Clustering Safety Observations based on Text Features

Clustering Safety Observations using Natural Language Learning Process to validate the manual labelling of the dataset.

An assignment submitted in fulfillment of the requirements of IBM Introduction to Machine Learning Specialization

By

Minhaj Ahmed Ansari

1. Introduction

1.1. The objective of this program would be to develop a Machine Learning Model which would be able to cluster the EHS Observations based on the historical data and return the optimal numbers of clusters which we will use to guide or further correct our manual labeling process.

1.2. A root cause analysis refers to the process of identifying the underlying cause of an issue or incident. Whether you're investigating an employee who crushed their foot when they dropped a heavy box or one who tripped over a cord in a poorly lit office, an RCA goes beyond simply removing the immediate cause of the accident

1.3. By implication this exercise is a Natural Language Processing Exercise, as we will be extracting features from text (Observation Description) and using those for training our clustering model.

2. Brief description of the data set and a summary of its attributes

2.1. The data which I am using here is a dump from the Safety Observation System of a Major Company. Each row of data consists of the observation description along with pre-selected category and root cause (Classes). These labeling have been done manually. Thus, we need to drop these labels first to proceed with Clustering.

2.2. We will employ various dimensionality reduction and feature extraction methods which would form the basis of our exercise. By implication this is a Natural Language Learning Process.

2.3. The number of columns in the dataset is 3 and the no of rows is 1028493. The target variable here will be 'Sub-Category' and the input variable will be 'Observation'. Another column is 'UA/UC' which broadly categorizes observations into different types of unsafe acts and conditions.

2.4. The plan includes

1. Cleaning the data by removing stops and performing EDA.
2. Feature extraction to arrive on most important features.
3. Feature selection and dimension reduction using Non-Negative Matrix Factorization.
4. Training our different models and measure the performance using pre-identified class labels
5. Evaluation of Models and selection of best model.
6. Summary and Future action.

3. Data Cleaning and Wrangling

3.1. **Plan**

- Bring all words on same case (lower case).
- Detect the missing values and treat them accordingly.
- Drop various Stop words which don't add any information to the model.

3.2. **Actions**

- Importing the necessary module for text preprocessing and feature extraction.

```
[4]: #Let us first bring in all the required modules

     from sklearn.feature_extraction.text import TfidfVectorizer
     from sklearn.preprocessing import LabelEncoder
     from nltk.corpus import stopwords
     from nltk.stem import WordNetLemmatizer
     from collections import Counter
```

- During this process it was identified that there are lot of null value columns in the entire datasets.

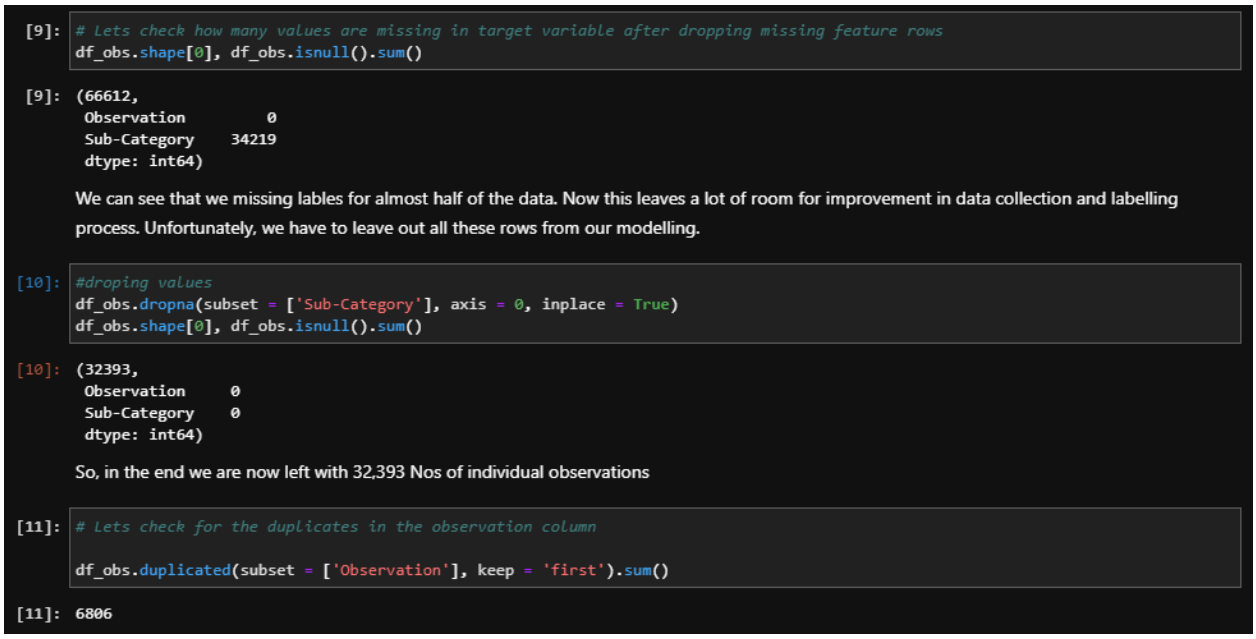
```
[5]: #Lets first drop the 'UA/UC' Column which is not required for this exercise.

     df_obs.drop('UA/UC', axis = 1, inplace = True)

[6]: df_obs.head()
```

[6]:

	Observation	Sub-Category
0	worker used drilling machine without green tag.	NaN
1	Worker used a plug top for multiple electrica...	No Plug top
2	worker use drilling machine at site without gr...	NaN
3	Site supervisor is not available at site whi...	NaN
4	worker use electric grinder without green tag.	No Inspection Done

```
[7]: # Lets check the missing values

     df_obs.isnull().sum()

[7]: Observation     961881
     Sub-Category    996100
     dtype: int64
```

- Also, there were lot of missing labels for almost half of the data. Now this leaves a lot of room for improvement in data collection and labelling process. Unfortunately, we must leave out all these rows from our modelling.



```
[9]: # Lets check how many values are missing in target variable after dropping missing feature rows
     df_obs.shape[0], df_obs.isnull().sum()

[9]: (66612,
      Observation         0
      Sub-Category    34219
      dtype: int64)
```

We can see that we missing lables for almost half of the data. Now this leaves a lot of room for improvement in data collection and labelling process. Unfortunately, we have to leave out all these rows from our modelling.

```
[10]: #droping values
      df_obs.dropna(subset = ['Sub-Category'], axis = 0, inplace = True)
      df_obs.shape[0], df_obs.isnull().sum()

[10]: (32393,
       Observation     0
       Sub-Category    0
       dtype: int64)
```

So, in the end we are now left with 32,393 Nos of individual observations

```
[11]: # Lets check for the duplicates in the observation column

      df_obs.duplicated(subset = ['Observation'], keep = 'first').sum()

[11]: 6806
```

- There were also lot of duplicate observations which may skew the results of the modelling.

```
[12]: df_obs.drop_duplicates(subset = 'Observation', keep = 'first', ignore_index = True, inplace = True)
```

- After all, above exercise, the number of columns in the dataset is 2 and the no of rows is 25587

3.3. **Summary**

- We dropped the null values and duplicated observations to reduce the dimensions of the data
- This also ensures that we avoid assigning higher weightage to certain duplicated observations at the time of feature extraction.
- We also dropped UA/UC columns as we want to explore this dataset purely from NLP perspective.

4. Exploratory Data Analysis

4.1. **Plan**

- Remove punctuation marks and any other extra space.
- Lemmatize all the words so that different words stemming from same root word are not counted as separate features.
- Extract features from our observation text using tf-idf method.
- Perform Dimensionality Reduction using NMF.

4.2. **Actions**

- Lemmatization is the process of grouping together the different inflected forms of a word so they can be analyzed as a single item. Lemmatization is similar to stemming but it brings context to the words. So, it links words with similar meaning to one word.
- For effective lemmatization, punctuations and other similar spaces are also removed.
- For lemmatization, the module which has been used is Word Net Lemmatizer under NLTK library.

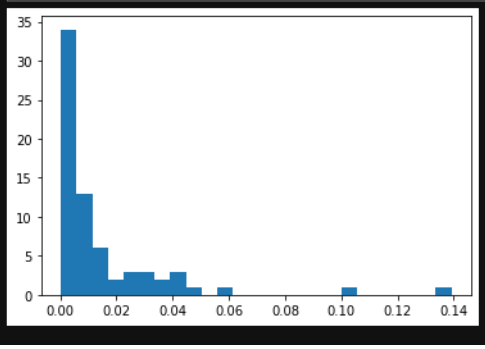
```
[17]: def lemmatize_text(word_list):
          lem_word_list = [wordlemmatize.lemmatize(word, pos = 'v') for word in word_list.split()]
          new_lem_word_list = [wordlemmatize.lemmatize(word) for word in lem_word_list]
          return " ".join(new_lem_word_list)

[18]: df_obs.Observation = df_obs.Observation.apply(lemmatize_text)

[19]: #Lets check our cleaned dataframe

      df_obs.sample(10)
```

[19]:

	Observation	Sub-Category
3847	Poor electrical joint in cable lay on grind	Improper Cable joints
3741	plug top not implement naked wire use to opera...	No Plug top
22650	improper housekeep be find in work area	Poor Housekeeping
22764	Wooden ladder be find to be use for access egr...	Improper ladder arrangement
20453	material keep on road without barricade	Hard Barricades not provided
19321	Electrical cable tray be use for access egress...	Unsafe/ Blocked Access & Egress
16366	contractor supervisor engage to it work withou...	Workmen engaged without screening/ Induction
15664	Enercom Workmen bhupendra kumar deploy directl...	Workmen engaged without screening/ Induction
23298	Trench pit we re find open condition and witho...	Openings not covered/ barricaded
6842	weld work be carry out without remove water fr...	Unsafe Electrical connections/Practices ...

- If you watch closely lemmatization has taken effect on many words for ex- against index 7778 the word 'been' has been stemmed to 'be', the word 'kept' has been changed back to 'keep' at 7656. These words would have been extracted as separate features thereby increasing the overall feature dimension

```
[21]: plt.hist(df_obs['Sub-Category'].value_counts(normalize = True), bins = 25,);
```



- This shows that our classes have quite severe imbalance. However, we are not going to use these labels for clustering, but we will use them to validate our results. Also, this gives an idea about what should be No. of Components that we should be targeting during the Non-Negative Matrix Factorization
- Next and probably most important step for modelling is to extract features from text data. Now there are many methods to achieve this objective but the most important from all is Term Frequency and Inverse Document Frequency Method.
- In information retrieval, tf–idf, TF*IDF, or TFIDF, short for term frequency–inverse document frequency, is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus. It is often used as a weighting factor in searches of information retrieval, text mining, and user modeling. The tf–idf value increases proportionally to the number of times a word appears in the document and is offset by the number of documents in the corpus that contain the word, which helps to adjust for the fact that some words appear more frequently in general. tf–idf is one of the most popular term-weighting schemes today.
- Another important term that you will come across would be N-grams. In the fields of computational linguistics and probability, an n-gram is a contiguous sequence of n items from a given sample of text or speech. The items can be phonemes, syllables, letters, words, or base pairs according to the application. The n-grams typically are collected from a text or speech corpus. When the items are words, n-grams may also be called shingles.
- Using Latin numerical prefixes, an n-gram of size 1 is referred to as a "unigram"; size 2 is a "bigram" (or, less commonly, a "digram"); size 3 is a "trigram". For example, a feature - 'Safety' is unigram whereas a feature 'Safety Helmet' would be bigram
- The words which are generally filtered out before processing a natural language are called stop words. These are the most common words in any language (like articles, prepositions, pronouns, conjunctions, etc.) and does not add much information to the text. Also, I have set min_df = 10, which means words that occurred in too few documents, in this case less than 10, should be filtered out.

```
[23]: # So lets instantiate our tfidfvectorizer by setting feature output for both unigrams and bigrams and Stop words from English
tfidf = TfidfVectorizer(ngram_range = (1,2),
                        stop_words = 'english',
                        min_df = 10,
                       )
```

- I would highly recommend the reader to go through sklearn's working with text data resources. https://scikit-learn.org/stable/tutorial/text_analytics/working_with_text_data.html
- Next, I have extracted features for further modelling using tfidfvectorizer. A Sample feature matrix is shown below.

```
[27]: features = tfidf.fit_transform(X_train, y_train).toarray()

      print(f'The feature matrix contains {features.shape[0]} observations with {features.shape[1]} features')

      The feature matrix contains 17910 observations with 2582 features

[28]: # Let us create the feature matrix for our future reference

      feature_matrix = pd.DataFrame(data = features, columns = tfidf.get_feature_names())
      feature_matrix.head()
```

[28]:

	aaa	absence	access	access approach	access area	access arrangement	access available	access block	access egress	access filter	...	workplace screen	wtp	xi	xi im	yadav	yadav engage	yard	zone
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

5 rows × 2582 columns

- The feature selection is done using the chi-square test. The Null hypothesis that there is no significance association between the input variable and target variable. While our alternate hypothesis is that there is significance association between the input variable and target variable. Consequently, we will select only those features which have p_value less that 0.05 (Significance level), so that we can reject our null hypothesis.
- Finally, there are 1834 statistically significant features.
- Now, it's time to reduce the dimensionality of the feature matrix so that the process of further modelling can be eased and improved
- First step of which would be to assign the number of the components that we may want to extract. Best policy in this case could be to use the pre-determined labels in our data which we separated as y variable. Let's check for the number of unique

classes in our y variable. The number of classes in our label variable is 70. Thus, I decomposed all my features into 70 features.

```
[63]: # Creating the nmf model

      # nmf_model = NMF(n_components = y.nunique(),
      #             init = 'random',
      #             random_state = 100)

[34]: # nmf_features = nmf_model.fit_transform(final_feature_matrix)

[35]: ## Save the model in pickled form for future reference

      def pickle_model(model, filename):
          pkl.dump(model, open(filename, 'wb'))

      def de_pickle(model):
          return pkl.load(open(model, 'rb'))

[39]: #pickle_model(nmf_features, 'nmf_features_pkl.sav')

[40]: nmf_features = de_pickle('nmf_features_pkl.sav')

[41]: nmf_features.shape

[41]: (25587, 70)
```

- At this point I also check for the predominant feature for each component.

```
[45]: # Lets check for each components identified which is the largest contributing featur

      column_1 = pd.Series(nmf_features_components_df.idxmax(axis = 1))
      column_2 = pd.Series(nmf_features_components_df.max(axis = 1))

      largest_component_nmf = pd.concat([column_1, column_2], axis = 1)
      largest_component_nmf.columns = ['Feature Names', 'Values']
      largest_component_nmf
```

[45]:

	Feature Names	Values
0	cable	0.450917
1	engage site	0.440872
2	soil	0.514036
3	place	0.513937
4	process	0.483933

4.3. **Summary**

- We realized that our data consists of highly imbalanced classes.
- Next step was to prepare the text data for further feature extraction. We applied various transformations such as lemmatization and achieving homogenization by converting all words into lower case.
- We dropped the pre-defined labels in the dataset and saved them.
- Next, we used tfidfvectorizer to extract features from our text data. The tf–idf value increases proportionally to the number of times a word appears in the document and is offset by the number of documents in the corpus that contain the word, which helps to adjust for the fact that some words appear more frequently in general.

- We performed chi2 test to extract the best associated features with p-value less than 0.05.
- Finally, we used NMF method to decompose our features into 70 components

5. **Model Selection and Evaluation**

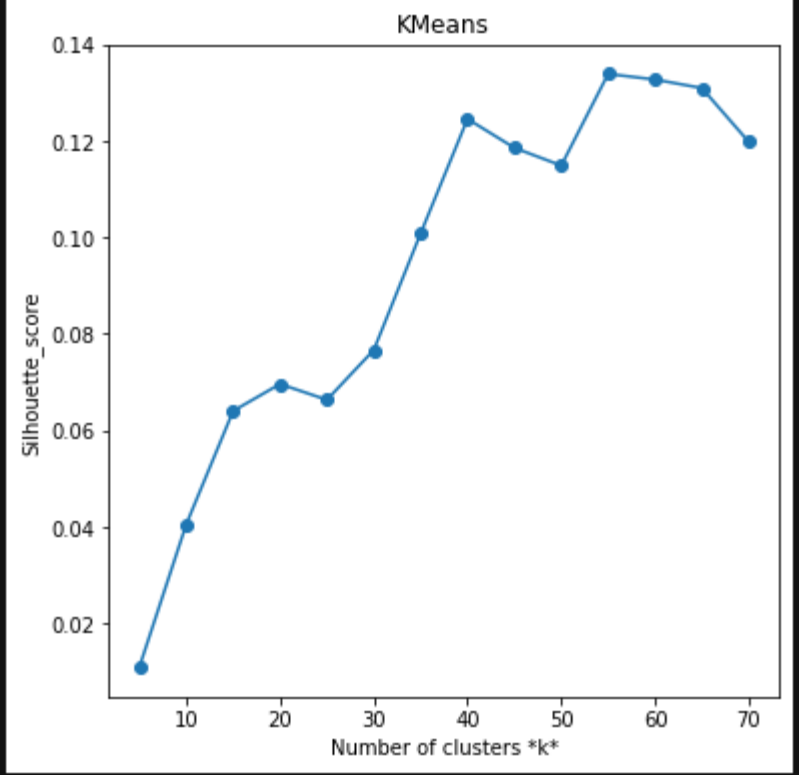
5.1. **Plan**

- Select the best model and subsequently the best values for hyperparameters.
- Select Best Model using our target values.
- Select the best method among
  - K Means
  - DBSCAN
  - Agglomerative Clustering

5.2. **K Means**

- It is the most commonly used model, which works on the principle of finding clusters with distinct centroids.
- We will use this method with smart initialization of centroids using 'KMeans++' method and setting random state so that we can re-produce the results.
- We will loop the algorithm over cluster numbers varying from 5 to 70.
- We will select the best cluster number based on Silhouette Score. This will be method which will be used for all other algorithms also.

```
[48]: Text(0, 0.5, 'Silhouette_score')
```



- Our Silhouette score (SS) indicates that the clusters are not very well defined, as the SS is very less (A silhouette score of 1 suggest perfect cluster boundaries, 0 means that cluster is chosen at random and -1 means not proper boundaries at all).
- Best number of clusters based on SS is somewhere between 50 to 60.

5.3. **DBSCAN**

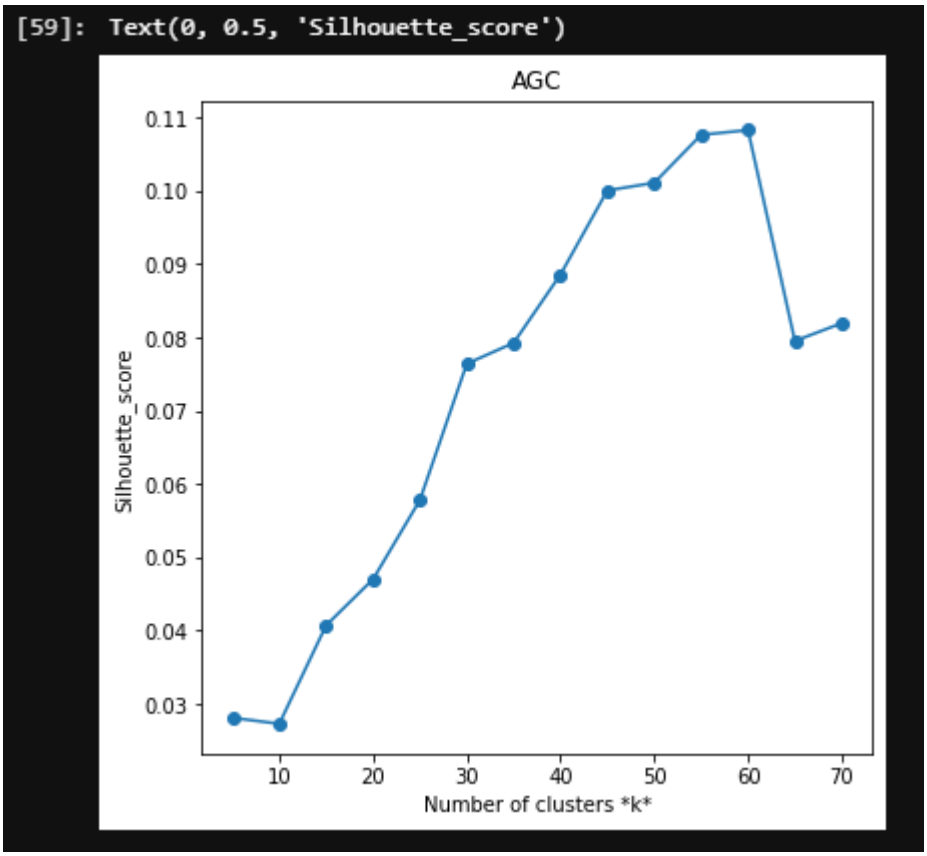
- This is a density-based clustering which doesn't require any initialization in terms of cluster numbers, however, we do optimize two hyper parameters - eps and min_samples to achieve the best clustering results.

```
[51]: min_samples = list(range(5,15,3))
      epses = list(np.linspace(15, 45, 3))
      sse = []

      for min_sample in min_samples:
          for eps in epses:
              db_estimator = Pipeline([('scaler', StandardScaler()),
                               ('dbscan', DBSCAN(eps = eps,
                                                 min_samples = min_sample,
                                                 metric = 'jaccard'
                                                ))
                              ])
              print(eps, min_sample)
              db_estimator.fit(nmf_features)
              labels = db_estimator.named_steps['dbscan'].labels_
              sse.append(silhouette_score(nmf_features, labels))

      15.0 5
```

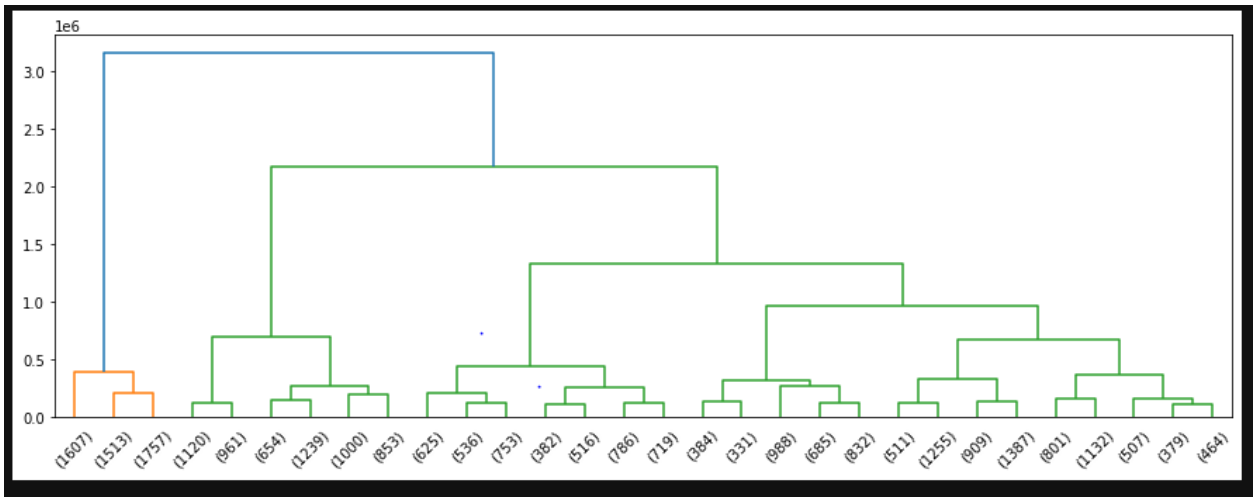
- Here, I actually tried varied range of values for eps and min_samples to achieve convergence.
- Unfortunately, all my efforts came to naught, and the model failed to converge over a wide range of values of hyperparameters.
- Here, I also deviated from K-Means algorithm by choosing a distance metric more suitable for NLP tasks i.e. Jaccard.

5.4. **Agglomerative Clustering**

- One of the widely used and easily interpretable algorithm. Again, since the algorithm provides the option to stop clustering at a predefined mark (Cluster Numbers), I opted to iterate that number over the range from 5 to 70.



- Once again, the SS suggests that clustering results obtained are not very clear.
- Agglomerative Clustering method does converge eventually here but with worst results when compared to KMeans.
- The Best SS score that I obtained using Euclidean distance and ward linkage was around 0.11 with resulting number of clusters between 50 to 55.
- I also tried Cosine distance, but the results were even worser.



5.5. **Summary**

- We tried to train 3 models and chose hyperparameters by using for loop
    1. K Means Clustering
    2. DBSCAN
    3. Agglomerative Clustering
- We set forth our expectations clearly, by setting goals as to identify/ validate the clusters using pre-defined labels available with us in the dataset.
- The model which presented best solution was surprisingly KMeans with ideal number of clusters between 50 - 55
- We saw that DBSCAN failed to converge, and Agglomerative clustering also suggested similar number of clusters as K-Mean but with lower Silhouette Score.
- We will discuss the implications of our modelling results in next section.

6. Conclusion and Future Directions

6.1. In conclusion, we can conclude that KMeans provides the best clustering results with optimal number of clusters somewhere between 50 to 55.

6.2. Going further, in future we may improve our modelling performance by,

- At the time of feature extraction, we only used unigrams and bigrams, we can also use further higher degree of feature such as trigram (combination of three words). In fact, to keep it simple we can reduce the features by only choosing unigram.
- The no of components that we used for further modelling was fixed to 70. However, we may try to further minimize the components and check the explained variance to choose the optimal number. We can then use the new components to again perform the clustering to check for any improved performance.

- While training DBSCAN Clustering algorithm I failed to obtain any convergence. Maybe we can try for even larger or smaller values of eps and min_samples to check for convergence.
- Both KMEANS as well as the Agglomerative clustering methods suggest that the optimal number of clusters is somewhere between 55-60. This may be useful for re-classifying our original dataset and reduce the number of features to the optimal cluster numbers. This is the final outcome of our project.