uOttawa

NLP Clustering

Group Assignment Two

Mostafa Mohamed Mahmoud
Mohamed Fouad Abdelnaby
Sameh Mohamed Ismail
Mohamed Helmy Mostafa

Prof. Arya Rahgozar
Dr. Migao Wu

10/31/2021

Content:

1) List of Figures and Graphs:

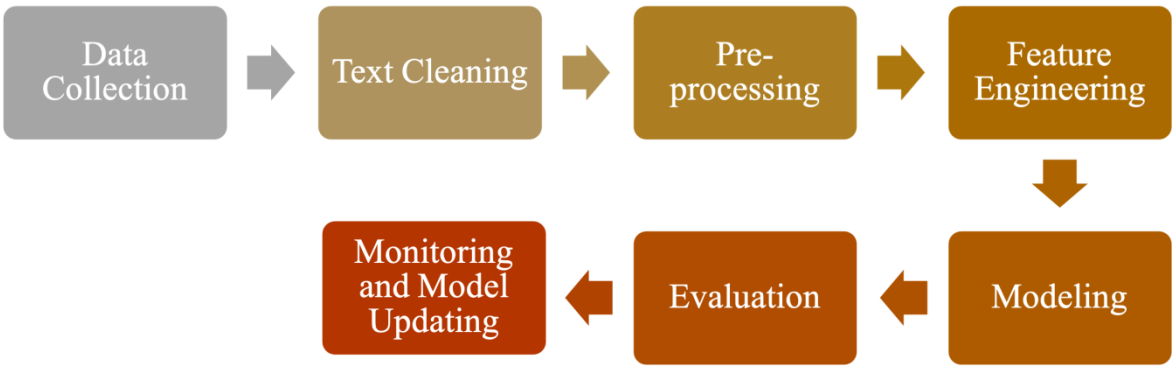
2) Abstract:

Recently, the rise of big data and natural language processing algorithm has gained a huge amount interest from the competing technology companies. Text Clustering is one of the problems solved by NLP algorithm. Text clustering refers to the process of un-supervised learning of specified text. This report describes the various techniques of text clustering, including text representation, feature engineering and clustering algorithms, and draws the basic ideas, advantages and disadvantages of several current mainstream clustering techniques.

3) Goal:

The overall aim is to produce similar clusters and compare them; analyse the pros and cons of algorithms, generate and communicate the insights.

4) Project Pipeline:



5) Introduction:

Document Clustering is a method for finding structure within a collection of documents, so that similar documents can be grouped into categories. The first step in the Clustering process is to create word vectors for the documents we wish to cluster. A vector is simply a numerical representation of the document, where each component of the vector

refers to a word, and the value of that component indicates the presence or importance of that word in the document. The distance matrix between these vectors is then fed to algorithms, which group similar vectors together into clusters.

6) Setting Up the Environment:

- Importing Libraries:

```
import requests
import re
import random
from nltk.stem import *
from nltk import word_tokenize
from nltk.stem import PorterStemmer
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score, plot_confusion_matrix
import pandas as pd
import numpy as np
import string
import nltk
from nltk.corpus import stopwords,gutenberg
from nltk.tokenize import RegexpTokenizer
from nltk.probability import FreqDist
from collections import Counter
from sklearn import datasets
from sklearn.model_selection import ShuffleSplit,KFold,cross_val_score,train_test_split
from sklearn.feature_extraction.text import CountVectorizer,TfidfTransformer
from sklearn.decomposition import LatentDirichletAllocation
from sklearn.pipeline import Pipeline
from sklearn import metrics
from sklearn.metrics import plot_confusion_matrix
import matplotlib.pyplot as plt
from sklearn.cluster import k_means
from sklearn.cluster import KMeans
import matplotlib.pyplot as mtp
from sklearn.metrics import cohen_kappa_score
from sklearn.decomposition import PCA
from sklearn.metrics import silhouette_samples, silhouette_score
from sklearn.mixture import GaussianMixture
from sklearn.metrics import homogeneity_score
from tqdm import tqdm
from gensim.test.utils import common_corpus, common_dictionary
from gensim.models.ldamodel import LdaModel
from gensim.models.coherencemodel import CoherenceModel
```

- Importing Data:

```
def get_book(url):
    book = requests.get(url).content.decode("utf-8")
    return book
```

Title	Author	Genre	Link
1. The Pirates of Ersatz	Murray Leinster	Science Fiction	https://www.gutenberg.org/ebooks/24035
2. How it Works	Archibald Williams	Technology	https://www.gutenberg.org/ebooks/28553
3. Surgical Anatomy	Joseph Maclise	Medicine	https://www.gutenberg.org/ebooks/24440
4. Quick and easy cooking	Crocker, Betty	Food	https://www.gutenberg.org/files/62841/62841-0.txt
5. Fifty Years In The Northwest	William H. C. Folsom	Travel	https://www.gutenberg.org/ebooks/36375

7) Cleaning, Partitioning and Labelling Data:

```
urls = ['https://www.gutenberg.org/cache/epub/24035/pg24035.txt',
 'https://www.gutenberg.org/cache/epub/28553/pg28553.txt',
 'https://www.gutenberg.org/ebooks/24440.txt.utf-8',
 'https://www.gutenberg.org/files/61719/61719-0.txt',
 'https://www.gutenberg.org/cache/epub/36375/pg36375.txt']


lables = ['a', 'b', 'c', 'd', 'e']

url_label = list(zip(urls, lables))
[(url, label) for url, label in url_label]
```

```
[('https://www.gutenberg.org/cache/epub/24035/pg24035.txt', 'a'),
 ('https://www.gutenberg.org/cache/epub/28553/pg28553.txt', 'b'),
 ('https://www.gutenberg.org/ebooks/24440.txt.utf-8', 'c'),
 ('https://www.gutenberg.org/files/61719/61719-0.txt', 'd'),
 ('https://www.gutenberg.org/cache/epub/36375/pg36375.txt', 'e')]
```

```
def remove_numbers(book):
    # Remove Punctuation using regex
    return re.sub('[^A-Za-z]+', ' ', book)
```

```
def remove_stopwords(book):
    pattern = re.compile(r'\b(' + r'|'.join(stopwords.words('english')) + r')\b\s*')
    return pattern.sub('', book.lower())
```

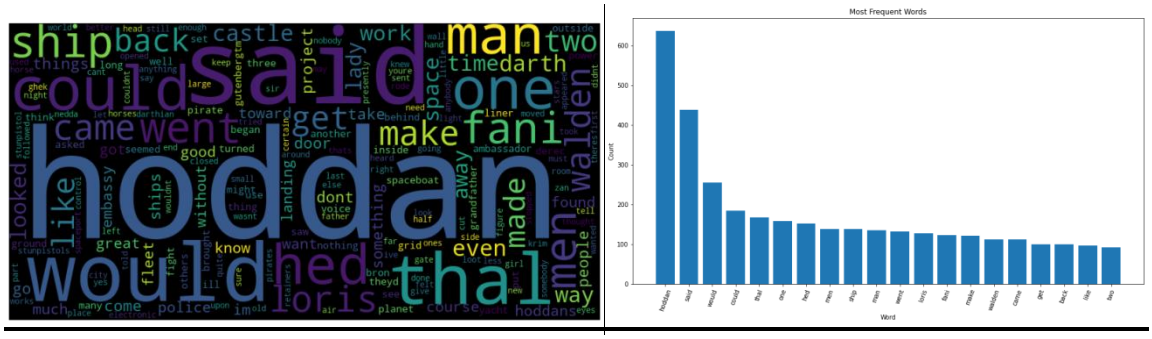
```
def split_book(book):
    # Split book to 150 words
    book_words = book.split()
    n = 150
    parts = [(book_words[i*n : (i+1)*n]) for i in range((len(book_words)+ n-1)//n)]

    # Split book to paragraphs
    book_list = []
    for part in parts[:-1]:
        book_list.append([" ".join(part)])
    del book_list[0:1]
    # Select 200 random paragraphs from the book
    random_200 = random.sample(book_list, 200)

    return random_200
```

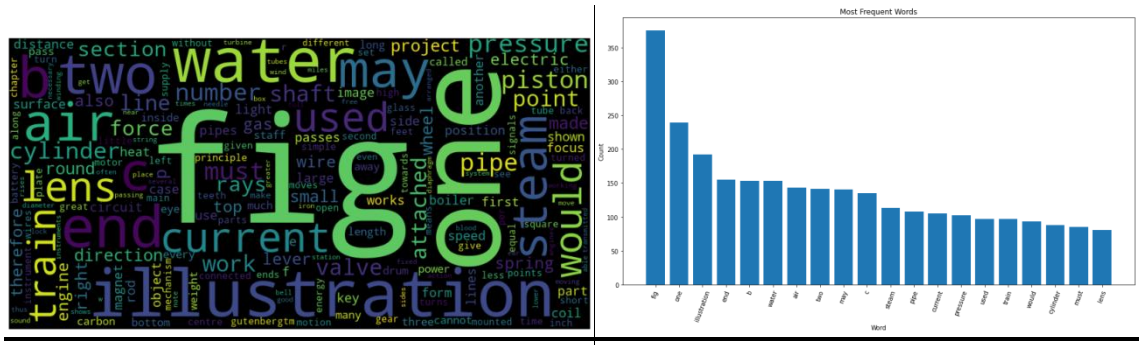
```
def parts_into_df(parts_list, label):
    df = pd.DataFrame(parts_list, columns=['Paragraphs'])
    # add label
    df['Label'] = label
    # add index
    df['Index'] = range(1, len(df)+1)

    return df
```

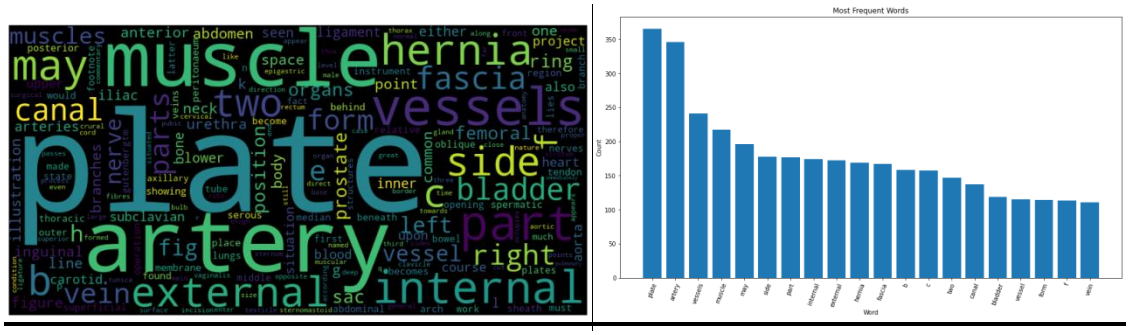
We cleaned the words from symbols and removing any not needed numbers and punctuations, lastly we removed any stop words from text. Then we performed partitioning, by taking random 200 partitions from each book, each partition consists of 150 words. Then we performed labelling and indexing and creating a Dataframe. Then we performed Exploratory Data Analysis (EDA) and showed the word cloud to know and take insights of the most repeated words and terminologies in each book as it will definitely affect our model through the different transformation techniques.

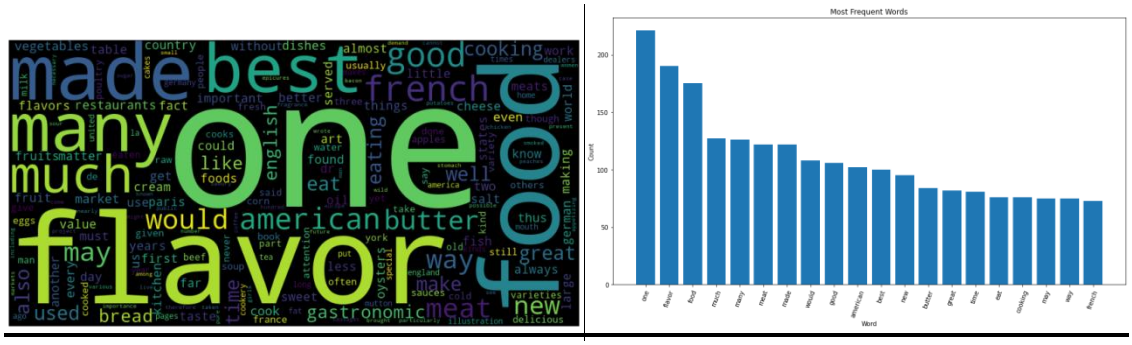
- Exploratory Data Analysis (EDA):

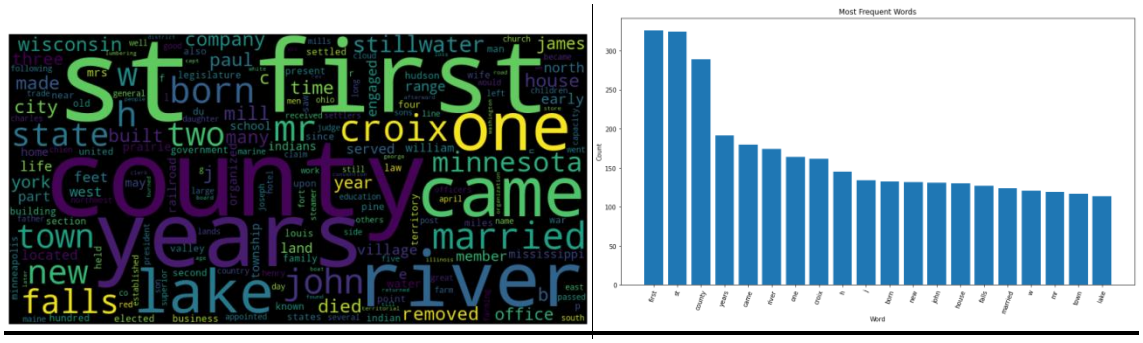
```
def plot_word_bar(book):

  book1_str = [" ".join(doc) for doc in book]

  #convert it to dictionary with values and its occurences
  word_could_dict=Counter(",".join(book1_str).split())

  plt.figure(figsize=(15,8))
  plt.bar(range(len(word_could_dict.most_common(20))), [val[1] for val in word_could_dict.most_common(20)], align='center')
  plt.xticks(range(len(word_could_dict.most_common(20))), [val[0] for val in word_could_dict.most_common(20)])
  plt.xticks(rotation=70)
  plt.title("Most Frequent Words")
  plt.xlabel("Word")
  plt.ylabel("Count")
  plt.show()
```











We can notice different word distribution from the 5 books plots…

8) Text Transformation and Feature Engineering:

We implement 4 approaches of text transformation:

- BOW:

  A bag of words is a representation of text that describes the occurrence of words within a document. We used sklearn count vectorizer which converts a collection of text documents to a matrix of token counts.

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.decomposition import PCA

ngram1_vectorizer_obj = CountVectorizer(analyzer='word', ngram_range=(1, 1)) # , max_features=200
ngram1_vectorizer_model = ngram1_vectorizer_obj.fit_transform(books_df.iloc[:,0])
ngram1_vectorizer_names = ngram1_vectorizer_obj.get_feature_names()

features1_df = pd.DataFrame(ngram1_vectorizer_model.toarray() ,columns = ngram1_vectorizer_names)
bag_of_word1= features1_df
print("Data dimensions before PCA with BOW:",bag_of_word1.shape)

count =ngram1_vectorizer_obj.fit_transform(books_df.iloc[:,0]).todense()
bag_of_word1
#perform PCA to plot
pca = PCA(n_components=2)
pca.fit(count)
BOW_2Dtransformed_data=pca.transform(count)
print("Data dimensions after PCA with BOW:",BOW_2Dtransformed_data.shape)
#print(BOW_2Dtransformed_data)
```

```
Data dimensions before PCA with BOW: (1000, 19913)
Data dimensions after PCA with BOW: (1000, 2)
```

- TF-IDF:

  Bag of word doesn't capture the importance of the word it gives you the frequency of the word. TF-IDF resolves this matter through computation of two values. TF is count of occurrences of the word in a document. IDF of the word across a set of documents. It tells us how common or rare a word is in the entire document set. The closer it is to 0, the more common is the word. This metric can be calculated by taking the total number of documents, dividing it by the number of documents that contain a word, and calculating the logarithm. We then multiply these two values TF and IDF. We used sklearn t_df-transformer which transforms a count matrix to a normalized tf or tf-idf representation.

```
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf_obj = TfidfVectorizer()
tfidf_pickle = tfidf_obj.fit(books_df.iloc[:,0])
tfidf_model = tfidf_obj.fit_transform(books_df.iloc[:,0])
tfidf_obj_names = tfidf_obj.get_feature_names()

tfidf_df1 = pd.DataFrame(tfidf_model.toarray(), columns=tfidf_obj_names)
#tfidf_df1

print("Data dimensions before PCA with BOW:",tfidf_df1.shape)

count =tfidf_obj.fit_transform(books_df.iloc[:,0]).todense()
tfidf_df1
#perform PCA to plot
pca = PCA(n_components=2)
pca.fit(count)
tfidf_2Dtransformed_data=pca.transform(count)
print("Data dimensions after PCA with BOW:",tfidf_2Dtransformed_data.shape)
tfidf_df1
```

- LDA:

  Latent Dirichlet Allocation (LDA) algorithm is an unsupervised learning algorithm that attempts to describe a set of observations as a mixture of distinct categories. LDA is most commonly used to discover a user-specified number of topics shared by documents within a text corpus. Here each observation is a document, the features are the presence (or occurrence count) of each word, and the categories are the topics. Since the method is unsupervised, the topics are not specified up front, and are not guaranteed to align with how a human may naturally categorize documents. The topics are learned as a probability distribution over the words that occur in each document. Each document, in turn, is described as a mixture of topics.

```
#LDA Transform
from sklearn.decomposition import LatentDirichletAllocation
count_vect = CountVectorizer()
X_t = count_vect.fit_transform(books_df.iloc[:,0]).todense()
lda = LatentDirichletAllocation(n_components=5,random_state=0)
LDA_out= lda.fit_transform( X_t)
print("Data dimensions before PCA with LDA and BOW:",LDA_out.shape)
#perform PCA to plot
pca = PCA(n_components=2)
pca.fit(LDA_out)
LDA_2Dtransformed_data=pca.transform(LDA_out)
print("Data dimensions after PCA with LDA and BOW:",LDA_2Dtransformed_data.shape)
print(LDA_2Dtransformed_data)
```

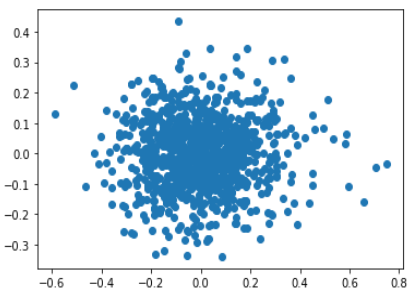
```
Data dimensions before PCA with LDA and BOW: (1000, 5)
Data dimensions after PCA with LDA and BOW: (1000, 2)
[[-0.28244911  0.52253547]
 [-0.21482941  0.81663915]
 [-0.21910471  0.80175492]
 ...
 [-0.59617455 -0.50859509]
 [-0.50127924 -0.39514889]
 [-0.59610858 -0.50855262]]
```

- Word-Embedding:

  A word embedding is a learned representation for text where words that have the same meaning have a similar representation. Each word is represented by a real-valued vector, often tens or hundreds of dimensions. This is contrasted to the thousands or millions of dimensions required for sparse word representations, such as a one-hot encoding.

```
# make word2vec and apply in document
from gensim.test.utils import common_texts
from gensim.models.doc2vec import Doc2Vec, TaggedDocument

docs = [d for d in books_df['Paragraphs']]
documents = [TaggedDocument(doc, [i]) for i, doc in enumerate(docs)]

vec_size = 50
alpha = 0.025
d2v_model = Doc2Vec(vector_size=vec_size,
                alpha=alpha,
                min_alpha=0.0025,
                min_count=1,
                dm =1)
d2v_model.build_vocab(documents)
d2v_model.train(documents, total_examples=d2v_model.corpus_count, epochs=d2v_model.epochs)
d2v=d2v_model.docvecs.doctag_syn0
#print(d2v)
print(d2v.shape)
pca = PCA(n_components=2)
result = pca.fit_transform(d2v)
plt.scatter(result[:, 0], result[:, 1])
```

```
(1000, 50)
```



9) Clustering Algorithms:

We implement 3 algorithms:

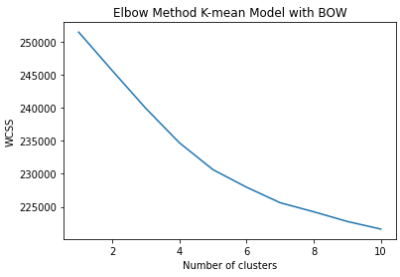
- K-Means:
  The goal of this algorithm is to find groups in the data, with the number of groups represented by the variable K. The algorithm works iteratively to assign each data point to one of K groups based on the features that are provided. Data points are clustered based on feature similarity. The results of the K-means clustering algorithm are:

  - The centroids of the K clusters, which can be used to label new data
  - Labels for the training data (each data point is assigned to a single cluster)

We implement the *K-Means Model* with one of each transformation techniques and we use El-Bow Method to optimize the number of clusters we however we need only 5 clusters:

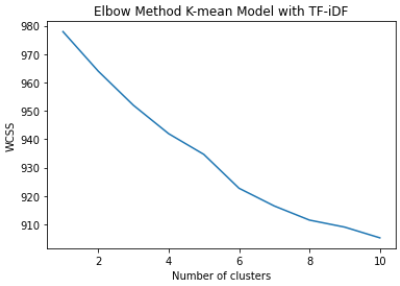
- **K-Means With Bow**:

```
# applay K-mean with BOW
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', max_iter=300, n_init=40, random_state=7)
    kmeans.fit(bow_helmy)
    wcss.append(kmeans.inertia_)
plt.plot(range(1, 11), wcss)
plt.title('Elbow Method K-mean Model with BOW')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()

kmeans = KMeans(n_clusters=5, init='k-means++', max_iter=300, n_init=140, random_state=7)
pred_y= kmeans.fit_predict(bow_helmy)
plot_BOW( pred_y ,'K-mean Model with BOW')
# get accuricy of model
names =['a', 'b', 'c', 'd', 'e']
print("\n")
```



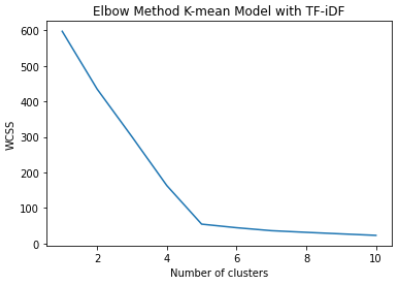
- **K-Means With TF-IDF:**

```
# apply the k-mean on TF-IDF data transformation
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', max_iter=300, n_init=10, random_state=0)
    kmeans.fit(tfidf_helmy)
    wcss.append(kmeans.inertia_)
plt.plot(range(1, 11), wcss)
plt.title('Elbow Method K-mean Model with TF-iDF')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()

kmeans = KMeans(n_clusters=5, init='k-means++', max_iter=300, n_init=150, random_state=7)
pred_y = kmeans.fit_predict(tfidf_helmy)
plot_TF_IDF( pred_y ,'K-mean Model with TF-iDF')
# get accuricy of model
names =['a', 'b', 'c', 'd', 'e']
print("\n")
```



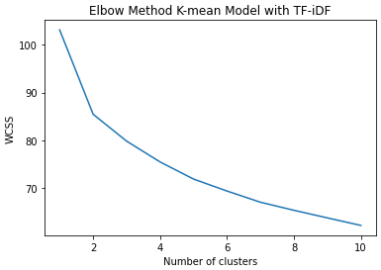
- **K-Means With LDA:**

```
# apply the k-mean on lad data transformation
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', max_iter=300, n_init=10, random_state=0)
    kmeans.fit(helmy_lda)
    wcss.append(kmeans.inertia_)
plt.plot(range(1, 11), wcss)
plt.title('Elbow Method K-mean Model with LDA')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()

kmeans = KMeans(n_clusters=5, init='k-means++', max_iter=300, n_init=150, random_state=7)
pred_y = kmeans.fit_predict(helmy_lda)

plot_LDA( pred_y ,'K-mean Model with LDA')
# get accuricy of model
names =['a', 'b', 'c', 'd', 'e']
print("\n")
```



- **K-Means With Word-Embedding:**

```
# apply the k-mean on word to vec data transformation
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', max_iter=300, n_init=10, random_state=0)
    kmeans.fit(helmy_word_vec)
    wcss.append(kmeans.inertia_)
plt.plot(range(1, 11), wcss)
plt.title('Elbow Method K-mean Model with word to vec')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()

kmeans = KMeans(n_clusters=5, init='k-means++', max_iter=300, n_init=50, random_state=7)
pred_y = kmeans.fit_predict(helmy_word_vec)
print(pred_y)
plot_word__em( pred_y ,'K-mean Model with word to vec ')
#model_accuricy(helmy_word_vec,y_predict)
# get accuricy of model
names =['a', 'b', 'c', 'd', 'e']
print("\n")
```



- Expectation Maximization (EM):

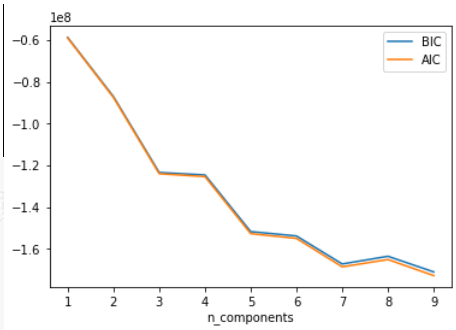
  The expectation-maximization algorithm is an approach for performing maximum likelihood estimation in the presence of latent variables. It does this by first estimating the values for the latent variables, then optimizing the model, then repeating these two steps until convergence. It is an effective and general approach and is most commonly used for density estimation with missing data, such as clustering algorithms like the Gaussian Mixture Model.

```
def EM_model(X):
  n_components = np.arange(1, 10)
  models = [GaussianMixture(n, covariance_type='diag', random_state=0).fit(X) for n in n_components]
  plt.plot(n_components, [m.bic(X) for m in models], label='BIC')
  plt.plot(n_components, [m.aic(X) for m in models], label='AIC')
  plt.legend(loc='best')
  plt.xlabel('n_components');
  plt.show()

  GM = GaussianMixture(n_components=5, covariance_type='spherical', random_state=7)
  GM.fit(X)

  y_pred = GM.predict(X)
  return y_pred
```

We implement the *EM Model* with one of each transformation techniques:

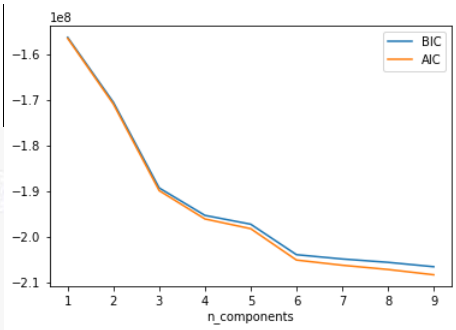
- **EM With Bow**:

```
label = EM_model(bow__sameh)
plot_BOW(label , 'GaussianMixture prediction using BoW' )
#model_accuricy(bow__sameh ,label)
names =['a', 'b', 'c', 'd', 'e']
print("\n")
x=[]
```



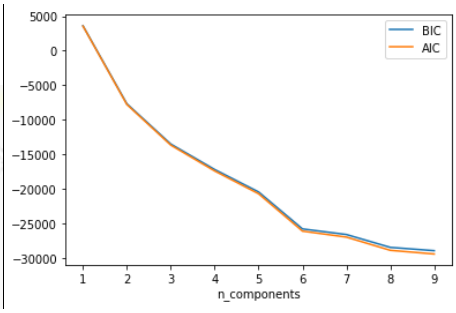
- **EM With TF-IDF:**

```
# apply GaussianMixture on TF-IDF on and make predicton
label =EM_model(fidf_sameh)
plot_TF_IDF(label , 'GaussianMixture prediction using TF-IDF ' )

# get accurecy
names =['a', 'b', 'c', 'd', 'e']
print("\n")
x=[]
```



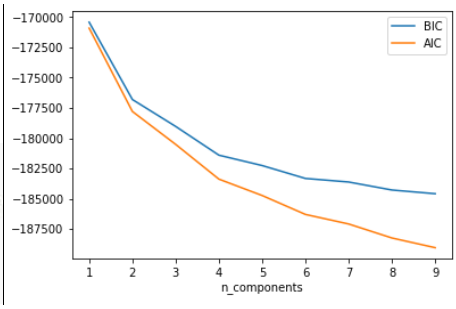
- **EM With LDA:**

```
# apply GaussianMixture on LDA data  on and make predicton
label = EM_model(lda_sameh)
plot_LDA(label , 'GaussianMixture prediction using LDA  ' )

# get accurecy
names =['a', 'b', 'c', 'd', 'e']
print("\n")
x=[]
```



- **EM Word-Embedding:**

```
# apply GaussianMixture on word embeded on and make predicton
label = EM_model(sameh_word_vec)
plot_word__em(label , 'GaussianMixture prediction using word embeded: ' )
# get accurecy of the model
names =['a', 'b', 'c', 'd', 'e']
print("\n")
x=[]
```



- Hierarchical Agglomerative:

  The agglomerative clustering is the most common type of hierarchical clustering used to group objects in clusters based on their similarity. It's also known as

AGNES (Agglomerative Nesting). The algorithm starts by treating each object as a singleton cluster. Next, pairs of clusters are successively merged until all clusters have been merged into one big cluster containing all objects. The result is a tree-based representation of the objects, named dendrogram.

```
def dist_matrix(features):
  dist_matrix = euclidean_distances(features, features)
  Z_using_dist_matrix = hierarchy.linkage(dist_matrix, 'complete')
  return Z_using_dist_matrix

def draw_dendo(Z_using_dist_matrix):
  fig = pylab.figure(figsize=(18,50))
  dendro = hierarchy.dendrogram(Z_using_dist_matrix, leaf_rotation=0, leaf_font_size =12, orientation = 'right')
```

We implement the *Hierarchical Agglomerative Model* with one of each transformation techniques:

- **Hierarchical Agglomerative With Bow:**

```
agg_bow_model = AgglomerativeClustering(n_clusters=5).fit(bow_model.toarray(), y)
#agg_bow_model
```

```
AgglomerativeClustering(affinity='euclidean', compute_full_tree='auto',
                        connectivity=None, distance_threshold=None,
                        linkage='ward', memory=None, n_clusters=5)
```

```
label =agg_bow_model.labels_
data =bow_model
names =['a', 'b', 'c', 'd', 'e']
print("\n")
x=[]
```

- **Hierarchical Agglomerative With TF-IDF:**

```
agg_tfidf_model = AgglomerativeClustering(n_clusters=5).fit(tfidf_model.toarray(), y)
#agg_tfidf_model
```

```
label =agg_tfidf_model.labels_
data =tfidf_model
names =['a', 'b', 'c', 'd', 'e']
print("\n")
x=[]
```

- **Hierarchical Agglomerative With LDA:**

```
agg_lda_bow_model = AgglomerativeClustering(n_clusters=5).fit(lda_bow_model.toarray(), y)
#agg_lda_bow_model
```

```
label =agg_lda_bow_model.labels_
data =lda_bow_model
names =['a', 'b', 'c', 'd', 'e']
print("\n")
x=[]
```

- **Hierarchical Agglomerative Word-Embedding:**

```
agg_doc2vec_model = AgglomerativeClustering(n_clusters=5, linkage='complete').fit(d2v_sparse.toarray(), y)
#agg_doc2vec_model
```

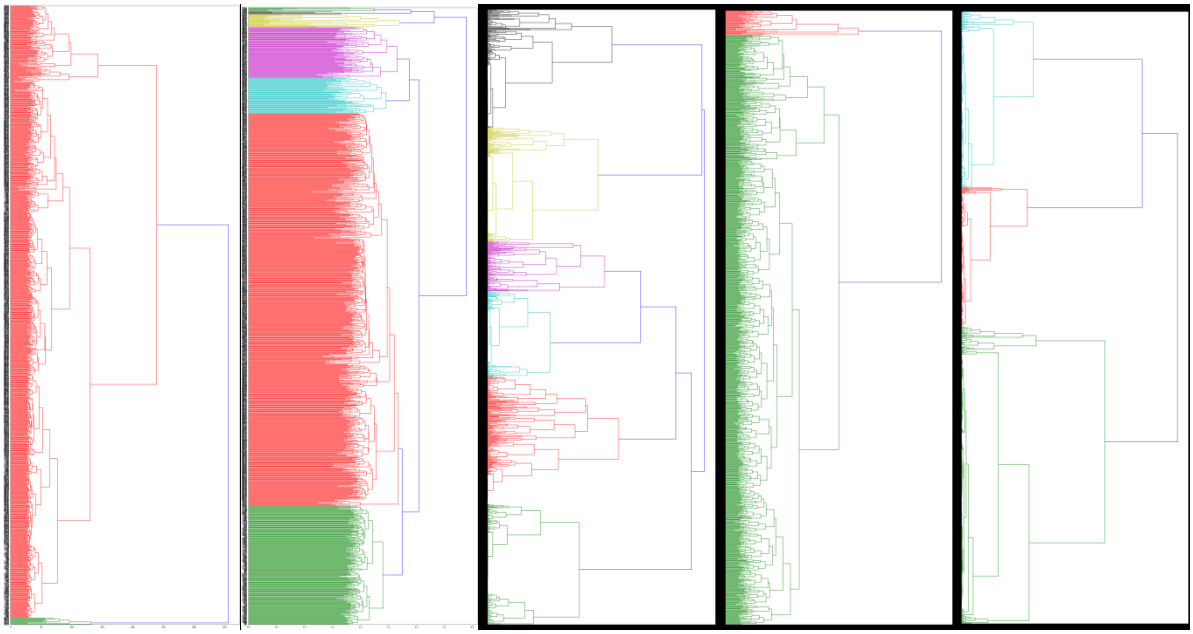
```
label =agg_doc2vec_model.labels_
data =d2v
names =['a', 'b', 'c', 'd', 'e']
print("\n")
x=[]
```

- **Hierarchical Agglomerative LDA + TFIDF:**

```
agg_lda_tfidf_model = AgglomerativeClustering(n_clusters=5).fit(lda_tfidf_model.toarray(), y)
#agg_lda_tfidf_model
```

```
label =agg_lda_tfidf_model.labels_
data =lda_tfidf_model
names =['a', 'b', 'c', 'd', 'e']
print("\n")
x=[]
```

- ***Dendrogram for (BoW, TF-IDF, LDA, Word-Embedding, LDA+TF-IDF)*** **From left to right:**



10) Evaluation:

We try 3 different techniques/measurements:

```
def calc_scores(transformation_model, clustering_model, y):
  silh = silhouette_score(transformation_model.toarray(), clustering_model.fit_predict(transformation_model.toarray()))
  kap = cohen_kappa_score(y, clustering_model.labels_)

  print("Silhouette Coefficient is:", round(silh, 3))
  print("Cohen's kappa Score is:", round(kap, 4))
```

- Cohen's Kappa:

  Cohen's kappa coefficient (κ) is a statistic that is used to measure inter-rater reliability (and also intra-rater reliability) for qualitative (categorical) items. **Cohen's kappa is more informative than overall accuracy** when working with unbalanced data. Keep this in mind when you compare or optimize clustering models!

We evaluate the *K-Means* Model using Cohen's Kappa with one of each transformation techniques:

- **K-Means With Bow**
- **K-Means With TF-IDF**
- **K-Means With LDA**
- **K-Means With Word-Embedding**

- Coherence:
  Topic coherence formulae take top-n words of a topic and output a real valued score- based on some threshold one can identify that the topic is coherent/interpretable. Point wise mutual information (PMI) based formulae is well-known because of their high correlation with human judgments. In, PMI and its variations are used as term weights whereas in used as semantic similarity measure.

We evaluate the *EM Model* using coherence with one of each transformation techniques:

- **EM With Bow**
- **EM With TF-IDF**
- **EM With LDA**
- **EM Word-Embedding**

```
import gensim
from gensim.utils import simple_preprocess
from gensim.models import CoherenceModel
# Enable logging for gensim - optional
import warnings
warnings.filterwarnings("ignore",category=DeprecationWarning)

book_doc = books_df['Paragraphs'].values.tolist()
doc_list =[]
for doc in book_doc:
  doc_list.append(doc.split())
print(doc_list)
# Create a dictionary representation of the documents.
dictionary = Dictionary(doc_list)
dictionary.filter_extremes(no_below=10, no_above=0.2)

#Create dictionary and corpus required for Topic Modeling
corpus = [dictionary.doc2bow(doc) for doc in doc_list]
# View
print('Number of unique tokens: %d' % len(dictionary))
print('Number of documents: %d' % len(corpus))
print([[(dictionary[id], freq) for id, freq in cp] for cp in corpus[:1]])
lda_model =LdaModel(corpus=corpus,
                    id2word=dictionary,
                    num_topics=5,
                    random_state=100,
                    update_every=1,
                    chunksize=100,
                    passes=10,
                    alpha='auto',
                    per_word_topics=True)
cm = CoherenceModel(model =lda_model,texts=doc_list,corpus=corpus,coherence='c_v')
# Compute Perplexity
print('\nPerplexity: ', lda_model.log_perplexity(corpus))  # a measure of how good the model is. lower the better.

# Compute Coherence Score
coherence_lda = cm.get_coherence()
print('\nCoherence Score: ', round(coherence_lda, 4))
```

- Silhouette:
  Silhouette analysis allows you to calculate how similar each observation is with the cluster it is assigned relative to other clusters. This metric (silhouette width)

ranges from -1 to 1 for each observation in your data and can be interpreted as follows:

- Values close to 1 suggest that the observation is well matched to the assigned cluster
- Values close to 0 suggest that the observation is borderline matched between two clusters
- Values close to -1 suggest that the observations may be assigned to the wrong cluster

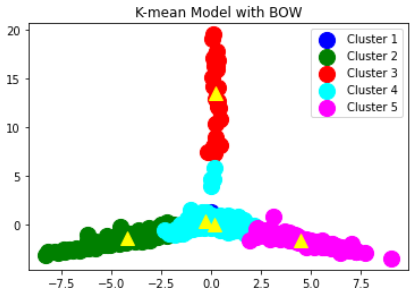
We evaluate the *Hierarchical Agglomerative Model* using Silhouette with one of each transformation techniques:

- **Hierarchical Agglomerative With Bow**
- **Hierarchical Agglomerative With TF-IDF**
- **Hierarchical Agglomerative With LDA**
- **Hierarchical Agglomerative Word-Embedding**

❖ Evaluation Graphs and Scores for Each transformation technique:

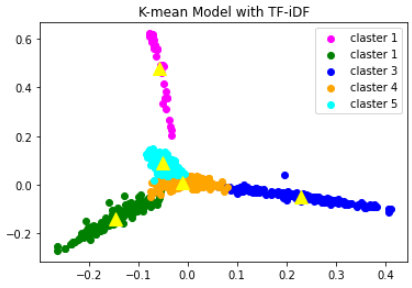
○ K-Means With Bow:

	Accuracy	Kappa	Silhouette	homogeneity
Score	0.71	0.6375	0.0264	0.6207



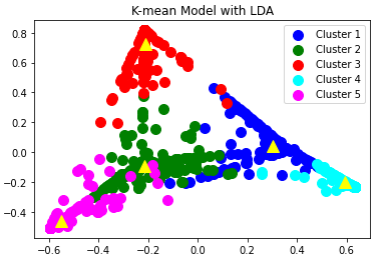
○ K-Means With TF-IDF:

	Accuracy	Kappa	Silhouette	homogeneity
Score	0.739	0.674	0.027	0.710



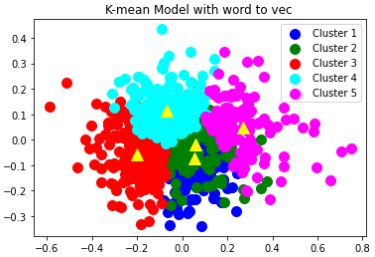
- **K-Means With LDA:**

	Accuracy	Kappa	Silhouette	homogeneity
Score	0.936	0.92	0.0215	0.832



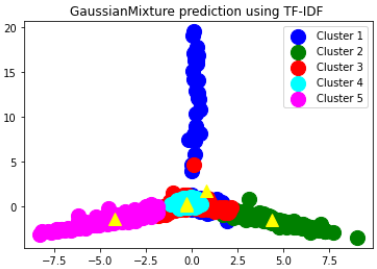
- **K-Means With Word-Embedding:**

	Accuracy	Kappa	Silhouette	homogeneity
Score	0.178	-0.007	0.086	0.0191



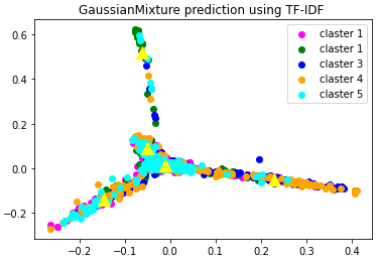
- **EM With BoW:**

	Accuracy	Kappa	Silhouette	homogeneity
Score	0.832	0.79	0.00371	0.6597



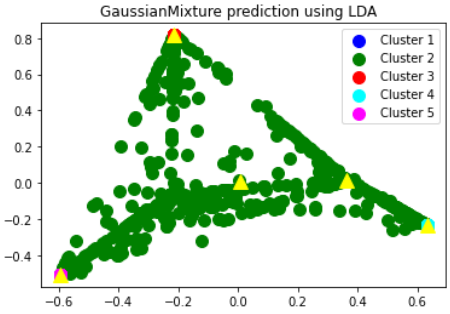
- **EM With TF-IDF:**

	Accuracy	Kappa	Silhouette	homogeneity
Score	0.744	0.68	0.0269	0.7142



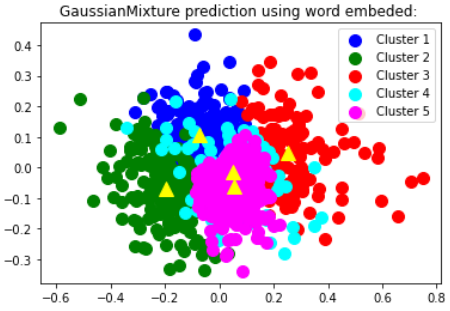
- **EM With LDA:**

	Accuracy	Kappa	Silhouette	homogeneity
Score	0.577	0.47125	0.2056	0.375



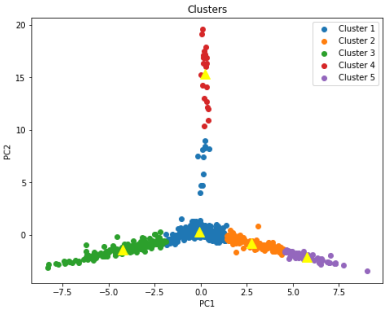
- **EM With Word-Embedding:**

	Accuracy	Kappa	Silhouette	homogeneity
Score	0.224	0.03	0.078	0.0184



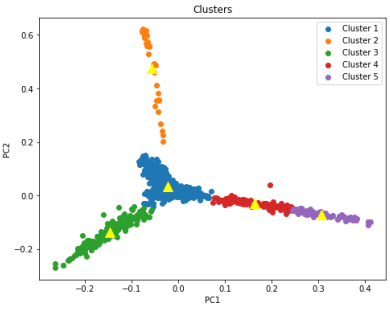
- **Hierarchical Agglomerative With BoW:**

	Accuracy	Kappa	Silhouette	homogeneity
Score	0.764	0.705	0.0247	0.762



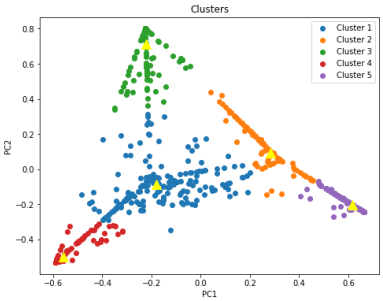
- **Hierarchical Agglomerative With TF-IDF:**

	Accuracy	Kappa	Silhouette	homogeneity
Score	0.768	0.71	0.027	0.777



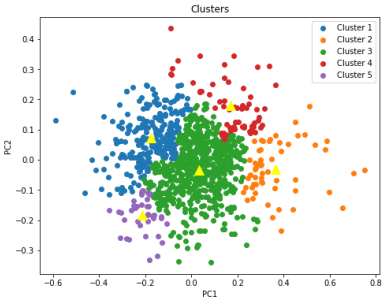
- **Hierarchical Agglomerative With LDA:**

	Accuracy	Kappa	Silhouette	homogeneity
Score	0.933	0.91625	0.723	0.8205



- **Hierarchical Agglomerative With Word-Embedding:**

	Accuracy	Kappa	Silhouette	homogeneity
Score	0.179	-0.02625	0.0326	0.0192

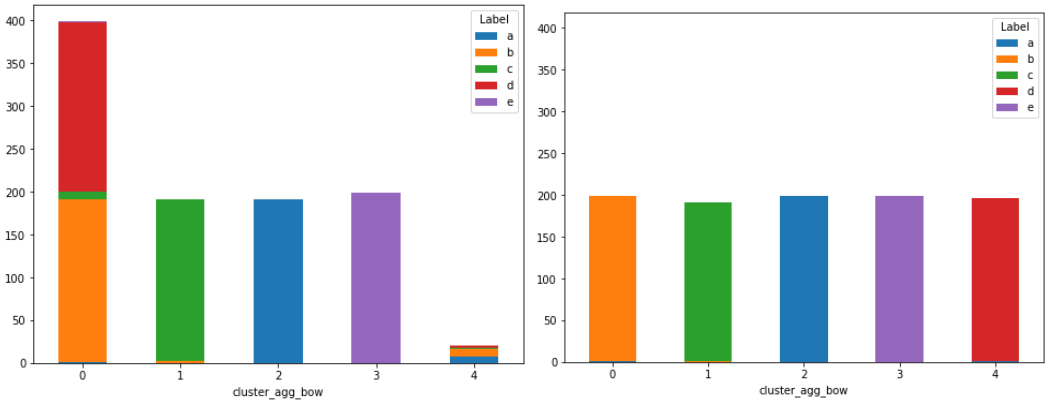


11) Models Comparison and Champion (RED):

The average accuracy of the Kappa for all combinations:

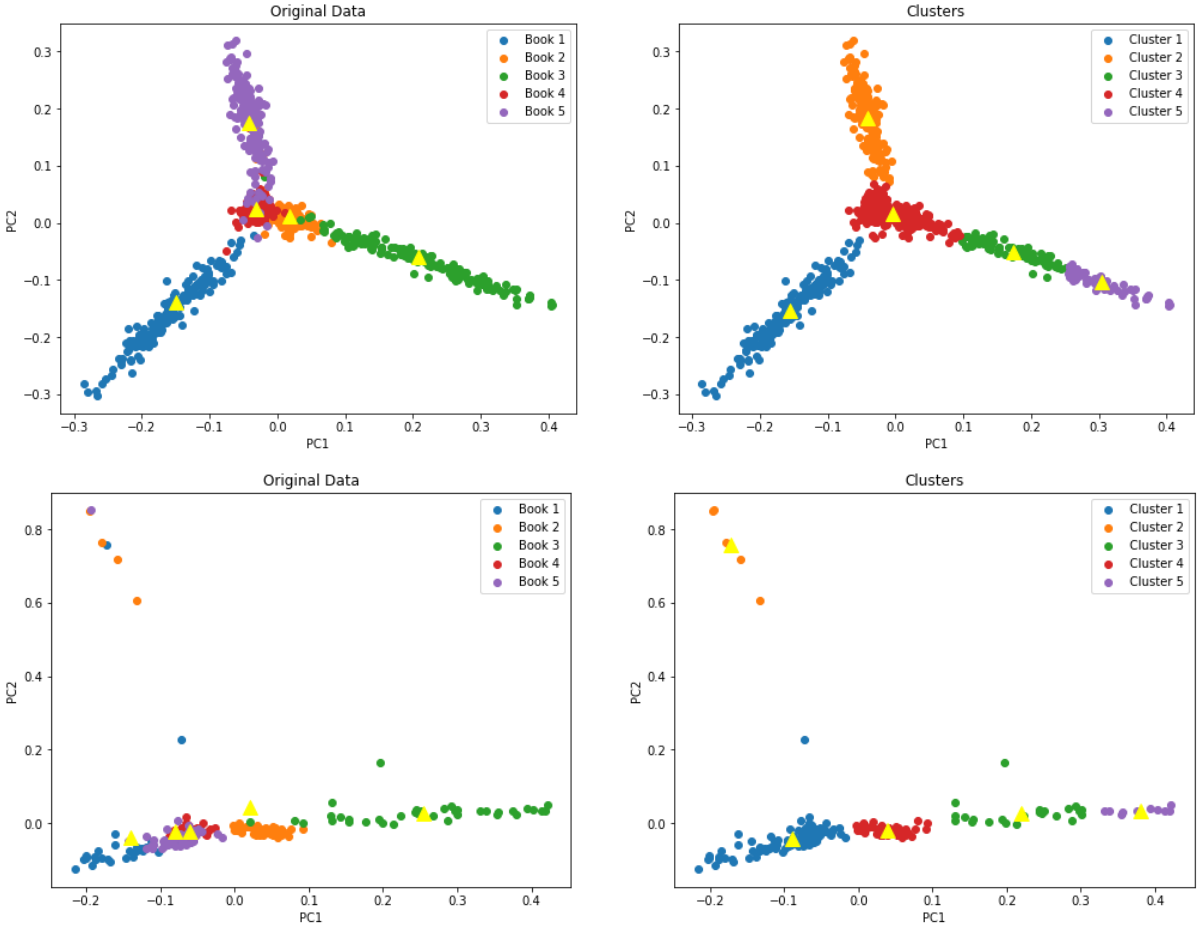
	BOW	TF-IDF	LDA	WE
K-Means	0.6375	0.674	0.92	-0.007
EM	0.77	0.68	0.47125	0.03
HA	0.705	0.71	0.91625	-0.02625

Now we have for each combination between (model and transformation) the best accuracy of human accuracy…..



12) Error Analysis:

- ❖ Remove the very beginning and the very last paragraphs of each book
- ❖ Remove the most frequent and the least weighted words from TF-IDF Transformation
- ❖ Try different collections of books



14) References:

- Aletras N and Stevenson M. Evaluating topic coherence using distributional semantics. In: Proceedings of the 10th International Conference on Computational Semantics (IWCS) – Long Papers, 2013, pp. 13-22.
- McHugh, Mary L. (2012). "Interrater reliability: The kappa statistic". Biochemia Medica. 22 (3): 276–282. doi:10.11613/bm.2012.031. PMC 3900052
- https://www.cs.cmu.edu/~schneide/tut5/node42.html
- https://campus.datacamp.com/courses/cluster-analysis-in-r/k-means-clustering?ex=9
- https://docs.aws.amazon.com/sagemaker/latest/dg/lda.html
- https://nlp.stanford.edu/IR-book/html/htmledition/hierarchical-agglomerative-clustering-1.html