](#_Toc524420404)

Introduction

This document describes the proposed approach towards implementation of a Text Analytics tool for Data Catalogue Management .The purpose of the Data Catalogue and Dictionary project is to provide a platform for managing Data Catalogues (Unstructured MRO catalog data ) . This project is proposed to be implemented on a Text Analytics platform, using text /document processing libraries and Machine learning algorithms to process unstructured data so as to store, classify and perform data quality /health checks.

The text analytics engine will serve the following core objectives –

a) Creation of a Master Data dictionary to store the verified and validated data of Nouns , Modifier and Character set.

.b) The tool will use a supervised classification algorithm to correctly detect the Noun , Modifier and Character values.

c) The final part of the tool capability is to produce a data health check output as a visualization report .

Scope of Work

The following section describes the Scope of Work that will be implemented with the Text Analytics tool .

* Master Data Catalogue (Data Dictionary )

The Tool will facilitate the maintenance of the Master Data /Data Catalog for the categories of data including the following :

a – Noun (Product Category Level 1)

b- Modifier (Product Category Level 2)

c-Quantitative Metadata

d- Qualitative Metadata ( Description fields)

e- Manufacturer

f- SKU

* Descriptive Statistics :
  + Word Count of Nouns , Modifier
  + Average Word Length
  + Duplicate Values
  + Spelling Check
  + Known Abbreviations
* Supervised Classification
  + Text processing for Completeness, Consistency , Conformity , Duplicate values and Accuracy
  + Number of NCs(Non Conformities ) – Noun , Modifier and Description
  + Multi label /Multi Class supervised classification of Unstructured dataset to correct labels (Noun /Modifier )
* Data Quality/Health Check Report

The final output of the tool will be a Data Visualization report that gives a visual output from the Text processing (NCs) as well as results from the Supervised Classification .

Development Platform

The following development /deployment platforms are proposed. For the purposes of Code framework design ,the platform design will be proposed with Python .

## Option A : KNIME Analytics Server

## Option B : Open Source (R/ Python Platform)

Design Steps

This section describes the describes the design steps to implement the Machine Learning framework for Text Analytics :

## Creating the Master Data Dictionary of Product Master Classes (Nouns)

The dictionary will be implemented using the reference and correct classification of the Noun /Modifier and Character taxonomy and corresponding values .

## Basic feature extraction using text data

* Number of words
* Number of characters
* Average word length
* Number of stopwords
* Number of special characters
* Number of numerics
* Number of uppercase words

## Basic Data pre-Processing

* Punctuation removal
* Stopwords removal
* Spelling correction
* Tokenization
* Stemming
* Lemmatization

## 4.4 Basic Text Pre-processing of text data

* Lower casing
* Punctuation removal
* Stopwords removal
* Frequent words removal
* Rare words removal
* Spelling correction
* Tokenization
* Stemming
* Lemmatization

## Advanced Text Processing

Advance Text Processing

* N-grams
* Term Frequency
* Inverse Document Frequency
* Term Frequency-Inverse Document Frequency (TF-IDF)
* Bag of Words
* Sentiment Analysis

## Finding the terms that are the most correlated with each of the products

## Supervised Classification Modeling - Multi-Class Classifier: Features and Design

For the modelling phase , we can benchmark on the following algorithms:

* Naïve Bayes
* Support Vector Machine
* Logistic Regression

## Model Tuning and Performance

The final step in the Machine Learning /Algorithm development step will be a Model performance evaluation (ROC /Error Matrix /other) to determine the most favourable approach to

Code Framework (Python)

The following section describes the code framework for implementing the Design steps .

## Creating the Master Data Dictionary of MRO Data Catalogue Master (Noun , Modifier and Character )

**#Importing the Dataset and Basic Data Preparation**

import pandas as pd

df = pd.read\_csv('c:/Text Analytics Engine/Gasket1.csv', low\_memory=False)

df.head()

col = ['Noun', 'Modifier' , 'DESCRIPTION']

df = df[col]

**# Creating the Data Dictionary**

from pandas import \*

category\_id\_df1 = df[['Noun', 'category\_id\_Noun']].drop\_duplicates().sort\_values('category\_id\_Noun')

category\_to\_id1 = dict(category\_id\_df1.values)

id\_to\_category = dict(category\_id\_df1[['category\_id\_Noun', 'Noun']].values)

id\_to\_category

## Basic Data pre-Processing

1. Punctuation removal
2. Stopwords removal
3. Spelling correction
4. Tokenization
5. Stemming
6. Lemmatization

**Removing Punctuation**

Remove punctuation is a key data pre-processing step , as it doesn’t add any extra information while treating text data. Therefore removing all instances of it will help us reduce the size of the training data.

df[' DESCRIPTION '] = df['DESCRIPTION'].str.replace('[^\w\s]','')

df['DESCRIPTION'].head()

**Removing Stopwords**

Stop words (or commonly occurring words) should be removed from the text data. For this purpose, we can either create a list of stopwords ourselves or we can use predefined libraries.

from nltk.corpus import stopwords

stop = stopwords.words('english')

df['DESCRIPTION'] = df['DESCRIPTION'].apply(lambda x: " ".join(x for x in x.split() if x not in stop))

df['DESCRIPTION'].head()

**Spelling Correction**

from textblob import TextBlob

df['DESCRIPTION'][:5].apply(lambda x: str(TextBlob(x).correct()))

**Stemming**

Stemming refers to the removal of suffices, like “ing”, “ly”, “s”, etc. by a simple rule-based approach. We use *PorterStemmer*from the NLTK library.

from nltk.stem import PorterStemmer

st = PorterStemmer()

train['df'][:5].apply(lambda x: " ".join([st.stem(word) for word in x.split()]))

**Lemmatization**

Lemmatization converts the word into its root word, rather than just stripping the suffices. It makes use of the vocabulary and does a morphological analysis to obtain the root word.

from textblob import Word

df['DESCRIPTION'] = df['DESCRIPTION'].apply(lambda x: " ".join([Word(word).lemmatize() for word in x.split()]))

df['DESCRIPTION'].head()

**Remove the Missing values &Add a column encoding Noun/Modifier**

|  |
| --- |
| from io import StringIO  df['category\_id'] = df['MODIFIER'].factorize()[0]  category\_id\_df = df[['MODIFIER', 'category\_id']].drop\_duplicates().sort\_values('category\_id')df = df[col]  df = df[pd.notnull(df['DESCRIPTION'])]  df.columns = ['Modifier ', DESCRIPTION ']  df['category\_id'] = df['Product'].factorize()[0] category\_id\_df = df[['Product', 'category\_id']].drop\_duplicates().sort\_values('category\_id') category\_to\_id = dict(category\_id\_df.values) id\_to\_category = dict(category\_id\_df[['category\_id', 'Product']].values) df.head() |

# Advanced Text Processing

Advance Text Processing involves creating Bag of Words model , Feature extraction , and the Inverse Document: a model where for each raw data input ( a Description entry in our case) , the presence and the frequency of words is taken into consideration, but the order in which they occur is ignored.

* + Feature Extraction , N-grams processing
  + Term Frequency ,Term Frequency-Inverse Document Frequency (TF-IDF)
  + Bag of Words

sklearn.feature\_extraction.text.TfidfVectorizer

from sklearn.feature\_extraction.text import TfidfVectorizer

tfidf = TfidfVectorizer(sublinear\_tf=True, min\_df=5, norm='l2', encoding='latin-1', ngram\_range=(1, 2), stop\_words='english')

features = tfidf.fit\_transform(df.DESCRIPTION).toarray()

labels = df.category\_id

features.shape

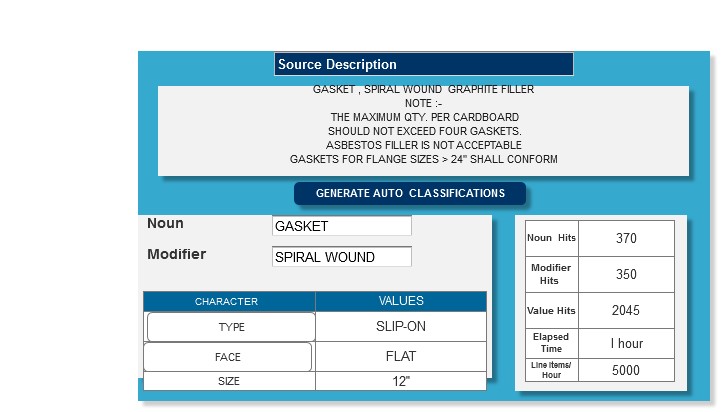
# Train supervised classifiers /Predict the Noun /Modifier /Character values

from sklearn.model\_selection import train\_test\_split  
from sklearn.feature\_extraction.text import CountVectorizer  
from sklearn.feature\_extraction.text import TfidfTransformer  
from sklearn.naive\_bayes import MultinomialNB

X\_train, X\_test, y\_train, y\_test = train\_test\_split(df['DESCRIPTION'], df['MODIFIER'], random\_state = 0)  
count\_vect = CountVectorizer()  
X\_train\_counts = count\_vect.fit\_transform(X\_train)  
tfidf\_transformer = TfidfTransformer()  
X\_train\_tfidf = tfidf\_transformer.fit\_transform(X\_train\_counts)

clf = MultinomialNB().fit(X\_train\_tfidf, y\_train)

# Sample UI Output



Sample Code Output< Under Construction >

category\_id\_df1 = df[['Noun', 'category\_id\_Noun']].drop\_duplicates().sort\_values('category\_id\_Noun')

category\_to\_id1 = dict(category\_id\_df1.values)

id\_to\_category = dict(category\_id\_df1[['category\_id\_Noun', 'Noun']].values)

id\_to\_category

Out[26]: {0: 'GASKET'}

n [49]: category\_id\_df2 = df[['Modifier', 'category\_id\_Modifer']].drop\_duplicates().sort\_values('category\_id\_Modifer')

df['category\_id\_Modifer'] = df['Modifier'].factorize()[0]

category\_to\_id2 = dict(category\_id\_df2.values)

id\_to\_category2 = dict(category\_id\_df2[['category\_id\_Modifer', 'Modifier']].values)

id\_to\_category2

Out[53]: {0: 'SPIRAL WOUND', 1: 'ACTUATOR', 2: 'AC CHILLER UNIT'}