Hacettepe University

Department of Computer Engineering

AIN442 Intro. to NLP Laboratory

Assignment V

Name and Surname: Abdullah Palaz

Student Number: 21993016

Subject: LSA and Word2Vec

Submission Date: 22.12.2021

Due Date: 09.01.2022 – 23:00

Instructors: Ebru Sezer, Hayriye Çelikbilek

I. Introduction

Given text and their categories, the task is to predict the category of a given text. The assignment is to represent given texts in Word2Vec, LSA and LSA2Vec (mix of the W2V and LSA methods) methods. Given texts are in Turkish hence requires different preprocessing methods. This task is a multiclass classification since there are 7 different classes.

II. Data

The dataset consists of 4900 entries, 700 per class, meaning that the dataset has no class imbalance problem. Each entry has a text and a class label associated with it. These texts are in informal language, meaning that they should be normalized.

7 classes are the following:

- teknoloji (technology)
- siyaset (politics)
- kultur (culture)
- dunya (world)
- ekonomi (economy)
- saglik (health)
- spor (sports)

III. Method

I've selected Naïve Bayes, Logistic Regression and Random Forest Classifier to see which will fit the best for this data in the aforementioned methods. I will use 5-Fold Grid Search to tune the parameters of these models.

IV. Development

a. Plan

The plan is to first try to represent given texts with LSA and try the selected models on the data, then same procedure with W2V, then with their combination.

To create the combination, LSA2Vec, there will be 3 different methods.

1. Concatenation
2. Addition
3. Natural Mix

In **Concatenation** method, we basically concatenate the LSA representation of a sentence with its W2V representation to obtain a LSA2Vec representation.

Example:

Assuming sentence A has the following:

- LSA representation: [0, 23, 1, 42, 59]
- W2V representation: [14, 2, 5, 7, 89, 75, 14]

Then, we concatenate them to get the LSA2Vec representation:

- LSA2Vec representation: [14, 2, 5, 7, 89, 75, 14, 0, 23, 1, 42, 59]

In **Addition** method, we create a representation by adding the highest score of LSA representation of a sentence to its W2V representation.

Example:

Assuming sentence A has the following:

- LSA representation: [0, 23, 1, 42, 59]
- W2V representation: [14, 2, 5, 7, 89, 75, 14]

Then, highest score of LSA representation is 59. We add this to every element of W2V representation. At the end, we have:

- LSA2Vec representation: [73, 61, 64, 66, 148, 134, 73]

Note that all the numbers here are positive. It is because we get the absolute of all of them in all the approaches.

In **Natural Mix** method, we feed W2V representation of the texts to LSA model to get a representation that looks like LSA but instead being a mixture of both methods.

Hardware and time are limitations in this assignment. Due to hardware and time difficulties, I couldn’t do all the work I intended to do.

b. Analysis

I have checked if the dataset consists of NA values, as well as empty strings. Since there is only one column with a long text, I didn't perform any further analysis. The outcome of my analysis showed that the data is clean and ready to be used in the models after preprocessing.

c. Design

First, reading data. Then, preprocessing the data. Preprocessing includes lowercasing and punctuation removal. It does not include number replacement, stop-word removal and lemmatization due to language of the texts.

In fact, I tried to use Zemberek, a library for Turkish language processing but due to hardware limitations, I couldn't do it. It crashed my computer and I think it is because RAM wasn't enough.

d. Implementation

In [1]:
```
import pandas as pd
import gensim
import numpy as np

import warnings
warnings.filterwarnings('ignore')
```

Reading Data

In [67]:
```
df = pd.read_csv("turkish_dataset.csv")
df
```

Out[67]:

	category	text
0	siyaset	3 milyon ile ön seçim vaadi mhp nin 10 olağan...
1	siyaset	mesut_yılmaz yüce_divan da ceza alabilirdi pr...
2	siyaset	disko lar kaldırılıyor başbakan_yardımcısı ar...
3	siyaset	sarıgül anayasa_mahkemesi ne gidiyor mustafa_...
4	siyaset	erdoğan idamın bir haklılık sebebi var demek ...
...	...	...
4895	teknoloji	iphone lara geri dönüyor ios 6 sürümüyle tele...
4896	teknoloji	muslukta devrim sadece elimizi yıkadığımız mu...
4897	teknoloji	halka iyi anlatılmalı bilgi_teknolojileri ile...
4898	teknoloji	çöpe gidiyorlar apple 775 bin uygulamayla app...
4899	teknoloji	google bu kez edward_gorey dedi ! google bu k...

4900 rows × 2 columns

In [3]:
```
df["text"].iloc[0]
```

Out[3]:
```
' 3 milyon ile ön seçim vaadi mhp nin 10 olağan büyük kurultayı nda konuşan genel başkan adayı koray_aydın seçimlerden önce par
tinin üye sayısının 3 milyona ulaştırılması hedefini koyarak ön seçim uygulaması vaadinde bulundu mhp nin 10 olağan büyük kurul
tayı nda konuşan genel başkan adayı koray_aydın seçimlerden önce partinin üye sayısının 3 milyona ulaştırılması hedefini koyara
k ön seçim uygulaması vaadinde bulundu genel_başkan adayı koray_aydın kürsüye beklenirken yapılan tezahüratlar ve ıslıklamalar
üzerine divan başkanı tuğrul_türkeş mhp nin genel başkanlığı da genel başkan adaylığı da saygıdeğer işlerdir bu salondaki herke
s ciddiye almak zorundadır dedi ve taşkınlıklara izin verilmeyeceğini salonda sükunet sağlanmadan konuşmaların başlamayacağını
vurguladı türkeş devlet_bahçeli nin kurultay açılışında konuştuğu için adaylık nedeniyle ikinci bir konuşma yapmayacağını açıkl
adı konuşmasında kurultayın mhp nin tek başına iktidarına vesile olmasını dileyen aydın ak_parti nin mhp yi eleştirirken kalele
ri bir bir fethederek yollarına devam ettiklerini söylediğini hatırlatarak iktidarın basın ve sivil toplumu susturduğunu ifade
```

Empty string check

In [4]:
```
a = df[(df["text"] == "")]

unique, counts = np.unique(a, return_counts=True)
dict(zip(unique, counts))
```

Out[4]:
```
{}
```

Proof that this approach works

In [5]:
```
a = df[(df["text"] == ' 3 milyon ile ön seçim vaadi mhp nin 10 olağan büyük kurultayı nda konuşan genel başkan adayı koray_aydın

unique, counts = np.unique(a, return_counts=True)
dict(zip(unique, counts))
```

Out[5]:
```
{' 3 milyon ile ön seçim vaadi mhp nin 10 olağan büyük kurultayı nda konuşan genel başkan adayı koray_aydın seçimlerden önce pa
rtinin üye sayısının 3 milyona ulaştırılması hedefini koyarak ön seçim uygulaması vaadinde bulundu mhp nin 10 olağan büyük kuru
ltayı nda konuşan genel başkan adayı koray_aydın seçimlerden önce partinin üye sayısının 3 milyona ulaştırılması hedefini koyar
ak ön seçim uygulaması vaadinde bulundu genel_başkan adayı koray_aydın kürsüye beklenirken yapılan tezahüratlar ve ıslıklamalar
üzerine divan başkanı tuğrul_türkeş mhp nin genel başkanlığı da genel başkan adaylığı da saygıdeğer işlerdir bu salondaki herke
s ciddiye almak zorundadır dedi ve taşkınlıklara izin verilmeyeceğini salonda sükunet sağlanmadan konuşmaların başlamayacağını
vurguladı türkeş devlet_bahçeli nin kurultay açılışında konuştuğu için adaylık nedeniyle ikinci bir konuşma yapmayacağını açıkl
adı konuşmasında kurultayın mhp nin tek başına iktidarına vesile olmasını dileyen aydın ak_parti nin mhp yi eleştirirken kalele
ri bir bir fethederek yollarına devam ettiklerini söylediğini hatırlatarak iktidarın basın ve sivil toplumu susturduğunu ifade
etti ak_parti nin bürokraside taş üstüne taş bırakmadığını ileri süren aydın ülkücüleri düşman kabule ederek onları kıyma makin
elerinden geçirecek bir zihniyetle sürgün ederek oraya buraya saldırarak bürokrasideki ülkücü kadrolara savaş açtılar dedi yaşa
nan bütün skandalların ardından devleti çete mantığıyla yöneten siyasi iktidarın olduğunu savunan aydın iktidarın belediyelere
sahte raporlarla ve dinlemelerle saldırdığını savunan aydın arkasından habur dan içeri soktukları vatan hainlerine karşılama tö
renleri yetmez gibi oslo da teröristlerle kurdukları pazarlık masalarında suçüstü yakalanınca da ben görmedim diyerek bunu ispa
t edecek biri varsa şerefle ispat etsin diyerek ses kayıtları çıkınca da kıvırarak sahiplenemeyerek yaptığı işin üzerine şal ör
tmeye çalışarak siyasi riyakarlıkta sınır tanımayan siyasi iktidarla karşı karşıyayız diye konuştu mintika temizliği yapiyorlar
ak_parti nin ne yaptığını iyi bildiğini türkiye de ihtilal teşebbüsü var diyerek ordunun subaylarını yargılama adı altında ceza
evine koyduğunu savunan aydın önce milletin bu darbeciler ortadan kalksın diyerek desteklediği sonra plan gereği sürek avına çe
virerek türk ordusunun neredeyse yarısını içeri atan bu zihniyet mıntıka temizliği yapıyor kuracakları yeni türkiye modeline en
gel olmasın diye bunları kaldırıyorlar bu işi hainçe yapanlar ne zaman ki şehit cenazeleri türkiye yi ağlatmaya başlarken acıla
rımızla yaşarken türkiye nin başbakanı gerekirse öcalan la yeniden görüşebilirim diyor sayın_başbakan ne görüşeceksin öcalan la
ne söyleyeceksin oraya bir masa koymuşsun masanın üstünde türkiye karşında öcalan ne kadar istiyorsun şu kadar versem yeter mi
diyeceksin öcalan yüzsüzlük eder de türkiye nin tamamını isterse ne yapacaksın diye konuştu o zaman ne yapacağız yeni anayasa k
apsamında türk milletinin adının anayasadan çıkarılarak bir alt kimlik haline getirileceğini türk milletine etnisite temelli ya
klaşılacağını savunan aydın bu_türk milletinin varlığını ötüken de söğüt te türk olan türk milletinin varlığını ortadan kaldırm
a çabasıdır bu işin sonudur çünkü şu anda kendisiyle benzeşen anamuhalefetle anlaşıp anayasadan türk milletinin adını çıkarırla
rsa yapılacak bir şey kalmaz meclis sayısal çoğunlukla yönetiliyor bunlar sinsi her işi alttan alttan götürüyorlar böyle bir ad
```

In [6]:
```
df["category"].value_counts()
```

Out[6]:
```
saglik        700
teknoloji     700
dunya         700
spor          700
siyaset       700
ekonomi       700
kultur        700
Name: category, dtype: int64
```

In [73]:
```
replace({"spor": 1, "kultur": 2,  "saglik": 3, "ekonomi": 4, "dunya": 5, "teknoloji": 6, "siyaset": 7}, inplace=True, regex=True)
```

In [8]:
```
df["category"]
```

Out[8]:
```
0       7
1       7
2       7
3       7
4       7
       ..
4895    6
4896    6
4897    6
4898    6
4899    6
Name: category, Length: 4900, dtype: int64
```

In [9]:
```
df
```

Out[9]:

	category	text
0	7	3 milyon ile ön seçim vaadi mhp nin 10 olağan...
1	7	mesut_yılmaz yüce_divan da ceza alabilirdi pr...
2	7	disko lar kaldırılıyor başbakan_yardımcısı ar...
3	7	sarıgül anayasa_mahkemesi ne gidiyor mustafa_...
4	7	erdoğan idamın bir haklılık sebebi var demek ...
...	...	...
4895	6	iphone lara geri dönüyor ios 6 sürümüyle tele...
4896	6	muslukta devrim sadece elimizi yıkadığımız mu...

In [10]:
```
df.text.isna().value_counts()
```

Out[10]:
```
False    4900
Name: text, dtype: int64
```

Preprocessing

In [11]:
```
import re
import inflect
import nltk
from nltk.corpus import stopwords
nltk.download('stopwords')
from nltk.stem import LancasterStemmer, WordNetLemmatizer

def to_lowercase(words):
    """Convert all characters to lowercase from list of tokenized words"""
    new_words = []
    for word in words:
        new_word = word.lower()
        new_words.append(new_word)
    return new_words

def remove_punctuation(words):
    """Remove punctuation from list of tokenized words"""
    new_words = []
    for word in words:
        new_word = re.sub(r'[^\w\s]', '', word)
        if new_word != '':
            new_words.append(new_word)
    return new_words

def replace_numbers(words):
    """Replace all interger occurrences in list of tokenized words with textual representation"""
    p = inflect.engine()
    new_words = []
    for word in words:
        if word.isdigit():
            new_word = p.number_to_words(word)
            new_words.append(new_word)
        else:
            new_words.append(word)
    return new_words

def remove_stopwords(words):
    """Remove stop words from list of tokenized words"""
    new_words = []
    for word in words:
        if word not in stopwords.words('english'):
            new_words.append(word)
    return new_words

def lemmatize_verbs(words):
    """Lemmatize verbs in list of tokenized words"""
    lemmatizer = WordNetLemmatizer()
    lemmas = []
    for word in words:
        lemma = lemmatizer.lemmatize(word, pos='v')
        lemmas.append(lemma)
    return lemmas

def tokens_to_sentence(words):
    return ' '.join(words)
```

In [12]:
```
from nltk import word_tokenize
X = df["text"].apply(word_tokenize)

print("Lowercasing")
X = X.apply(to_lowercase)

print("Punctuation removal imminent")
X = X.apply(remove_punctuation)

#CANT REPLACE NUMBERS DUE TO LANGUAGE
#print("Replacing numbers")
#X = X.apply(replace_numbers)

#CANT REMOVE STOPWORDS DUE TO LANGUAGE
#print("Removing stopwords")
#X = X.apply(remove_stopwords)

#CANT LEMMETIZE DUE TO LANGUAGE
#print("Lemme lemme lemme a man after midnight")
#X = X.apply(lemmatize_verbs)

print("Old sentence with a new look...")
X = X.apply(tokens_to_sentence)
```

```
Lowercasing
Punctuation removal imminent
Old sentence with a new look...
```

Zemberek Preprocessing killed my computer due to RAM issues

In [13]:
```
#from zemberek import TurkishSentenceNormalizer, TurkishMorphology, TurkishTokenizer

#morphology = TurkishMorphology.create_with_defaults()
#normalizer = TurkishSentenceNormalizer(morphology)

#tokenizer = TurkishTokenizer.DEFAULT

#df.text = df.text.apply(normalizer.normalize)
#for index, row in enumerate(df.text):
#    new_row = []
#    tokens = tokenizer.tokenize(row)
#    for token in tokens:
#        new_row.append(token.content)
#    df.text[index] = new_row
```

LSA

In [14]:
```
from sklearn.feature_extraction.text import CountVectorizer

vectorizer = CountVectorizer(min_df=1)

bag_of_words = vectorizer.fit_transform(X)
```

In [15]:
```
from sklearn.decomposition import TruncatedSVD

svd = TruncatedSVD(n_components=7)

lsa = svd.fit_transform(bag_of_words)
```

In [17]:
```
sample = X.sample(5)
sample_indices = sample.index

topic_encoded_df = pd.DataFrame(lsa, columns = ["topic_1", "topic_2", "topic_3", "topic_4", "topic_5", "topic_6", "topic_7"])

lsa_df = topic_encoded_df.copy()

topic_encoded_df['sentence'] = df.text

topic_encoded_df['truth'] = df.category

display(topic_encoded_df.iloc[sample_indices])
```

	topic_1	topic_2	topic_3	topic_4	topic_5	topic_6	topic_7	sentence	truth
1529	2.991316	0.367396	0.220165	0.340752	-0.927359	0.967119	-1.710103	euro / dolar 1 2820 direncinin üzerine çıktı ...	4
40	27.147343	14.646311	-0.586750	1.741561	-1.313818	-4.255911	-6.743409	başbakan gündem mühendisi başbakan bir idam s...	7
2444	4.866691	1.954087	-0.777286	0.742119	1.065964	-3.287606	-0.438132	temel üç güdü sahnede theatron tiyatrosu teme...	2
826	5.051529	-3.614992	-0.952048	-1.274786	0.451511	0.412036	-0.832524	ab den iran gazına yasak ab dışişleri bakanla...	5
2971	21.316231	-2.483473	-3.255836	1.019556	-1.362750	-3.056650	-1.951467	yarın tüm türkiye de hastaneler eylemde geçti...	3

In [18]:
```
lsa_df['topic_1'] = np.abs(lsa_df['topic_1'])
lsa_df['topic_2'] = np.abs(lsa_df['topic_2'])
lsa_df['topic_3'] = np.abs(lsa_df['topic_3'])
lsa_df['topic_4'] = np.abs(lsa_df['topic_4'])
lsa_df['topic_5'] = np.abs(lsa_df['topic_5'])
lsa_df['topic_6'] = np.abs(lsa_df['topic_6'])
lsa_df['topic_7'] = np.abs(lsa_df['topic_7'])
```

In [19]:
```
lsa_df
```

Out[19]:

	topic_1	topic_2	topic_3	topic_4	topic_5	topic_6	topic_7
0	34.522715	14.868083	2.723825	5.146039	11.650177	1.225394	5.788550
1	14.308660	5.736456	0.538176	1.528043	2.188375	2.043363	4.027517
2	33.688237	4.996255	0.515829	3.870347	1.121752	2.459077	8.480376
3	2.846591	0.946266	0.649103	0.104401	0.099298	0.968713	0.771294
4	9.471025	5.475884	0.576090	0.179237	4.148684	1.418151	0.101522
...	...	...	...	...	...	...	...
4895	4.857378	0.942469	0.263947	0.398643	0.058440	0.998750	1.814951
4896	4.125957	0.263710	0.657029	1.014527	0.004868	0.821150	0.128186
4897	24.707777	12.128305	6.739085	0.933470	0.777224	8.421262	2.400687
4898	11.262924	3.205489	0.140360	3.196213	0.143032	0.215460	1.146453
4899	5.987438	1.055281	1.029678	0.920349	0.855992	0.088472	0.328614

4900 rows × 7 columns

Training

In [20]:
```
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
```

In [21]:
```
X_lsa = lsa_df
y_lsa = df.category

print(X_lsa.shape)
y_lsa.shape
```

```
(4900, 7)
```

Out[21]:
```
(4900,)
```

ekonomi 1
spor 2
kultur 3
saglik 4
teknoloji 5
siyaset 6
dunya 7

In [22]:
```
X_lsa = lsa_df
y_lsa = df.category

X_train, X_test, y_train, y_test = train_test_split(X_lsa, y_lsa, test_size=0.25, random_state=23, stratify=y_lsa)

print("---Naive Bayes---")

nb_model = MultinomialNB()

parameters = {"alpha": [0, 0.1, 0.01, 0.001, 0.0001, 0.00001, 1, 10, 100, 1000, 10000],
              "fit_prior": [True, False]}

scoring = ["f1_micro"]
grid_search_nb = GridSearchCV(estimator=nb_model, param_grid=parameters, cv=5, refit="f1_micro", scoring=scoring)

grid_search_nb = grid_search_nb.fit(X_train, y_train)

print("Best Score: ",grid_search_nb.best_score_)
print("Best Estimator: ", grid_search_nb.best_estimator_)
print("Best Parameters: ", grid_search_nb.best_params_)
print()

y_pred = grid_search_nb.best_estimator_.predict(X_test)
print("Classification Report")
print(classification_report(y_test, y_pred))
print()

conf_matrix = confusion_matrix(y_pred, y_test)
print(conf_matrix)
print()
```

```
print("---Random Forest---")

rf_model = RandomForestClassifier(random_state=23)

parameters = {"n_estimators":[10*x for x in range(1, 11)],
              "criterion": ["gini", "entropy"],
               "max_features": ["auto", "sqrt", "log2", None]}

scoring = ["f1_micro"]
grid_search_rf = GridSearchCV(estimator=rf_model, param_grid=parameters, cv=5, refit="f1_micro", scoring=scoring)

grid_search_rf = grid_search_rf.fit(X_train, y_train)

print("Best Score: ",grid_search_rf.best_score_)
print("Best Estimator: ", grid_search_rf.best_estimator_)
print("Best Parameters: ", grid_search_rf.best_params_)
print()

y_pred = grid_search_rf.best_estimator_.predict(X_test)
print("Classification Report")
print(classification_report(y_test, y_pred))
print()

conf_matrix = confusion_matrix(y_pred, y_test)
print(conf_matrix)
print()
```

```python
print("---Logistic Regression---")

lr_model = LogisticRegression(random_state=23)

parameters = {"penalty": ["l1", "l2", "elasticnet", None],
              "solver": ["newton-cg", "lbfgs", "liblinear", "sag", "saga"],
               "multi_class" :["auto", "ovr", "multinomial"]}

scoring = ["f1_micro"]
grid_search_lr = GridSearchCV(estimator=lr_model, param_grid=parameters, cv=5, refit="f1_micro", scoring=scoring)

grid_search_lr = grid_search_lr.fit(X_train, y_train)

print("Best Score: ",grid_search_lr.best_score_)
print("Best Estimator: ", grid_search_lr.best_estimator_)
print("Best Parameters: ", grid_search_lr.best_params_)
print()

y_pred = grid_search_lr.best_estimator_.predict(X_test)
print("Classification Report")
print(classification_report(y_test, y_pred))
print()

conf_matrix = confusion_matrix(y_pred, y_test)
print(conf_matrix)
```

```
---Naive Bayes---
Best Score:  0.28190476190476194
Best Estimator:  MultinomialNB(alpha=100)
Best Parameters:  {'alpha': 100, 'fit_prior': True}

Classification Report
              precision    recall  f1-score   support

           1       0.29      0.34      0.31       175
           2       0.27      0.14      0.18       175
           3       0.29      0.30      0.29       175
           4       0.37      0.41      0.39       175
           5       0.29      0.33      0.31       175
           6       0.21      0.06      0.09       175
           7       0.21      0.37      0.27       175

    accuracy                           0.28      1225
   macro avg       0.28      0.28      0.26      1225
weighted avg       0.28      0.28      0.26      1225


[[60 40 10 15 36 21 26]
 [10 24 15  4 10 18  8]
 [14 32 52 19 15 31 19]
 [16 15 39 71 16 22 11]
 [21 17 13 23 57 27 38]
 [ 7  7  5  6  5 10  8]
 [47 40 41 37 36 46 65]]
```

```
---Random Forest---
Best Score:  0.4010884353741496
Best Estimator:  RandomForestClassifier(n_estimators=80, random_state=23)
Best Parameters:  {'criterion': 'gini', 'max_features': 'auto', 'n_estimators': 80}

Classification Report
              precision    recall  f1-score   support

           1       0.35      0.46      0.40       175
           2       0.47      0.43      0.45       175
           3       0.49      0.58      0.53       175
           4       0.56      0.44      0.49       175
           5       0.33      0.29      0.31       175
           6       0.36      0.33      0.34       175
           7       0.41      0.43      0.42       175

    accuracy                           0.42      1225
   macro avg       0.43      0.42      0.42      1225
weighted avg       0.43      0.42      0.42      1225


[[ 80  24  10  18  50  32  15]
 [ 21  76  13  11  10  16  14]
 [ 10  19 102  22  13  19  22]
 [  5  11   7  77  17   9  12]
 [ 18  12  11   9  50  30  20]
 [ 25  15  19   8  18  57  17]
 [ 16  18  13  30  17  12  75]]

 ---Logistic Regression---
 Best Score:  0.30448979591836733
 Best Estimator:  LogisticRegression(multi_class='ovr', penalty='l1', random_state=23,
                    solver='saga')
 Best Parameters:  {'multi_class': 'ovr', 'penalty': 'l1', 'solver': 'saga'}

 Classification Report
               precision    recall  f1-score   support

            1       0.31      0.46      0.37       175
            2       0.31      0.12      0.17       175
            3       0.33      0.31      0.32       175
            4       0.38      0.37      0.37       175
            5       0.25      0.39      0.30       175
            6       0.25      0.14      0.18       175
            7       0.32      0.36      0.34       175

     accuracy                           0.31      1225
    macro avg       0.31      0.31      0.29      1225
 weighted avg       0.31      0.31      0.29      1225


 [[81 42 19 17 51 35 19]
  [ 4 21 11  9  6  7 10]
  [ 5 31 54 18 13 21 20]
  [ 7 17 41 64 11 18 11]
  [35 28 16 30 68 51 43]
  [22 14  8 10 10 24  9]
  [21 22 26 27 16 19 63]]
```

Word2Vec

In [23]:
```
X = X.apply(word_tokenize)
```

In [24]:
```
model = gensim.models.Word2Vec(window=10, workers=4)
model.build_vocab(X)
model.train(X, total_examples=model.corpus_count, epochs=model.epochs)
```

Out[24]:
```
(5383184, 6453300)
```

In [25]:
```
model.wv["ankara"]
```

Out[25]:
```
array([-1.338236  ,  2.320235  ,  0.7993745 ,  1.684368  , -0.8946845 ,
       -0.95827824, -2.2457924 ,  1.5763606 , -0.4371776 , -1.1859081 ,
        0.6104842 ,  0.36811218,  0.641078  ,  0.25754428,  0.9790532 ,
        0.21832757,  1.3240594 , -1.0630761 ,  1.2308713 ,  1.2841293 ,
        1.864199  , -0.16160823,  0.08097138,  0.39217117, -0.8819247 ,
        0.77865326, -0.50923043,  0.01727713, -1.577542  , -1.0347171 ,
       -0.08266636,  1.0945752 , -0.11743616, -0.13604677, -0.1764687 ,
       -0.04576231, -1.4493245 , -0.1542116 , -0.69809675,  1.9418875 ,
        0.319451  ,  0.36457565,  0.14798582, -0.5371393 ,  1.593182  ,
       -1.9530717 , -0.72596914,  0.5217091 ,  0.9486239 , -0.7929973 ,
        0.94923884,  1.8964986 ,  0.06160415,  0.51587945,  0.00982273,
        0.42907628,  2.2941885 , -1.2687217 , -1.5964487 ,  0.5410135 ,
        0.6002015 ,  0.3769262 ,  0.57496667, -0.8545158 , -3.0471327 ,
        0.2746317 , -0.28079933, -0.13198456,  1.384139  , -1.1575696 ,
        1.2375388 , -1.123738  , -1.551804  ,  0.30932403,  1.2477862 ,
        2.3068728 , -0.87432617,  2.0488634 , -1.5680919 ,  1.1250057 ,
       -0.19499946,  1.3605856 ,  0.5035367 ,  2.1578598 ,  0.3613399 ,
       -0.18113275,  1.0108398 ,  1.1583288 ,  1.4026005 , -0.36857116,
        0.80243194,  0.17391683, -0.05185337,  2.0264812 ,  0.5451321 ,
        0.30973637,  1.3839206 ,  0.7845757 , -1.0277009 , -0.3847008 ],
      dtype=float32)
```

In [26]:
```
model.wv["ankara"].shape
```

Out[26]:
```
(100,)
```

In [27]:
```
w2v_df = pd.DataFrame((df.text.copy()))
```

In [28]:
```
X
```

Out[28]:
```
0       [3, milyon, ile, ön, seçim, vaadi, mhp, nin, 1...
1       [mesut_yılmaz, yüce_divan, da, ceza, alabilird...
2       [disko, lar, kaldırılıyor, başbakan_yardımcısı...
3       [sarıgül, anayasa_mahkemesi, ne, gidiyor, must...
4       [erdoğan, idamın, bir, haklılık, sebebi, var, ...
                              ...
4895    [iphone, lara, geri, dönüyor, ios, 6, sürümüyl...
4896    [muslukta, devrim, sadece, elimizi, yıkadığımı...
4897    [halka, iyi, anlatılmalı, bilgi_teknolojileri,...
4898    [çöpe, gidiyorlar, apple, 775, bin, uygulamayl...
4899    [google, bu, kez, edward_gorey, dedi, google, ...
Name: text, Length: 4900, dtype: object
```

In [29]:
```
number_deleted_total = 0
```

In [42]:
```
number_deleted = 0
for row in X:
    for word in row:
        if not model.wv.has_index_for(word):
            row.remove(word)
            number_deleted += 1
            number_deleted_total += 1

number_deleted
```

Out[42]: 1

In [43]:
```
number_deleted_total
```

Out[43]: 139688

In [44]:
```
w2v_list = []
for row in X:
    a = [[model.wv[word] for word in row]]
    a = np.sum(a, axis=1)
    a = np.ravel(a)
    w2v_list.append(a)


len(w2v_list)
```

Out[44]: 4900

In [45]:
```
w2v_df
```

Out[45]:

	text
0	3 milyon ile ön seçim vaadi mhp nin 10 olağan...
1	mesut_yılmaz yüce_divan da ceza alabilirdi pr...
2	disko lar kaldırılıyor başbakan_yardımcısı ar...
3	sarıgül anayasa_mahkemesi ne gidiyor mustafa_...
4	erdoğan idamın bir haklılık sebebi var demek ...
...	...
4895	iphone lara geri dönüyor ios 6 sürümüyle tele...
4896	muslukta devrim sadece elimizi yıkadığımız mu...
4897	halka iyi anlatılmalı bilgi_teknolojileri ile...
4898	çöpe gidiyorlar apple 775 bin uygulamayla app...
4899	google bu kez edward_gorey dedi ! google bu k...

4900 rows × 1 columns

In [46]:
```
w2v_df["value"] = w2v_list
```

In [47]:
```
w2v_df["truth"] = df.category
```

In [48]:
```
w2v_df.drop(["text"], axis=1, inplace=True)
```

In [49]:
```
w2v_df
```

Out[49]:

	value	truth
0	[-45.66982, 177.33319, 112.476166, 179.04965, ...	7
1	[-102.909195, 205.68434, 39.197083, 104.91502,...	7
2	[-141.37473, 227.56273, 65.13089, 180.54752, 1...	7
3	[-13.52038, 47.48405, 23.732637, 19.021654, -1...	7
4	[-7.079339, 141.49669, 68.54283, 43.929302, 10...	7
...	...	...
4895	[-22.166348, 78.21562, 12.994923, 21.11129, 15...	6
4896	[-10.839109, 14.878312, 16.388739, 20.717709, ...	6
4897	[-45.750362, 191.0645, 105.712326, 99.840935, ...	6
4898	[-65.943794, 90.25321, 9.466799, 28.564238, 50...	6
4899	[-20.573933, 33.175762, 0.2995367, 22.287855, ...	6

4900 rows × 2 columns

In [50]:
```
columns = range(100)
X_w2v = pd.DataFrame()

for row in w2v_df.value.values:
    df = pd.DataFrame(np.abs(row.reshape(-1, len(row))), columns=columns)
    X_w2v = X_w2v.append(df, ignore_index=True)

y_w2v = w2v_df.truth

X_train, X_test, y_train, y_test = train_test_split(X_w2v, y_w2v, test_size=0.25, random_state=23, stratify=y_w2v)

print("---Naive Bayes---")

nb_model = MultinomialNB()

parameters = {"alpha": [0, 0.1, 0.01, 0.001, 0.0001, 0.00001, 1, 10, 100, 1000, 10000],
              "fit_prior": [True, False]}

scoring = ["f1_micro"]
grid_search_nb = GridSearchCV(estimator=nb_model, param_grid=parameters, cv=5, refit="f1_micro", scoring=scoring)

grid_search_nb = grid_search_nb.fit(X_train, y_train)

print("Best Score: ",grid_search_nb.best_score_)
print("Best Estimator: ", grid_search_nb.best_estimator_)
print("Best Parameters: ", grid_search_nb.best_params_)
print()

y_pred = grid_search_nb.best_estimator_.predict(X_test)
print("Classification Report")
print(classification_report(y_test, y_pred))
print()

conf_matrix = confusion_matrix(y_pred, y_test)
print(conf_matrix)
print()
```

```
print("---Random Forest---")

rf_model = RandomForestClassifier(random_state=23)

parameters = {"n_estimators":[10*x for x in range(1, 11)],
              "criterion": ["gini", "entropy"],
               "max_features": ["auto", "sqrt", "log2", None]}

scoring = ["f1_micro"]
grid_search_rf = GridSearchCV(estimator=rf_model, param_grid=parameters, cv=5, refit="f1_micro", scoring=scoring)

grid_search_rf = grid_search_rf.fit(X_train, y_train)

print("Best Score: ",grid_search_rf.best_score_)
print("Best Estimator: ", grid_search_rf.best_estimator_)
print("Best Parameters: ", grid_search_rf.best_params_)
print()

y_pred = grid_search_rf.best_estimator_.predict(X_test)
print("Classification Report")
print(classification_report(y_test, y_pred))
print()

conf_matrix = confusion_matrix(y_pred, y_test)
print(conf_matrix)
print()
```

```
print("---Logistic Regression---")

lr_model = LogisticRegression(random_state=23)

parameters = {"penalty": ["l1", "l2", "elasticnet", None],
              "solver": ["newton-cg", "lbfgs", "liblinear", "sag", "saga"],
               "multi_class" :["auto", "ovr", "multinomial"]}

scoring = ["f1_micro"]
grid_search_lr = GridSearchCV(estimator=lr_model, param_grid=parameters, cv=5, refit="f1_micro", scoring=scoring)

grid_search_lr = grid_search_lr.fit(X_train, y_train)

print("Best Score: ",grid_search_lr.best_score_)
print("Best Estimator: ", grid_search_lr.best_estimator_)
print("Best Parameters: ", grid_search_lr.best_params_)
print()

y_pred = grid_search_lr.best_estimator_.predict(X_test)
print("Classification Report")
print(classification_report(y_test, y_pred))
print()

conf_matrix = confusion_matrix(y_pred, y_test)
print(conf_matrix)
```

```
---Naive Bayes---
Best Score:  0.5229931972789116
Best Estimator:  MultinomialNB(alpha=0)
Best Parameters:  {'alpha': 0, 'fit_prior': True}

Classification Report
              precision    recall  f1-score   support

           1       0.54      0.48      0.51       175
           2       0.50      0.54      0.52       175
           3       0.77      0.83      0.80       175
           4       0.57      0.42      0.48       175
           5       0.38      0.47      0.42       175
           6       0.53      0.41      0.46       175
           7       0.49      0.59      0.53       175

    accuracy                           0.53      1225
   macro avg       0.54      0.53      0.53      1225
weighted avg       0.54      0.53      0.53      1225


[[ 84  15   1  24  15  12   5]
 [ 29  95   2  16  15  23  11]
 [  3   6 145   8   3  20   4]
 [  2  12   3  73  18  19   2]
 [ 18  22   6  23  82  17  46]
 [  6  13  12  14  14  72   4]
 [ 33  12   6  17  28  12 103]]
```

```
---Random Forest---
Best Score:  0.676734693877551
Best Estimator:  RandomForestClassifier(criterion='entropy', max_features='log2',
                       random_state=23)
Best Parameters:  {'criterion': 'entropy', 'max_features': 'log2', 'n_estimators': 100}

Classification Report
              precision    recall  f1-score   support

           1       0.70      0.67      0.69       175
           2       0.67      0.69      0.68       175
           3       0.82      0.89      0.85       175
           4       0.69      0.61      0.65       175
           5       0.60      0.57      0.59       175
           6       0.65      0.61      0.63       175
           7       0.65      0.75      0.70       175

    accuracy                           0.68      1225
   macro avg       0.68      0.68      0.68      1225
weighted avg       0.68      0.68      0.68      1225


[[118  10   1   9  13   9   9]
 [ 14 120   2  12  11  18   3]
 [  3   2 155  13   3  10   4]
 [  5   6   4 107  11  15   7]
 [ 13  12   0  12 100  13  16]
 [  6  16   8  10  12 106   4]
 [ 16   9   5  12  25   4 132]]

---Logistic Regression---
Best Score:  0.6821768707482994
Best Estimator:  LogisticRegression(multi_class='ovr', random_state=23)
Best Parameters:  {'multi_class': 'ovr', 'penalty': 'l2', 'solver': 'lbfgs'}

Classification Report
              precision    recall  f1-score   support

           1       0.75      0.74      0.74       175
           2       0.69      0.71      0.70       175
           3       0.82      0.87      0.84       175
           4       0.64      0.57      0.60       175
           5       0.67      0.69      0.68       175
           6       0.64      0.62      0.63       175
           7       0.74      0.76      0.75       175

    accuracy                           0.71      1225
   macro avg       0.70      0.71      0.71      1225
weighted avg       0.70      0.71      0.71      1225


[[130  15   1  13   6   4   5]
 [ 12 124   2  14   7  18   3]
 [  2   4 152  10   3  10   5]
 [  9   6   4  99  13  12  12]
 [  9   7   2   9 120  19  14]
 [  3  16  11  19  10 108   3]
 [ 10   3   3  11  16   4 133]]
```

Concat

In [77]:
```
lsa2vec_df_concat = pd.concat([X_w2v, lsa_df], axis=1)
```

In [78]:
```
lsa2vec_df_concat
```

Out[78]:

	0	1	2	3	4	5	6	7	8	9	...	97	98	
0	45.669819	177.333191	112.476166	179.049652	213.833130	554.619263	142.828247	499.833832	460.745483	360.058807	...	342.648560	73.756104	78.5
1	102.909195	205.684341	39.197083	104.915024	58.167950	276.610504	53.507935	278.002014	184.458939	148.761108	...	107.689728	44.190434	15.9
2	141.374725	227.562729	65.130890	180.547516	138.122131	488.909058	48.034718	472.784912	397.700439	238.485489	...	253.774704	33.925842	51.5
3	13.520380	47.484051	23.732637	19.021654	10.160637	47.409710	14.027145	49.491959	29.601799	31.639366	...	26.784817	5.468457	3.0
4	7.079339	141.496689	68.542831	43.929302	109.005547	142.815674	66.391823	148.617554	144.017136	151.795929	...	118.740944	3.570172	8.2
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
4895	22.166348	78.215622	12.994923	21.111290	15.980445	74.171555	24.032709	83.377495	37.355606	41.608051	...	21.597710	8.495565	16.0
4896	10.839109	14.878312	16.388739	20.717709	17.206287	35.527596	23.257305	55.227898	27.033142	6.009076	...	35.636265	17.940058	8.6
4897	45.750362	191.064499	105.712326	99.840935	172.052963	403.346222	75.769691	494.088623	280.160645	240.602707	...	284.014923	26.999891	11.7
4898	65.943794	90.253212	9.466799	28.564238	50.089211	114.221153	56.694481	174.816650	57.518478	64.720276	...	80.608765	45.066814	6.3
4899	20.573933	33.175762	0.299537	22.287855	10.079779	53.064453	29.580824	60.291325	35.644905	24.377754	...	22.937803	23.607281	0.3

4900 rows × 107 columns

In [79]:
```
X_lsa2vec_concat = lsa2vec_df_concat
y_lsa2vec_concat = df.category

X_train, X_test, y_train, y_test = train_test_split(X_lsa2vec_concat, y_lsa2vec_concat, test_size=0.25, random_state=23, stratify

print("---Naive Bayes---")

nb_model = MultinomialNB()

parameters = {"alpha": [0, 0.1, 0.01, 0.001, 0.0001, 0.00001, 1, 10, 100, 1000, 10000],
              "fit_prior": [True, False]}

scoring = ["f1_micro"]
grid_search_nb = GridSearchCV(estimator=nb_model, param_grid=parameters, cv=5, refit="f1_micro", scoring=scoring)

grid_search_nb = grid_search_nb.fit(X_train, y_train)

print("Best Score: ",grid_search_nb.best_score_)
print("Best Estimator: ", grid_search_nb.best_estimator_)
print("Best Parameters: ", grid_search_nb.best_params_)
print()

y_pred = grid_search_nb.best_estimator_.predict(X_test)
print("Classification Report")
print(classification_report(y_test, y_pred))
print()

conf_matrix = confusion_matrix(y_pred, y_test)
print(conf_matrix)
print()
```

```
print("---Random Forest---")

rf_model = RandomForestClassifier(random_state=23)

parameters = {"n_estimators":[10*x for x in range(1, 11)],
              "criterion": ["gini", "entropy"],
               "max_features": ["auto", "sqrt", "log2", None]}

scoring = ["f1_micro"]
grid_search_rf = GridSearchCV(estimator=rf_model, param_grid=parameters, cv=5, refit="f1_micro", scoring=scoring)

grid_search_rf = grid_search_rf.fit(X_train, y_train)

print("Best Score: ",grid_search_rf.best_score_)
print("Best Estimator: ", grid_search_rf.best_estimator_)
print("Best Parameters: ", grid_search_rf.best_params_)
print()

y_pred = grid_search_rf.best_estimator_.predict(X_test)
print("Classification Report")
print(classification_report(y_test, y_pred))
print()

conf_matrix = confusion_matrix(y_pred, y_test)
print(conf_matrix)
print()
```

```
print("---Logistic Regression---")

lr_model = LogisticRegression(random_state=23)

parameters = {"penalty": ["l1", "l2", "elasticnet", None],
              "solver": ["newton-cg", "lbfgs", "liblinear", "sag", "saga"],
               "multi_class" :["auto", "ovr", "multinomial"]}

scoring = ["f1_micro"]
grid_search_lr = GridSearchCV(estimator=lr_model, param_grid=parameters, cv=5, refit="f1_micro", scoring=scoring)

grid_search_lr = grid_search_lr.fit(X_train, y_train)

print("Best Score: ",grid_search_lr.best_score_)
print("Best Estimator: ", grid_search_lr.best_estimator_)
print("Best Parameters: ", grid_search_lr.best_params_)
print()

y_pred = grid_search_lr.best_estimator_.predict(X_test)
print("Classification Report")
print(classification_report(y_test, y_pred))
print()

conf_matrix = confusion_matrix(y_pred, y_test)
print(conf_matrix)
```

```
---Naive Bayes---
Best Score:  0.523265306122449
Best Estimator:  MultinomialNB(alpha=0)
Best Parameters:  {'alpha': 0, 'fit_prior': True}

Classification Report
              precision    recall  f1-score   support

           1       0.54      0.49      0.51       175
           2       0.50      0.54      0.52       175
           3       0.77      0.83      0.80       175
           4       0.57      0.42      0.48       175
           5       0.38      0.47      0.42       175
           6       0.53      0.41      0.46       175
           7       0.49      0.59      0.53       175

    accuracy                           0.53      1225
   macro avg       0.54      0.53      0.53      1225
weighted avg       0.54      0.53      0.53      1225


[[ 85  15   1  24  15  12   5]
 [ 28  95   2  16  15  23  11]
 [  3   6 145   8   3  20   4]
 [  2  12   3  73  18  19   2]
 [ 18  22   6  23  82  17  46]
 [  6  13  12  14  14  72   4]
 [ 33  12   6  17  28  12 103]]

---Random Forest---
Best Score:  0.6764625850340137
Best Estimator:  RandomForestClassifier(criterion='entropy', random_state=23)
Best Parameters:  {'criterion': 'entropy', 'max_features': 'auto', 'n_estimators': 100}

Classification Report
              precision    recall  f1-score   support

           1       0.75      0.67      0.70       175
           2       0.68      0.72      0.70       175
           3       0.83      0.90      0.86       175
           4       0.71      0.63      0.67       175
           5       0.57      0.58      0.58       175
           6       0.64      0.60      0.62       175
           7       0.70      0.78      0.74       175

    accuracy                           0.70      1225
   macro avg       0.70      0.70      0.70      1225
weighted avg       0.70      0.70      0.70      1225


[[117  11   2   7  11   4   5]
 [ 16 126   3  11  10  17   2]
 [  3   3 157   9   2  13   3]
 [  3   8   3 110  12  14   4]
 [ 16   7   1  17 102  18  17]
 [  6  14   6  11  15 105   7]
 [ 14   6   3  10  23   4 137]]
```

```
---Logistic Regression---
Best Score:  0.6821768707482994
Best Estimator:  LogisticRegression(penalty='l1', random_state=23, solver='liblinear')
Best Parameters:  {'multi_class': 'auto', 'penalty': 'l1', 'solver': 'liblinear'}

Classification Report
              precision    recall  f1-score   support

           1       0.71      0.77      0.74       175
           2       0.68      0.74      0.70       175
           3       0.82      0.89      0.85       175
           4       0.63      0.53      0.57       175
           5       0.62      0.67      0.64       175
           6       0.64      0.60      0.62       175
           7       0.75      0.68      0.71       175

    accuracy                           0.70      1225
   macro avg       0.69      0.70      0.69      1225
weighted avg       0.69      0.70      0.69      1225


[[135  12   2  15   9   7   9]
 [ 10 129   3  16   8  19   6]
 [  0   5 155   9   4   8   7]
 [ 11   5   1  92  13  13  12]
 [ 10  10   3  11 117  20  18]
 [  3  11   9  22   9 105   4]
 [  6   3   2  10  15   3 119]]
```

Mixture

In [61]:
```
X_w2v
```

Out[61]:

	0	1	2	3	4	5	6	7	8	9	...	90	91	
0	45.669819	177.333191	112.476166	179.049652	213.833130	554.619263	142.828247	499.833832	460.745483	360.058807	...	385.030731	149.670166	53.
1	102.909195	205.684341	39.197083	104.915024	58.167950	276.610504	53.507935	278.002014	184.458939	148.761108	...	176.326706	111.144058	14.
2	141.374725	227.562729	65.130890	180.547516	138.122131	488.909058	48.034718	472.784912	397.700439	238.485489	...	356.659943	220.641403	53.
3	13.520380	47.484051	23.732637	19.021654	10.160637	47.409710	14.027145	49.491959	29.601799	31.639366	...	21.278189	11.307513	2.
4	7.079339	141.496689	68.542831	43.929302	109.005547	142.815674	66.391823	148.617554	144.017136	151.795929	...	139.585800	57.114563	13.
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
4895	22.166348	78.215622	12.994923	21.111290	15.980445	74.171555	24.032709	83.377495	37.355606	41.608051	...	62.885784	43.177895	12.
4896	10.839109	14.878312	16.388739	20.717709	17.206287	35.527596	23.257305	55.227898	27.033142	6.009076	...	36.075142	38.145763	21.
4897	45.750362	191.064499	105.712326	99.840935	172.052963	403.346222	75.769691	494.088623	280.160645	240.602707	...	391.744354	232.910126	53.
4898	65.943794	90.253212	9.466799	28.564238	50.089211	114.221153	56.694481	174.816650	57.518478	64.720276	...	145.268784	86.132904	71.
4899	20.573933	33.175762	0.299537	22.287855	10.079779	53.064453	29.580824	60.291325	35.644905	24.377754	...	42.341007	21.189827	22.

4900 rows × 100 columns

In [63]:
```
svd = TruncatedSVD(n_components=7)

lsa2vec_mix = svd.fit_transform(X_w2v)
```

In [68]:
```
sample = X.sample(5)
sample_indices = sample.index

topic_encoded_df_lsa2vec = pd.DataFrame(lsa2vec_mix, columns = ["topic_1", "topic_2", "topic_3", "topic_4", "topic_5", "topic_6",

lsa2vec_df_mix = topic_encoded_df_lsa2vec.copy()

topic_encoded_df_lsa2vec["text"] = df.text

topic_encoded_df_lsa2vec['truth'] = df.category

display(topic_encoded_df_lsa2vec.iloc[sample_indices])
```

	topic_1	topic_2	topic_3	topic_4	topic_5	topic_6	topic_7	text	truth
995	221.384644	8.093175	-63.376457	15.517049	2.382483	-5.057583	17.836981	israil in gazze saldırıları protesto edildi g...	dunya
3372	2565.373535	-782.349365	501.865356	-331.114258	281.645782	-10.544510	141.096222	yataktan kalktığımda doğrulamıyorum siz okurl...	saglik
816	555.097656	52.402260	-190.309113	-16.912708	21.720327	-10.108398	54.720783	sudan da silah fabrikasındaki patlama sudan y...	dunya
1707	176.891205	93.765366	45.595119	34.897636	-5.506378	-46.034103	11.900764	piyasa güne 11 3 milyar lira artı rezervle ba...	ekonomi
2869	831.594482	-207.254181	151.890717	-134.067688	-17.239511	-47.280457	-21.027920	prp yöntemiyle artık mümkün ! cilt lekelerind...	saglik

In [71]:
```
lsa2vec_df_mix['topic_1'] = np.abs(lsa2vec_df_mix['topic_1'])
lsa2vec_df_mix['topic_2'] = np.abs(lsa2vec_df_mix['topic_2'])
lsa2vec_df_mix['topic_3'] = np.abs(lsa2vec_df_mix['topic_3'])
lsa2vec_df_mix['topic_4'] = np.abs(lsa2vec_df_mix['topic_4'])
lsa2vec_df_mix['topic_5'] = np.abs(lsa2vec_df_mix['topic_5'])
lsa2vec_df_mix['topic_6'] = np.abs(lsa2vec_df_mix['topic_6'])
lsa2vec_df_mix['topic_7'] = np.abs(lsa2vec_df_mix['topic_7'])
```

In [72]:
```
lsa2vec_df_mix
```

Out[72]:

	topic_1	topic_2	topic_3	topic_4	topic_5	topic_6	topic_7
0	2273.973145	427.888855	309.966034	487.057434	32.530388	182.468689	178.353424
1	1070.090576	120.187210	88.462746	19.286659	175.532349	176.429199	121.758247
2	1905.773682	352.940369	286.417969	60.792442	143.357483	268.590607	157.654541
3	174.791916	12.495085	36.107422	15.599187	54.156883	31.149267	7.269622
4	731.507324	140.274139	15.928612	218.992889	120.008148	16.181915	27.191401
...	...	...	...	...	...	...	...
4895	328.621887	14.083620	6.269307	8.219036	87.813637	38.267254	8.126642
4896	227.998779	31.943708	60.276222	45.282696	5.625181	31.930780	16.066801
4897	1966.091919	320.745483	302.570404	197.897095	121.815132	36.076439	196.067551
4898	665.857666	38.254425	111.153313	22.916498	27.736742	87.808823	36.558674
4899	258.987427	25.909342	19.710543	4.359240	24.048349	52.576252	5.918579

4900 rows × 7 columns

In [80]:
```
X_lsa2vec_mix = lsa2vec_df_mix
y_lsa2vec_mix = df.category

X_train, X_test, y_train, y_test = train_test_split(X_lsa2vec_mix, y_lsa2vec_mix, test_size=0.25, random_state=23, stratify=y_lsa

print("---Naive Bayes---")

nb_model = MultinomialNB()

parameters = {"alpha": [0, 0.1, 0.01, 0.001, 0.0001, 0.00001, 1, 10, 100, 1000, 10000],
              "fit_prior": [True, False]}

scoring = ["f1_micro"]
grid_search_nb = GridSearchCV(estimator=nb_model, param_grid=parameters, cv=5, refit="f1_micro", scoring=scoring)

grid_search_nb = grid_search_nb.fit(X_train, y_train)

print("Best Score: ",grid_search_nb.best_score_)
print("Best Estimator: ", grid_search_nb.best_estimator_)
print("Best Parameters: ", grid_search_nb.best_params_)
print()

y_pred = grid_search_nb.best_estimator_.predict(X_test)
print("Classification Report")
print(classification_report(y_test, y_pred))
print()

conf_matrix = confusion_matrix(y_pred, y_test)
print(conf_matrix)
print()
```

```
print("---Random Forest---")

rf_model = RandomForestClassifier(random_state=23)

parameters = {"n_estimators":[10*x for x in range(1, 11)],
              "criterion": ["gini", "entropy"],
               "max_features": ["auto", "sqrt", "log2", None]}

scoring = ["f1_micro"]
grid_search_rf = GridSearchCV(estimator=rf_model, param_grid=parameters, cv=5, refit="f1_micro", scoring=scoring)

grid_search_rf = grid_search_rf.fit(X_train, y_train)

print("Best Score: ",grid_search_rf.best_score_)
print("Best Estimator: ", grid_search_rf.best_estimator_)
print("Best Parameters: ", grid_search_rf.best_params_)
print()

y_pred = grid_search_rf.best_estimator_.predict(X_test)
print("Classification Report")
print(classification_report(y_test, y_pred))
print()

conf_matrix = confusion_matrix(y_pred, y_test)
print(conf_matrix)
print()
```

```
print("---Logistic Regression---")

lr_model = LogisticRegression(random_state=23)

parameters = {"penalty": ["l1", "l2", "elasticnet", None],
              "solver": ["newton-cg", "lbfgs", "liblinear", "sag", "saga"],
               "multi_class" :["auto", "ovr", "multinomial"]}

scoring = ["f1_micro"]
grid_search_lr = GridSearchCV(estimator=lr_model, param_grid=parameters, cv=5, refit="f1_micro", scoring=scoring)

grid_search_lr = grid_search_lr.fit(X_train, y_train)

print("Best Score: ",grid_search_lr.best_score_)
print("Best Estimator: ", grid_search_lr.best_estimator_)
print("Best Parameters: ", grid_search_lr.best_params_)
print()

y_pred = grid_search_lr.best_estimator_.predict(X_test)
print("Classification Report")
print(classification_report(y_test, y_pred))
print()

conf_matrix = confusion_matrix(y_pred, y_test)
print(conf_matrix)
```

```
---Naive Bayes---
Best Score:  0.29687074829931975
Best Estimator:  MultinomialNB(alpha=1000)
Best Parameters:  {'alpha': 1000, 'fit_prior': True}

Classification Report
              precision    recall  f1-score   support

           1       0.24      0.40      0.30       175
           2       0.32      0.16      0.21       175
           3       0.52      0.61      0.56       175
           4       0.29      0.19      0.23       175
           5       0.32      0.48      0.39       175
           6       0.25      0.30      0.27       175
           7       0.40      0.12      0.18       175

    accuracy                           0.32      1225
   macro avg       0.33      0.32      0.31      1225
weighted avg       0.33      0.32      0.31      1225


[[ 70  54  14  52  35  38  25]
 [  8  28   5  16  12  11   8]
 [ 12  22 107  14  16  16  20]
 [ 15  22   8  34  10  15  14]
 [ 21  16  10  32  84  33  64]
 [ 44  31  24  20  16  53  23]
 [  5   2   7   7   2   9  21]]
```

```
---Random Forest---
Best Score:  0.45496598639455776
Best Estimator:  RandomForestClassifier(criterion='entropy', n_estimators=90, random_state=23)
Best Parameters:  {'criterion': 'entropy', 'max_features': 'auto', 'n_estimators': 90}

Classification Report
              precision    recall  f1-score   support

           1       0.44      0.41      0.42       175
           2       0.53      0.50      0.51       175
           3       0.63      0.73      0.68       175
           4       0.45      0.46      0.46       175
           5       0.41      0.41      0.41       175
           6       0.42      0.47      0.44       175
           7       0.49      0.41      0.44       175

    accuracy                           0.48      1225
   macro avg       0.48      0.48      0.48      1225
weighted avg       0.48      0.48      0.48      1225


[[ 72  18   4  21  17  24   9]
 [ 23  87   5  15  10  10  13]
 [ 13   9 128  15  10  10  17]
 [ 10  15   9  80  24  17  21]
 [ 19  10   6  12  72  25  32]
 [ 24  27  17  11  24  82  12]
 [ 14   9   6  21  18   7  71]]
```

```
---Logistic Regression---
Best Score:  0.33061224489795926
Best Estimator:  LogisticRegression(random_state=23, solver='newton-cg')
Best Parameters:  {'multi_class': 'auto', 'penalty': 'l2', 'solver': 'newton-cg'}

Classification Report
              precision    recall  f1-score   support

           1       0.27      0.43      0.33       175
           2       0.28      0.07      0.12       175
           3       0.55      0.51      0.53       175
           4       0.44      0.33      0.38       175
           5       0.30      0.57      0.39       175
           6       0.29      0.31      0.30       175
           7       0.38      0.18      0.25       175

    accuracy                           0.34      1225
   macro avg       0.36      0.34      0.33      1225
weighted avg       0.36      0.34      0.33      1225


[[76 59 21 44 30 33 20]
 [ 1 13  4 10  5  5  8]
 [10 19 89 13  2  7 21]
 [ 5 14 14 57  6 12 21]
 [41 34 13 25 99 59 63]
 [33 25 24 13 28 54 10]
 [ 9 11 10 13  5  5 32]]
```

e. Programmer Catalog

I've spent an hour for analysis and design, four to five hours for implementation and testing, two hours for reporting. One can reuse my code, especially the long code block with lots of GridSearchCV is pretty reusable as I reused it in different places in my code as well. All you need to do is to change the parameters in the beginning with your X (data) and your y (target values). Then, this code block will try to fit the best Naïve Bayes, Random Forest Classifier and Logistic Regression to your data. Also, preprocessing code blocks are reusable too. All the one need to the is to change the data parameter.

f. User Catalog

One can't use this code for other purposes as it is not intended to be used any other way. Or at least I don't have any other way in my mind.

V. Results, Discussion and Conclusion

I've used F1 as metric since it is a reliable parameter on classification tasks. Here is the results:

representation method / classifier	Naïve Bayes	Random Forest Classifier	Logistic Regression
LSA	0.281904	**0.401088**	0.304489
Word2Vec	0.522993	0.676734	**0.682176**
LSA2Vec-concatenation	0.523265	0.676462	**0.682176**
LSA2Vec-mixture	0.296870	**0.454965**	0.330612

As can be seen from the table, Logistic Regression with Word2Vec and LSA2Vec did the best, almost with the exact F1 score! I've double checked everything, and it seems like there is no problem. I think this is due to fact that there are a lot of columns in both these data and after some point, these representations do not represent these sentences well enough.