Ezana N. Beyenne

MSDS 453, Section 57 2020

Assignment 3: **Corpus Themes, Identifiable groups or clusters**

Abstract

This assignment builds on the three approaches from the previous assignment namely: (1) Analyst Judgement, (2) TF-IDF and (3) Neural Network Embeddings. We attempt to answer the following questions: are there common themes across the documents, do documents fall into identifiable clusters or groups? We start with the document vectorizations and apply multivariate methods, like cluster analysis and multidimensional scaling and then attempt to provide a picture of the corpus as a whole.

We have been gathering data on autonomous vehicle safety, applying various vectorization approaches with scaling, and analysis to see if we can deduce the ontology for this corpus by using various technique to identify the groups or clusters. We use various unsupervised techniques and try to find the most suited one.

Introduction

An increasing number of articles are being written about autonomous vehicles especially on the artificial intelligence that goes into building them. The technology involves a lot of computer vision and natural language processing algorithms, and software engineering that are complex in nature. The technology is expected to have a positive impact on issues ranging from the environment to road congestion.

Autonomous vehicles are expected to reduce the instance of impaired driving due to either alcohol or drug impairment. Secondly, it will help reduce emissions and costs by finding the fastest routes to destinations thereby improving fuel efficiency. According to the Department of Transportation and the National Highway Traffic Safety Administration, human error causes almost 94% of accidents on US roads (Alessandro, 2020). So, this study hopes to identify the vectorization method that does the best job of classifying autonomous vehicle safety articles into either being about safety or the technology subtopic. There have been other ontologies that have been developed for Safe Autonomous driving by Lihua Zhao et al, with a focus on the route of transportation, and another ontology based on scene creation for autonomous vehicles by Bagschik, Gerrit et al.

Literature review

Regulators and companies at all levels are trying to keep track of the ever-evolving nature of this technology with an eye on both the technology and the safety aspect of it. The organization's need to stay abreast of the rapidly changing technological and safety landscape of autonomous vehicles is a necessity. According to Rand Corporation's research, autonomous vehicles would have to be driven hundreds of millions of miles, (sometimes hundreds of billions of miles), translating to decades, if not centuries to achieve these goals. Since this is not feasible, Rand Corporation recommends regulations that are adaptive in nature, so such regulations harness the benefits while mitigating the risks of this rapidly evolving technology (Kalra et al., 2016). There are currently thirty-seven states, along with the District of Columbia, that have created legislation or issued executive orders about autonomous vehicles.

Methods

We first start by using the matrices for the 3 approaches in the last assignment, and perform a k-means cluster analysis with the documents as objects. I then determined the number of clusters, and prepared a summary of the list of documents within each cluster as seen below.

Table 1: Approach 1(CountVectorizer) top documents with K-means with documents as Objects

```
2A1 CountVectorizer + Kmeans with documents as Objects Top documents per cluster:
Cluster 0: curbanplanning cvehiclegoogle cdriverlesscar cubercities treportdata
Cluster 1: courcities wthisweek treportdata teurope2017 tfooddelivery
```

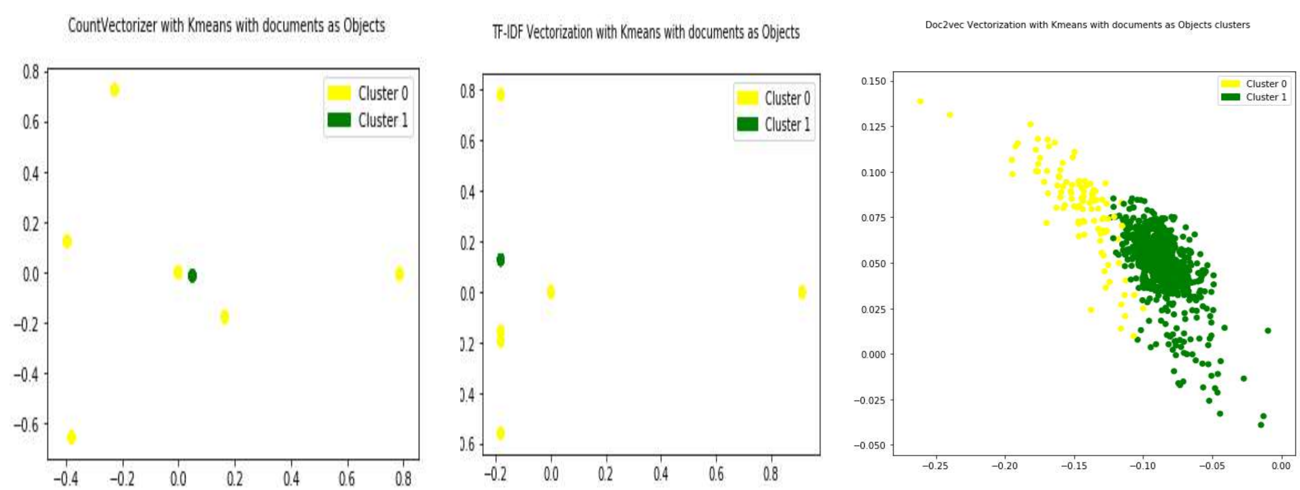
Table 2: Approach 2 (TF-IDF) top documents with K-means with documents as Objects

```
2A2 TF-IDF Vectorization with Kmeans with documents as Objects Top documents per cluster:
Cluster 0: cdriverlesscar courcities treportdata curbanplanning cvehiclegoogle
Cluster 1: cubercities wthisweek tdetroit2017 courcities cproscons
```

Table 3: Approach 3 (Doc2Vec) top documents with K-means documents as Objects

```
2A3 Doc2vec with Kmeans with documents as Objects Top documents per cluster:
Cluster 0: cproscons wregulatethemselves treportdata cubercities nvehiclessafety
Cluster 1: cproscons wregulatethemselves treportdata cubercities nvehiclessafety
```

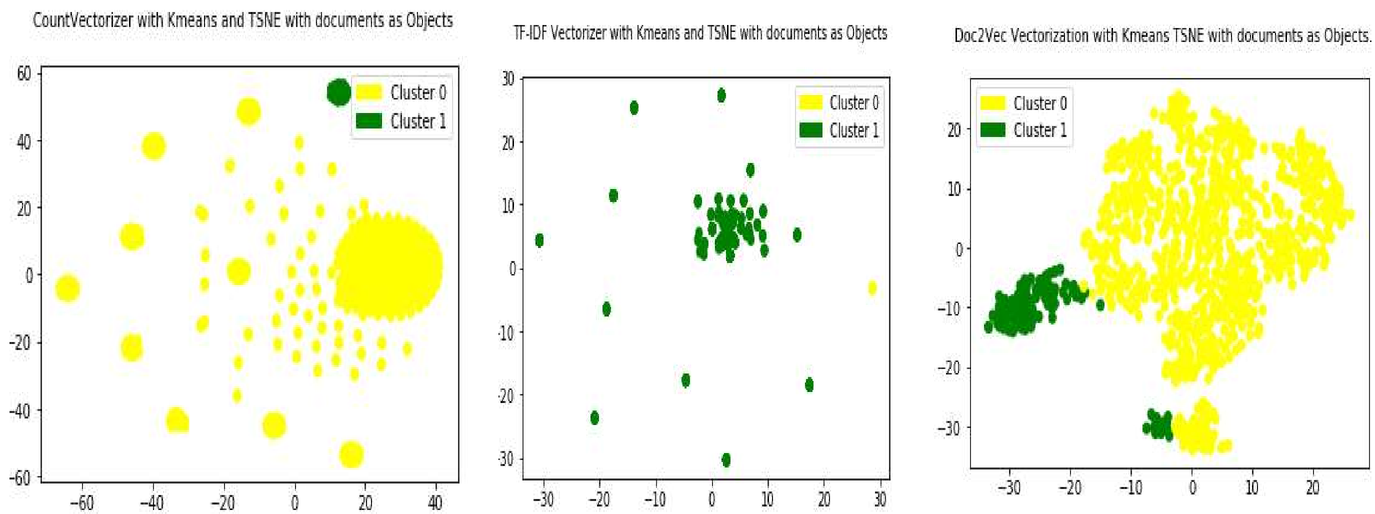
Table 4: Visual description of the 3 approaches with k-means with documents as Objects



We can see from the visual examples that the Doc2Vec algorithm had the best document cluster distribution. The list of top terms did not seem to match with the 3 approaches, although CountVectorizer and Tf-IDF seem to have some common documents. This seems to match the results in the previous assignment where we had similar terms for both Approaches 1 and 2.

The next step involved applying t-distributed stochastic neighbor embedding (t-SNE) for the multidimensional scaling and plotting the results as shown below:

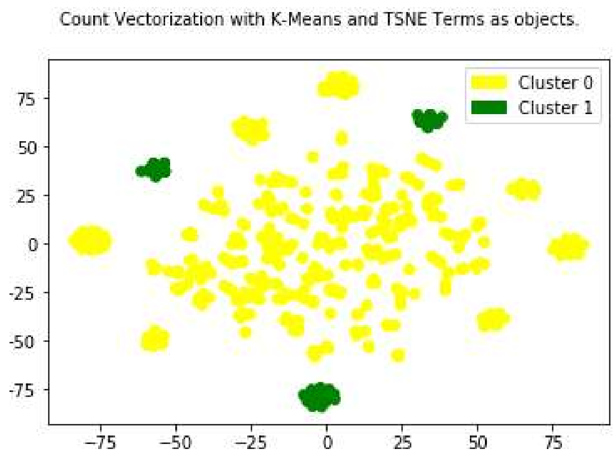
Table 5: Visual description of the 3 approaches with k-means and t-SNE



With t-SNE being applied to all three approaches, we can see that Doc2Vec has the best results, as was the case without the multidimensional scaling. I would say applying apply t-SNE might have exacerbated the situation, since the cluster zero seems to have gotten bigger.

I then took the matrices from Approaches one and two, and I applied t-SNE with the terms as objects, instead of the documents being the objects. Applying the terms as objects, confirmed what we have seen previously that there are larger number of cluster zeros than there are cluster ones. I do prefer this approach because the top terms for cluster zero and cluster one, was what I was expecting and also used that to build the ontology for autonomous vehicle safety. The previous assignment, also has similar results from the manually built terms, with slight different terms between approach one and two. If I was to pick the best result among the two approaches, with the top terms produced for each cluster, I would pick approach one. The reason is approach two included the terms “tierney” and “empty”.

Table 6: Visual description of the 2 approaches with k-means and t-SNE with terms as objects



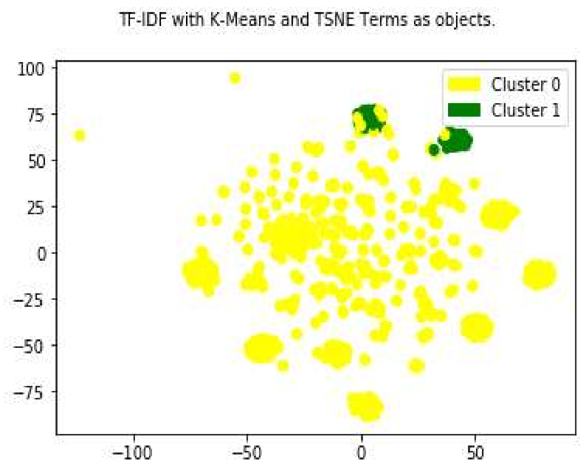


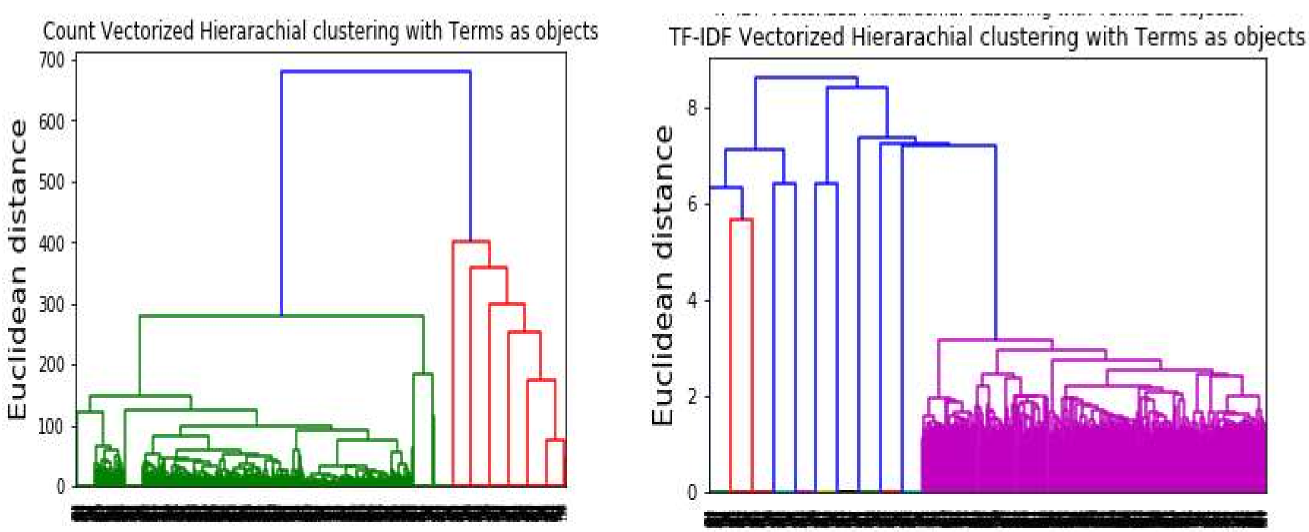
Table 7: Top terms of the 2 approaches with k-means clusters and t-SNE with terms as objects

```
Top Terms for CountVectorizer Kmeans per cluster:
Cluster 0:
 vehicle
 selfdriving
 company
 autonomous
 technology
 testing
 safety
 public
 driver
 waymo
Cluster 1:
 vehicle
 safety
 selfdriving
 technology
 autonomous
 crash
 automated
 driver
 human
 company
```

```
Top Terms per TF_IDF Kmeans cluster:
Cluster 0:
 vehicle
 selfdriving
 autonomous
 company
 safety
 waymo
 technology
 testing
 driver
 human
Cluster 1:
 parking
 space
 garage
 empty
 people
 development
 urban
 planning
 tierney
 vehicle
```

In order to understand the corpus using the terms as objects, I performed a hierarchical cluster analysis on the matrices for approaches one and two. I preferred the hierarchical cluster from approach one, since it showed a clear demarcation between the clusters, but if I wanted to build an ontology using hierarchical cluster analysis, I would go with approach two, since it seemed to give more information on the topic and subtopic breakdown as shown in Table 8.

Table 7: Hierarchical cluster analysis of the 2 approaches with terms as objects



I applied the latent Dirichlet allocation to do some topic modeling, on my way to understanding the corpus as a whole. I applied this algorithms to the two approaches namely: 1) CountVectorizer and 2) TF-IDF, and I was able to see that approach one seems to have better results in my opinion. The reason for this, is that some of the terms produced by the TF-IDF algorithms seems to have concatenated words together.

Table 8: Approach 1 LDA of the CounterVectorizer approaches with terms as objects

```
CountVectorizer + LatentDirichletAllocation
Topic 0:
would software vehicle engineer people company could selfdriving going potential
Topic 1:
space parking people vehicle design city public house urban transportation
Topic 2:
vehicle safety technology automated autonomous transportation driver crash human fully
Topic 3:
driver system tesla electric company vehicle would thing around might
Topic 4:
selfdriving vehicle safety driving technology company people human level would
Topic 5:
company selfdriving software project would engineer year could google might
Topic 6:
waymo program selfdriving around number service company system without human
Topic 7:
federal people engineer design road electric street might percent safer
Topic 8:
vehicle autonomous selfdriving company testing mile california street safety arizona
Topic 9:
selfdriving company waymo autonomous vehicle google sensor service lidar first
```

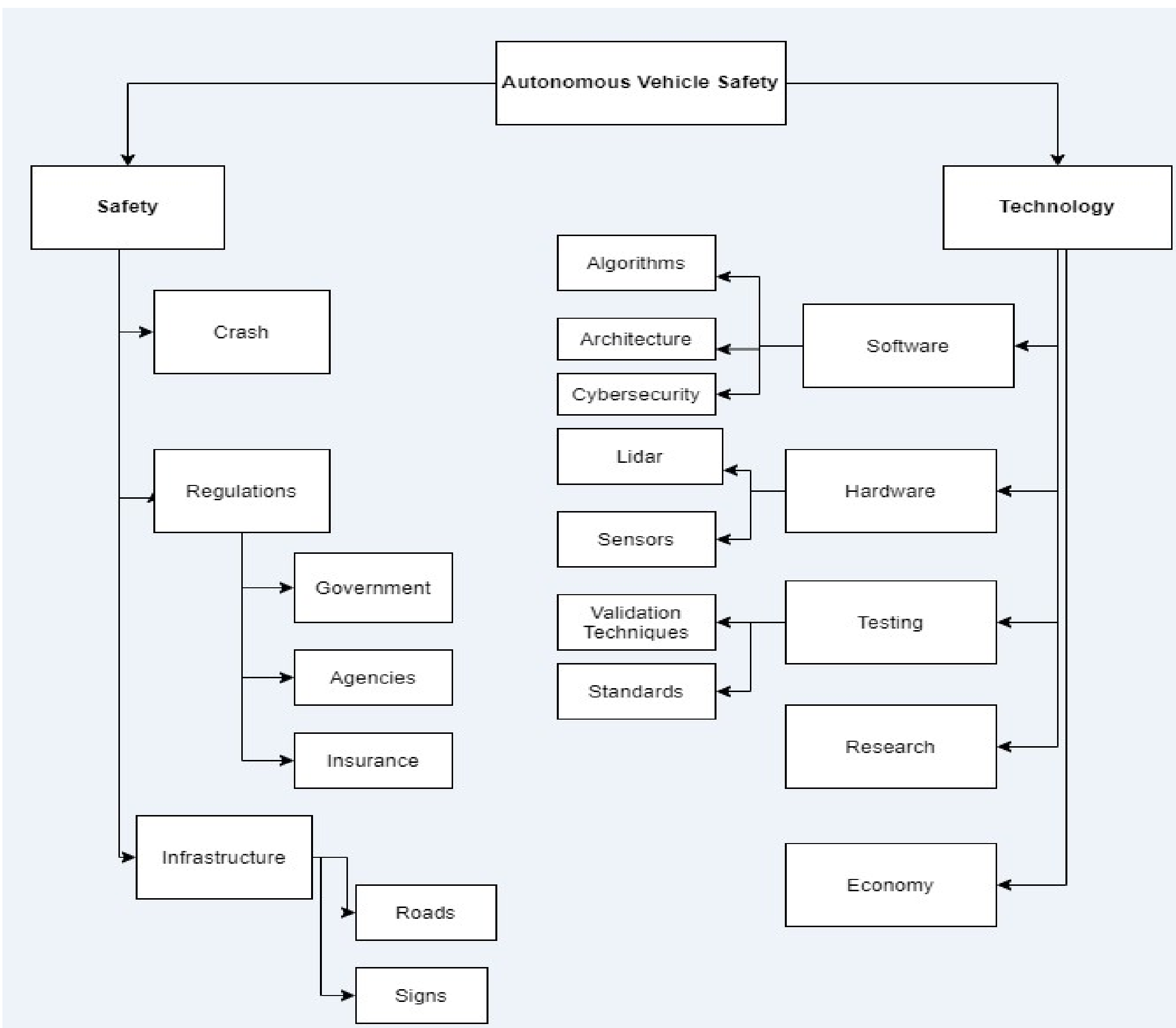
Table 9: Approach 2 LDA of the TF-IDF approaches with terms as objects

```
Tf-IDF + LatentDirichletAllocation
Topic 0:
vehicle lidar selfdriving safety driver luminar quartermile fuelcell university exposed
Topic 1:
disengagement vehicle mile driver drove testing selfdriving company autonomous registered
Topic 2:
vehicle automated safety selfdriving human waymo driver technology crash tesla
Topic 3:
vehicle selfdriving autonomous company safety technology waymo testing driver public
Topic 4:
concept driverless future model vehicle technology screen automaker towards integrating
Topic 5:
vehicle noticed competes garage wearable warning selfdriving novelizing tesla stone
Topic 6:
upcoming ioniq familyfocused form dispatcher shifted canceling displayed eliza mcclintock
Topic 7:
chaser hackett company neverthe wavering autonomous accused waymo stroke forgettable
Topic 8:
parking people space salesky could viability vehicle daylong needed profession
Topic 9:
cameraequipped waymoapproved fundjust commenting rejigger obtaining provide fumble expired levittowns
```

Results

Overall, I would say I prefer the algorithms with the terms as objects rather than the documents as objects, since it looked like they produced better results overall. The one exception would be the Doc2Vec without the t-SNE multidimensional scaling. It produced results in line with what I was expecting. The hierarchical clustering analysis, along with the topic modeling LDA algorithm helped convince me that the Ontology below is along the lines of what I was expecting when I saw the results of the cluster analysis and the top terms it generated.

Table 10: Autonomous vehicle safety ontology derived from the research above.



Conclusions

Autonomous vehicle technology is not a matter of if it happens, but when. This technology would evolve faster if safety is adopted and assessed at all stages. There are a lot of articles suggesting that safety is at the primary concern from developers, to the consumers, to the regulators. This research assignment's ability to classify the articles on autonomous vehicle safety into either "technology" or "safety" related subtopics would help the organization's various departments focus in on their interests quickly, without having to wade through ever growing number of articles that are out there on the topic. With an understanding of the corpus as whole after applying the cluster analysis, multidimensional scaling, topic modeling, we are able to build an ontology that will help us make sense of the current and well as future data. With all the other ontologies being developed (Zhao, Lihua et al) and (Bagschik, Gerrit), it is only a mater of time before we start to get the whole picture. This would help develop the industry along with the safety measures needs to bring this technology to market quickly.

Works Cited

Lori, Alessandro. (2020). Are Self-Driving Cars Safe? Retrieved from https://www.verizonconnect.com/resources/article/are-self-driving-cars-safe/

Allssa, Walker. (2020). Are self-driving cars safe for our cities? Retrieved from https://www.curbed.com/2016/9/21/12991696/driverless-cars-safety-pros-cons

Autonomous Vehicles. Retrieved from https://www.ghsa.org/issues/autonomous-vehicles

Kalra, N., Paddock, Susan M. (2016). Driving to Safety- How many Miles of Driving Would It Take to Demonstrate Autonomous Vehicle Reliability. Retrieved from https://www.rand.org/pubs/research_reports/RR1478.html

Allssa, Walker. (2018). It's time to delete Uber from our cities. Retrieved from https://www.curbed.com/transportation/2018/3/23/17153200/delete-uber-cities

Zhao, Lihua et al. Core Ontologies for Safe Autonomous Driving. Retrieved from http://ceur-ws.org/Vol-1486/paper_9.pdf

Bagschik, Gerrit et al. Ontology based Scene Creation for the Development of Automated Vehicles. Retrieved from https://arxiv.org/pdf/1704.01006.pdf

Code output from Spyder Editor (results can vary from the Jupyter notebook)

```
#!/usr/bin/env python
# coding: utf-8
import multiprocessing
import re,string
from nltk.corpus import stopwords
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
import scipy.cluster.hierarchy as shc
import json
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
import pandas as pd
from gensim.models.doc2vec import Doc2Vec, TaggedDocument
from sklearn.cluster import KMeans
from sklearn.manifold import TSNE
from sklearn.decomposition import PCA
from sklearn.decomposition import LatentDirichletAllocation
from nltk.stem import WordNetLemmatizer
#Functionality to turn stemming on or off
STEMMING = True  # judgment call, parsed documents more readable if False
MAX_NGRAM_LENGTH = 1  # try 1 and 2 and see which yields better modeling results
VECTOR_LENGTH = 100  # set vector length for TF-IDF and Doc2Vec
DISPLAYMAX = 10 # Dispaly count for head() or tail() sorted values
DROP_STOPWORDS = False
SET_RANDOM = 9999
NUMBER_OF_CLUSTERS = 2
##############################################################################
#  Number of cpu cores
##############################################################################
cores = multiprocessing.cpu_count()
```

```
print("\nNumber of processor cores:", cores)
################################################################################
#  Create document labels
################################################################################
def create_label(text):
    #print(text)
    text = text.replace('.html',")
    #print(text)
    text = text.replace('.htm',")
    regex = re.compile('[^a-zA-Z]')
    regex.sub(", text)
    return text
cluster_dict = {
    0:"safety",
    1:"technology"
}
def createCategory(text):
    if '-safe-' in text:
        return 'safety'
    return 'technology'
#################################################################################
#
### Function to process documents
#################################################################################
#
def clean_doc(doc):
    # split document into individual words
    tokens=doc.split()
    re_punc = re.compile('[%s]' % re.escape(string.punctuation))
    # remove punctuation from each word
    tokens = [re_punc.sub(", w) for w in tokens]
```

```
    # remove remaining tokens that are not alphabetic
    tokens = [word for word in tokens if word.isalpha()]
    # # filter out short tokens
    tokens = [word for word in tokens if len(word) > 4]
    # #lowercase all words
    tokens = [word.lower() for word in tokens]
    # # filter out stop words
    stop_words = set(stopwords.words('english'))
    tokens = [w for w in tokens if not w in stop_words]
    # # word stemming Commented
    if STEMMING:
        lem = WordNetLemmatizer()
        tokens = [lem.lemmatize(token) for token in tokens]
    return tokens
documents=[]
text_body=[]
text_titles = []
categories = []
regex = re.compile('[^a-zA-Z]')
with open('autonomous_vehicles_safety_corpus.jl') as json_file:
    data = json.load(json_file)
    for p in data:
        text_body.append(p['BODY'])
        text_titles.append(p['TITLE'][0:8])
        documents.append(create_label(p['FILENAME']))
        categories.append(createCategory(p['FILENAME']))
################################
# Final Processed File
############################
#empty list to store processed documents
processed_text=[]
```

```
#for loop to process the text to the processed_text list
for i in text_body:
    text=clean_doc(i)
    processed_text.append(text)

#stitch back together individual words to reform body of text
final_processed_text=[]

for i in processed_text:
    temp_DSI=i[0]
    for k in range(1,len(i)):
        temp_DSI=temp_DSI+' '+i[k]
    final_processed_text.append(temp_DSI)


# <h3>(2) Using matrices for Approaches 1, 2, and 3, perform partitioned cluster analysis (K-
means)
# with documents as objects. Utilize objective methods for determining the number of clusters K.
# Prepare summary lists of documents within each cluster. If there is a large number of
documents,
# show a sample of the documents in lists. Describe the results.</h3>

# <h5>(2) Approach 1 - CountVectorizer with Kmeans with documents as Objects</h5>

################################################################################
#######################
### Count Vectorization
################################################################################
#######################
print('\n\t\tCountVectorizer + Kmeans with documents as Objects')
```

```
count_vectorizer2A1 = CountVectorizer(ngram_range = (1, MAX_NGRAM_LENGTH),
max_features = VECTOR_LENGTH)
count_vectors_matrix2A1 = count_vectorizer2A1.fit_transform(documents)

kmCV2A1 = KMeans(n_clusters=NUMBER_OF_CLUSTERS, random_state=89)
kmCV2A1.fit(count_vectors_matrix2A1)
clustersCv2A1 = kmCV2A1.labels_.tolist()

y1 = clustersCv2A1
X1 = count_vectors_matrix2A1

termsCv2A1 = count_vectorizer2A1.get_feature_names()
x = count_vectors_matrix2A1.todense()

labels2A2 = kmCV2A1.fit_predict(count_vectors_matrix2A1)

reduced_data = PCA().fit_transform(x)

green_patch = mpatches.Patch(color="#FFFF00", label='Cluster 0')
yellow_patch = mpatches.Patch(color='#008000', label='Cluster 1')

fig = plt.figure()
fig.suptitle('CountVectorizer with Kmeans with documents as Objects', fontsize=10)
cluster_colors = ["#FFFF00", "#008000"]
color = [cluster_colors[i] for i in labels2A2]
plt.scatter(reduced_data[:, 0], reduced_data[:, 1], c=color)
plt.legend(handles=[green_patch, yellow_patch])
plt.show()

print("2A1 CountVectorizer + Kmeans with documents as Objects Top documents per cluster:")
order_centroids2A1 = kmCV2A1.cluster_centers_.argsort()[:, ::-1]
```

```
for i in range(NUMBER_OF_CLUSTERS):
    top_ten_words2A1 = [termsCv2A1[ind] for ind in order_centroids2A1[i, :5]]
    print("Cluster {}: {}".format(i, ' '.join(top_ten_words2A1)))

# <h5>(2) Approach 2 - TF-IDF Vectorization with Kmeans with documents as Objects</h5>
print('\nTF-IDF Vectorization with Kmeans with documents as Objects. . .')
tfidf2A2 = TfidfVectorizer(ngram_range=(1,1), min_df=0.0025)
tfidf2A2_matrix = tfidf2A2.fit_transform(documents)
'''create k-means model with custom config '''
km2A2 = KMeans(n_clusters=NUMBER_OF_CLUSTERS, random_state=89,
                        precompute_distances="auto",n_jobs=-1)

labels2A2 = km2A2.fit_predict(tfidf2A2_matrix)
x = tfidf2A2_matrix.todense()
terms2A2 = tfidf2A2.get_feature_names()
reduced_data = PCA().fit_transform(x)
green_patch = mpatches.Patch(color="#FFFF00", label='Cluster 0')
yellow_patch = mpatches.Patch(color='#008000', label='Cluster 1')

fig = plt.figure()
fig.suptitle('TF-IDF Vectorization with Kmeans with documents as Objects', fontsize=10)
cluster_colors = ["#FFFF00", "#008000"]
color = [cluster_colors[i] for i in labels2A2]
plt.scatter(reduced_data[:, 0], reduced_data[:, 1], c=color)
plt.legend(handles=[green_patch, yellow_patch])
plt.show()

print("2A2 TF-IDF Vectorization with Kmeans with documents as Objects Top documents per
cluster:")
order_centroids2A2 = km2A2.cluster_centers_.argsort()[:, ::-1]
```

```
for i in range(NUMBER_OF_CLUSTERS):
    top_ten_words2A2 = [terms2A2[ind] for ind in order_centroids2A2[i, :5]]
    print("Cluster {}: {}".format(i, ' '.join(top_ten_words2A2)))



# <h5>(2) Approach 3 - Doc2Vec Vectorization with Kmeans with documents as Objects</h5>

# train_corpus using TaggedDocument
train_corpus = [TaggedDocument(doc, [i]) for i, doc in enumerate(documents)]

print("\n\t\tDoc2Vec Vectorization with Kmeans with documents as Objects")

# Doc2Vec Vectorization
doc2vec2A3_model = Doc2Vec(vector_size = 50, window = 4, min_count = 2, workers = cores,
epochs = 40)
doc2vec2A3_model.build_vocab(train_corpus)
doc2vec2A3_model.train(train_corpus, total_examples = doc2vec2A3_model.corpus_count,
epochs = doc2vec2A3_model.epochs)
kmeans_model = KMeans(n_clusters=2, init='k-means++', max_iter=100)
X = kmeans_model.fit(doc2vec2A3_model.docvecs.vectors_docs)
labels2A3=kmeans_model.labels_.tolist()

datapoint = doc2vec2A3_model.docvecs.vectors_docs

green_patch = mpatches.Patch(color="#FFFF00", label='Cluster 0')
yellow_patch = mpatches.Patch(color='#008000', label='Cluster 1')



print("Doc2vec Vectorization with Kmeans with documents as Objects.")
fig = plt.figure(figsize =(8, 8))
```

```
fig.suptitle('Doc2vec Vectorization with Kmeans with documents as Objects clusters',
fontsize=10)
cluster_colors = ["#FFFF00", "#008000"]
color = [cluster_colors[i] for i in labels2A3]
plt.scatter(datapoint[:, 0], datapoint[:, 1], c=color)
plt.legend(handles=[green_patch, yellow_patch])
plt.show()
print("2A3 Doc2vec with Kmeans with documents as Objects Top documents per cluster:")
order_centroids2A3 = kmeans_model.cluster_centers_.argsort()[:, ::-1]

for i in range(NUMBER_OF_CLUSTERS):
    top_ten_words2A3 = [documents[ind] for ind in order_centroids2A3[i, :5]]
    print("Cluster {}: {}".format(i, ' '.join(top_ten_words2A3)))
# <h3>(3) Using matrices for Approaches 1, 2, and 3, perform multidimensional scaling with
documents
# as objects. Visualize the multidimensional scaling solutions in two-space, labeling points
# with document names. Identify clusters from the K-means clustering with colored points,
providing
# a legend on the visualization. Use t-distributed stochastic neighbor embedding (t-SNE) for the
multidimensional scaling.
# If there is a large number of documents, plot a sample of the documents. Describe the results.
</h3>

# <h5>(3) Approach 1 - CountVectorizer with Kmeans + TSNE with documents as
Objects</h5>
print('\n\t\tCountVectorizer + Kmeans + TSNE with documents as Objects')
count_vectorizer3A1 = CountVectorizer(ngram_range = (1, MAX_NGRAM_LENGTH),
max_features = VECTOR_LENGTH)
count_vectors_matrix3A1 = count_vectorizer2A1.fit_transform(documents)

kmCV3A1 = KMeans(n_clusters=NUMBER_OF_CLUSTERS, random_state=89)
```

```
kmCV3A1.fit(count_vectors_matrix2A1)
clustersCv3A1 = kmCV2A1.labels_.tolist()
centroids = kmCV3A1.cluster_centers_
print('\nTSNE of CountVectorizer  + Kmeans ')
tsne_perplexity = 20.0
tsne_early_exaggeration = 4.0
tsne_learning_rate = 1000
random_state = 1
model =  TSNE(n_components=2, verbose=1, perplexity=2.0, n_iter=1000)

transformed_centroids = model.fit_transform(count_vectors_matrix3A1)

green_patch = mpatches.Patch(color="#FFFF00", label='Cluster 0')
yellow_patch = mpatches.Patch(color='#008000', label='Cluster 1')

cluster_colors = ["#FFFF00", "#008000"]
color = [cluster_colors[i] for i in clustersCv3A1]
fig = plt.figure(figsize =(8, 8))
fig.suptitle('CountVectorizer with Kmeans and TSNE with documents as Objects', fontsize=10)
plt.scatter(transformed_centroids[:, 0], transformed_centroids[:, 1], c=color)
plt.legend(handles=[green_patch, yellow_patch])
plt.show()
# <h5>(3) Approach 2 -  TF-IDF Vectorizer with Kmeans + TSNE with documents as
Objects</h5>
tfidf3A2 = TfidfVectorizer(ngram_range=(1,1), min_df=0.0025)
tfidf3A2_matrix = tfidf2A2.fit_transform(documents)

kmeans_model = KMeans(n_clusters=2, init='k-means++', max_iter=100)
X = kmeans_model.fit(tfidf3A2_matrix)
labels3A2=kmeans_model.labels_.tolist()
```

```
tsne = TSNE(n_components=2, verbose=1, perplexity=40, n_iter=1000)
tsne_results = tsne.fit_transform(tfidf2A2_matrix)

green_patch = mpatches.Patch(color="#FFFF00", label='Cluster 0')
yellow_patch = mpatches.Patch(color='#008000', label='Cluster 1')

print("TF-IDF Vectorizer with Kmeans and TSNE with documents as Objects")
fig = plt.figure(figsize =(8, 8))
fig.suptitle('TF-IDF Vectorizer with Kmeans and TSNE with documents as Objects',
fontsize=10)
cluster_colors = ["#FFFF00", "#008000"]
color = [cluster_colors[i] for i in labels3A2]
plt.scatter(tsne_results[:, 0], tsne_results[:, 1], c=color)
plt.legend(handles=[green_patch, yellow_patch])
plt.show()
# <h5>(3) Approach 3 -  Doc2Vec Vectorizer with Kmeans + TSNE with documents as
Objects</h5>
import matplotlib.patches as mpatches
kmeans_model = KMeans(n_clusters=2, init='k-means++', max_iter=100)
X = kmeans_model.fit(doc2vec2A3_model.docvecs.vectors_docs)
labels2A3=kmeans_model.labels_.tolist()

tsne = TSNE(n_components=2, verbose=1, perplexity=40, n_iter=1000)
datapoint = tsne.fit_transform(doc2vec2A3_model.docvecs.vectors_docs)

print("Doc2Vec Vectorization with Kmeans TSNE with documents as Objects.")
plt.figure
cluster_colors = ["#FFFF00", "#008000"]

green_patch = mpatches.Patch(color="#FFFF00", label='Cluster 0')
```

```
yellow_patch = mpatches.Patch(color='#008000', label='Cluster 1')
fig = plt.figure(figsize =(8, 8))
fig.suptitle('Doc2Vec Vectorization with Kmeans TSNE with documents as Objects.',
fontsize=10)
color = [cluster_colors[i] for i in labels2A3]
plt.scatter(datapoint[:, 0], datapoint[:, 1], c=color)
plt.legend(handles=[green_patch, yellow_patch])
plt.show()
# <h3>(5) Using matrices for Approaches 1 and 2, perform multidimensional scaling (t-SNE)
with terms as objects.
# Visualize the multidimensional scaling solutions in
# two-space, labeling points as terms. Describe the results.</h3>

# <h5>Approach 1: Count Vectorization with K-Means + TSNE Terms as objects</h5>
# Count Vectorization
print('\n\t\tCount Vectorization with K-Means + TSNE Terms as objects')

count_vectorizer = CountVectorizer(ngram_range = (1, MAX_NGRAM_LENGTH),
max_features = VECTOR_LENGTH)
count_vectors_matrix = count_vectorizer.fit_transform(final_processed_text)
kmCV = KMeans(n_clusters=2, random_state=89)
kmCV.fit(count_vectors_matrix)
clustersCv = kmCV.labels_.tolist()

y1 = clustersCv
X1 = count_vectors_matrix

termsCv = count_vectorizer.get_feature_names()

cv_dictionary = {'FileName':documents, 'Cluster':clustersCv,  'Text': final_processed_text}
cv_df = pd.DataFrame(cv_dictionary, columns=['Cluster', 'FileName','Text'])
```

```
cv_df['Category'] = ((cv_df.Cluster)).map(cluster_dict)

cv_df.head(5)
print("Top Terms for CountVectorizer Kmeans per cluster:")

#sort cluster centers by proximity to centroid
order_centroidsCV = kmCV.cluster_centers_.argsort()[:, ::-1]

terms_dict1 = []
cluster_terms1 = {}
cluster_title1 = {}
for i in range(2):
    print("Cluster %d:" % i),
    temp_terms1 = []
    temp_titles1 = []
    for ind in order_centroidsCV[i, :10]:
        print(' %s' % termsCv[ind])
        terms_dict1.append(termsCv[ind])
        temp_terms1.append(termsCv[ind])
    cluster_terms1[i] = temp_terms1
print('\nTSNE of CountVectorizer  + Kmeans ')
tsne_perplexity = 20.0
tsne_early_exaggeration = 4.0
tsne_learning_rate = 1000
random_state = 1
model =  TSNE(n_components=2, verbose=1, perplexity=2.0, n_iter=1000)

cv_transformed_centroids = model.fit_transform(count_vectors_matrix)

green_patch = mpatches.Patch(color="#FFFF00", label='Cluster 0')
yellow_patch = mpatches.Patch(color='#008000', label='Cluster 1')
```

```
cluster_colors = ["#FFFF00", "#008000"]
color = [cluster_colors[i] for i in clustersCv]
fig = plt.figure(figsize =(8, 8))
fig.suptitle('Count Vectorization with K-Means and TSNE Terms as objects.', fontsize=10)
plt.scatter(cv_transformed_centroids[:, 0], cv_transformed_centroids[:, 1], c=color)
plt.legend(handles=[green_patch, yellow_patch])
plt.show()
# <h4>Approach 2: TF-IDF Vectorization with K-Means + TSNE Terms as objects</h4>
################################
### TF-IDF Vectorization
################################
# run tfidf (prevalent - require 25% of docs)

print('\nTF-IDF Vectorization K Means vectorization. . .')

tfidf1 = TfidfVectorizer(ngram_range=(1,1))
tfidf1_matrix = tfidf1.fit_transform(final_processed_text)

k=2
km2 = KMeans(n_clusters=k, random_state=89)
km2.fit(tfidf1_matrix)
clusters1 = km2.labels_.tolist()
terms2 = tfidf1.get_feature_names()

tf1_dictionary = {'FileName':documents, 'Cluster':clusters1,  'Text': final_processed_text}
tf1_df = pd.DataFrame(tf1_dictionary, columns=['Cluster', 'FileName','Text'])
tf1_df['Category'] = ((tf1_df.Cluster)).map(cluster_dict)

tf1_df.tail(5)
print("Top Terms per TF_IDF Kmeans cluster:")
```

```
#sort cluster centers by proximity to centroid
order_centroids2 = km2.cluster_centers_.argsort()[:, ::-1]

terms_dict2 = []
cluster_terms2 = {}
cluster_title2 = {}

for i in range(k):
    print("Cluster %d:" % i),
    temp_terms2 = []
    temp_titles2 = []
    for ind in order_centroids2[i, :10]:
        print(' %s' % terms2[ind])
        terms_dict2.append(terms2[ind])
        temp_terms2.append(terms2[ind])
    cluster_terms2[i] = temp_terms2
print('\nTSNE of TF-IDF Vectorizer  + Kmeans ')
tsne_perplexity = 20.0
tsne_early_exaggeration = 4.0
tsne_learning_rate = 1000
random_state = 1
model =  TSNE(n_components=2, verbose=1, perplexity=2.0, n_iter=1000)

tf_idf_transformed_centroids = model.fit_transform(tfidf1_matrix)

green_patch = mpatches.Patch(color="#FFFF00", label='Cluster 0')
yellow_patch = mpatches.Patch(color='#008000', label='Cluster 1')

cluster_colors = ["#FFFF00", "#008000"]
color = [cluster_colors[i] for i in clusters1]
```

```
fig = plt.figure(figsize =(8, 8))
fig.suptitle('TF-IDF with K-Means and TSNE Terms as objects.', fontsize=10)
plt.scatter(tf_idf_transformed_centroids[:, 0], tf_idf_transformed_centroids[:, 1], c=color)
plt.legend(handles=[green_patch, yellow_patch])
plt.show()
# <h5>(6) Using matrices for Approaches 1 and 2, perform hierarchical cluster analysis with
terms as objects.
# Visualize the clustering solution as a tree diagram, with terminal nodes labeled as terms.
Describe the results.</h5>
# <h5>Approach 1: Count Vectorized Hierarachial clustering with Terms as objects</h5>
X = count_vectors_matrix.todense()
fig = plt.figure(figsize =(8, 8))
fig.suptitle('Count Vectorized Hierarachial clustering with Terms as objects.', fontsize=10)
plt.title('Count Vectorized Hierarachial clustering with Terms as objects')
plt.ylabel('Euclidean distance', fontsize=16)
Dendrogram = shc.dendrogram((shc.linkage(X, method ='ward')))
# <h5>Approach 2: TF-IDF Vectorized Hierarachial clustering with Terms as objects</h5>
X = tfidf1_matrix.todense()
fig = plt.figure(figsize =(8, 8))
fig.suptitle('TF-IDF Vectorized Hierarachial clustering with Terms as objects.', fontsize=10)
plt.title('TF-IDF Vectorized Hierarachial clustering with Terms as objects')
plt.ylabel('Euclidean distance', fontsize=16)
Dendrogram = shc.dendrogram((shc.linkage(X, method ='ward')))
#print(Dendrogram)

# <h5>(7) Compare multidimensional scaling and clustering results for terms across Approaches
1 and 2.
# What do these analyses tell you about the corpus? In your opinion, which of the two
approaches provides
# the most clear-cut (interpretable) results?</h5>
```

```
# <h3>(9, optional) Try a topic modeling solution such as latent Dirichlet allocation to identify
documents with topics:
#     https://scikit-
learn.org/stable/modules/generated/sklearn.decomposition.LatentDirichletAllocation.html
# </h3>
def display_topics(model, feature_names, no_top_words):
   for topic_idx, topic in enumerate(model.components_):
      print ("Topic %d:" % (topic_idx))
      print (" ".join([feature_names[i]
                  for i in topic.argsort()[:-no_top_words - 1:-1]]))



# <h4>CountVectorizer +  LatentDirichletAllocation </h4>
cv_lda = LatentDirichletAllocation(max_iter=20, learning_method='online',
learning_offset=50.,random_state=89).fit(count_vectors_matrix)
print("\n\n\tCountVectorizer + LatentDirichletAllocation ")
no_top_words = 10
cv_feature_names = count_vectorizer.get_feature_names()
display_topics(cv_lda, cv_feature_names, no_top_words)
print('CountVectorizer seems to have produced better results than the Tf-IDF LDA example')
# <h4>Tf-IDF +  LatentDirichletAllocation </h4>
tf_idf_lda = LatentDirichletAllocation(max_iter=20, learning_method='online',
learning_offset=50.,random_state=89).fit(tfidf1_matrix)

print("\n\n\tTf-IDF + LatentDirichletAllocation ")
no_top_words = 10
tf_feature_names = tfidf1.get_feature_names()
display_topics(tf_idf_lda, tf_feature_names, no_top_words
```