CUKUROVA UNIVERSITY FACULTY OF ENGINEERING – COMPUTER ENGINEERING

COURSE: INTRODUCTION TO DATA MINING

"Dimensionality Reduction, Feature Selection, and Classification with Sentiment Labelled Dataset"

2020555003-ÖMER AKDOĞAN

2020555046-AHMET ÖZALP

2020555060-ALİ TEPELİ

2020555071-EMİRHAN YILMAZOĞLU

2020555072-SEMİH ZENGİNOĞLU

DELIVERED DATE : 02.01.2025

ADVISER = PROF. DR. SELMA AYŞE ÖZEL

INTRODUCTION :

Sentiment analysis is a significant topic in natural language processing and machine learning, aiming to understand and classify emotional expressions in text. In this study, the Sentiment Labelled Sentences Dataset is utilized to examine the impact of different feature selection and dimensionality reduction methods on sentiment classification.

Text data is vectorized using the TF-IDF (Term Frequency-Inverse Document Frequency) method with both unigram and bigram representations. To reduce high dimensionality, methods such as Chi^2, PCA, Lasso (L1 Regularization), and threshold-based feature selection are applied. Finally, the processed datasets are classified using Naïve Bayes, Decision Tree, and SVM models. The performance of the reduced datasets is compared to that of the original, unreduced datasets.

LINK TO OUR DATASET :
https://archive.ics.uci.edu/dataset/331/sentiment+labelled+sentences

STEP 1- TF*IDF VECTORIZATION :

First, we combined three datasets into a single comprehensive dataset. Next, we applied unigram and bigram TF-IDF vectorization separately to this dataset. Afterward, we converted the datasets with unigram and bigram applied into Excel tables. The purpose of this step was to enable the use of dimensionality reduction methods, as these methods require the dataset to be numerically represented.

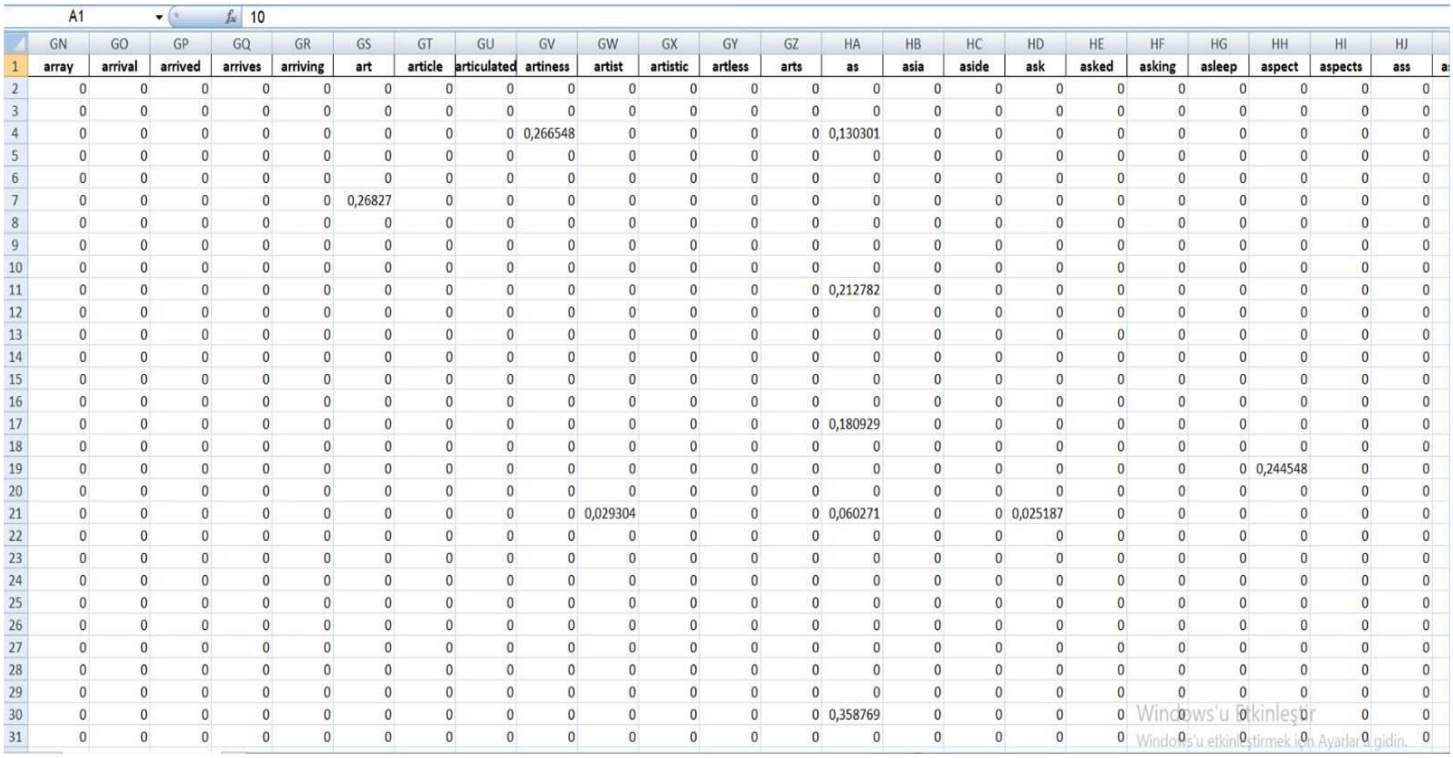
UNIGRAM - BIGRAM TF*IDF PYTHON CODE

tf-idf.py

```python
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer

# Veri setini okuma
file_name = 'merged_dataset.txt'  # Veri setinizin adı
output_folder = './'  # Çıktılar aynı klasöre kaydedilecek

# Veri setini pandas DataFrame olarak yükleme
data = pd.read_csv(file_name, sep='\t', header=None, names=['Text', 'Label'])

# CSV TF-IDF Kaydetme Fonksiyonu
def apply_tfidf_to_csv(data, ngram_range, output_file):
    vectorizer = TfidfVectorizer(ngram_range=ngram_range)
    tfidf_matrix = vectorizer.fit_transform(data['Text'])

    # TF-IDF sonuçlarını DataFrame'e dönüştürme
    tfidf_df = pd.DataFrame(tfidf_matrix.toarray(), columns=vectorizer.get_feature_names_out())

    # Label sütununu ekleme
    tfidf_df['Label'] = data['Label']

    # CSV olarak kaydetme
    tfidf_df.to_csv(output_file, index=False)
    print(f"TF-IDF ({ngram_range}) sonuçları kaydedildi: {output_file}")

# Unigram için TF-IDF CSV kaydetme
apply_tfidf_to_csv(data, (1, 1), f"{output_folder}tfidf_unigram.csv")

# Bigram için TF-IDF CSV kaydetme
apply_tfidf_to_csv(data, (2, 2), f"{output_folder}tfidf_bigram.csv")
```

RESULTS FOR TF*IDF VECTORIZATION :

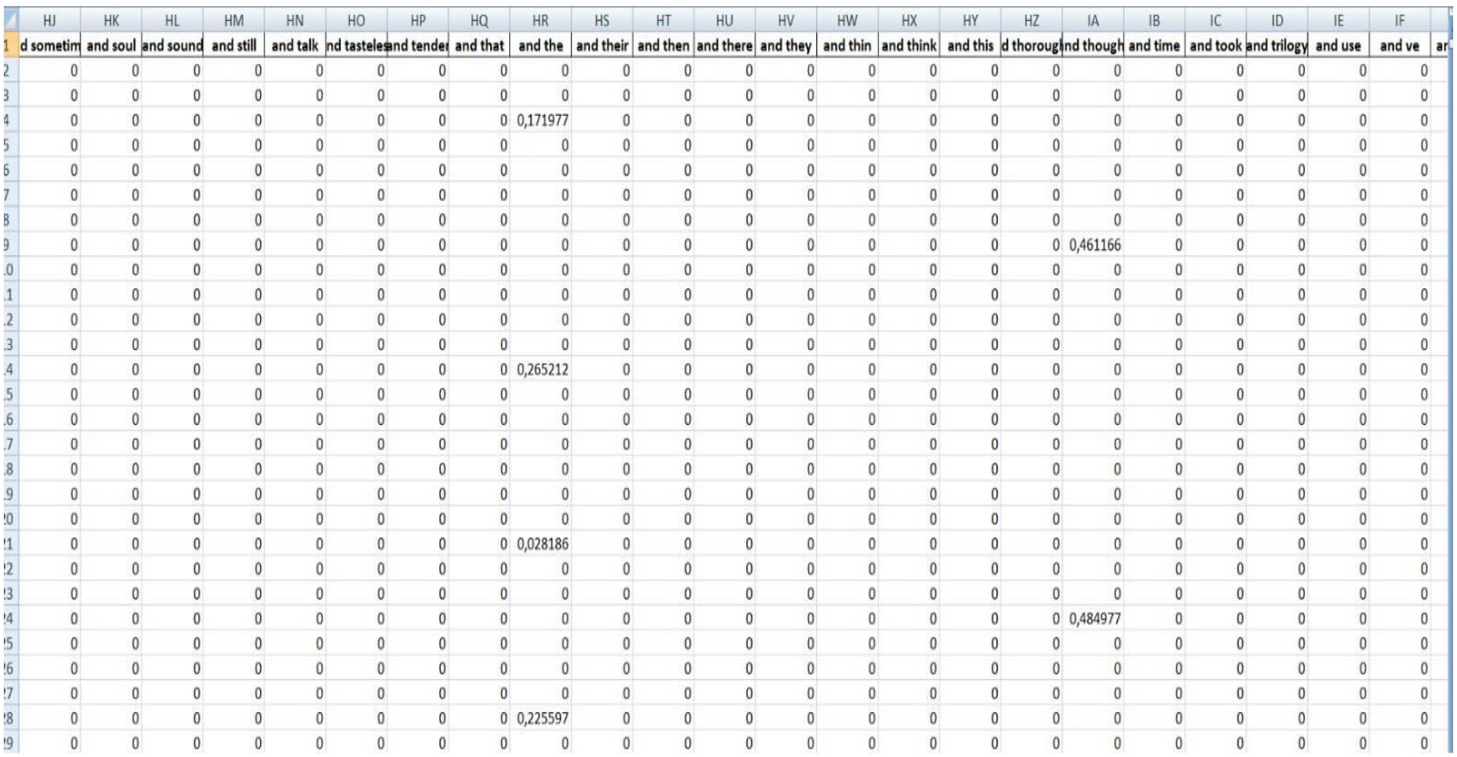
DATASET WITH UNIGRAM TF*IDF APPLIED . The number of unique unigrams (features) obtained: 5155.



A1 10

	GN	GO	GP	GQ	GR	GS	GT	GU	GV	GW	GX	GY	GZ	HA	HB	HC	HD	HE	HF	HG	HH	HI	HJ
1	array	arrival	arrived	arrives	arriving	art	article	articulated	artiness	artist	artistic	artless	arts	as	asia	aside	ask	asked	asking	asleep	aspect	aspects	ass
2	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
4	0	0	0	0	0	0	0	0	0,266548	0	0	0	0	0,130301	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0,26827	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0,212782	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0,180929	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0,244548	0	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0	0	0,029304	0	0	0	0,060271	0	0	0,025187	0	0	0	0	0	0
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
28	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
29	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
30	0	0	0	0	0	0	0	0	0	0	0	0	0	0,358769	0	0	0	0	0	0	0	0	0
31	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

DATASET WITH BIGRAM TF*IDF APPLIED . The number of unique bigrams (features) obtained: 20,402.



	HJ	HK	HL	HM	HN	HO	HP	HQ	HR	HS	HT	HU	HV	HW	HX	HY	HZ	IA	IB	IC	ID	IE	IF
1	d sometim	and soul	and sound	and still	and talk	nd tasteles	and tender	and that	and the	and their	and then	and there	and they	and thin	and think	and this	d thorough	nd though	and time	and took	and trilogy	and use	and ve
2	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
4	0	0	0	0	0	0	0	0	0,171977	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0,461166	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0,265212	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0	0,028186	0	0	0	0	0	0	0	0	0	0	0	0	0	0
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0,484977	0	0	0	0	0
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
28	0	0	0	0	0	0	0	0	0,225597	0	0	0	0	0	0	0	0	0	0	0	0	0	0
29	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

STEP 2- REDUCTION :

At this step , we applied four (chi_square, pca , l1 _regularization, variance threshold) dimensionality reduction methods to the vectorized dataset to reduce its size.

Chi_square reduction method :

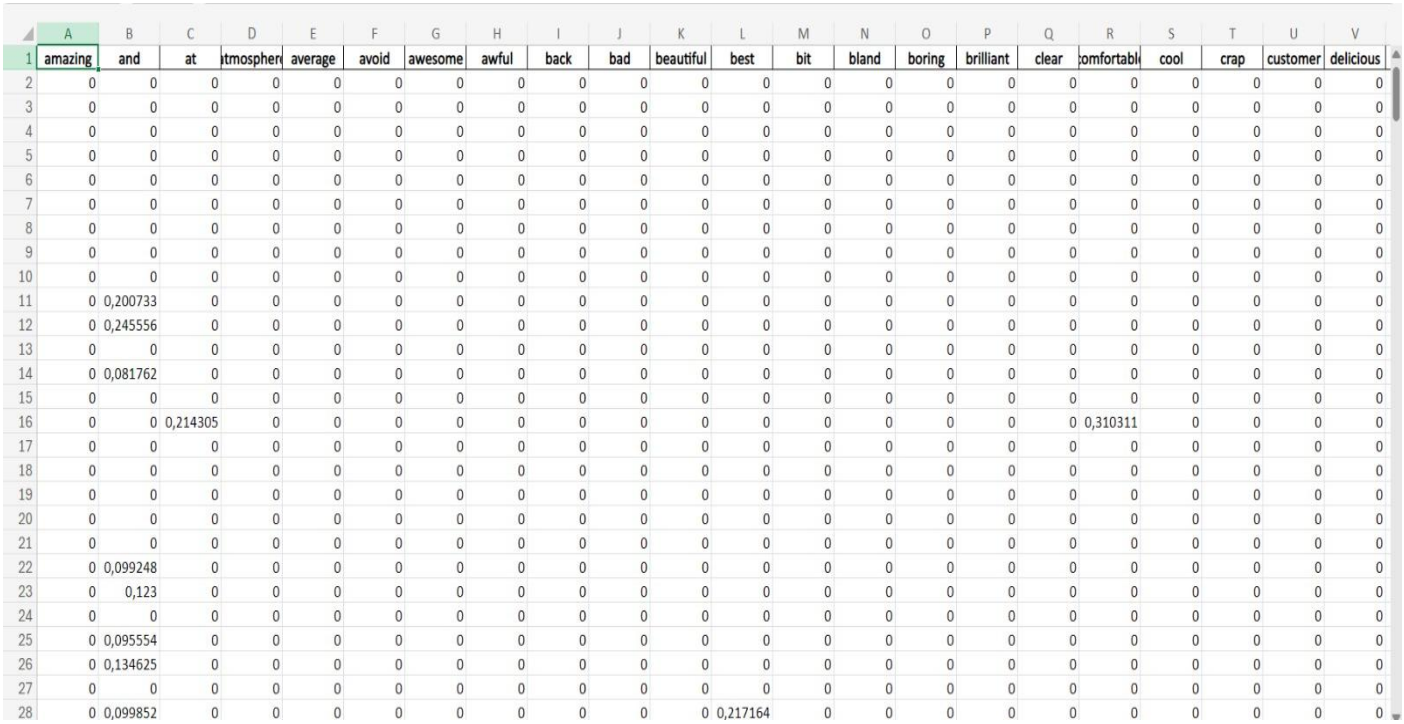
The value of ‘k’ indicates that we will select the top 100 features with the highest Chi-square scores.

Code :

chi-square.py > ...

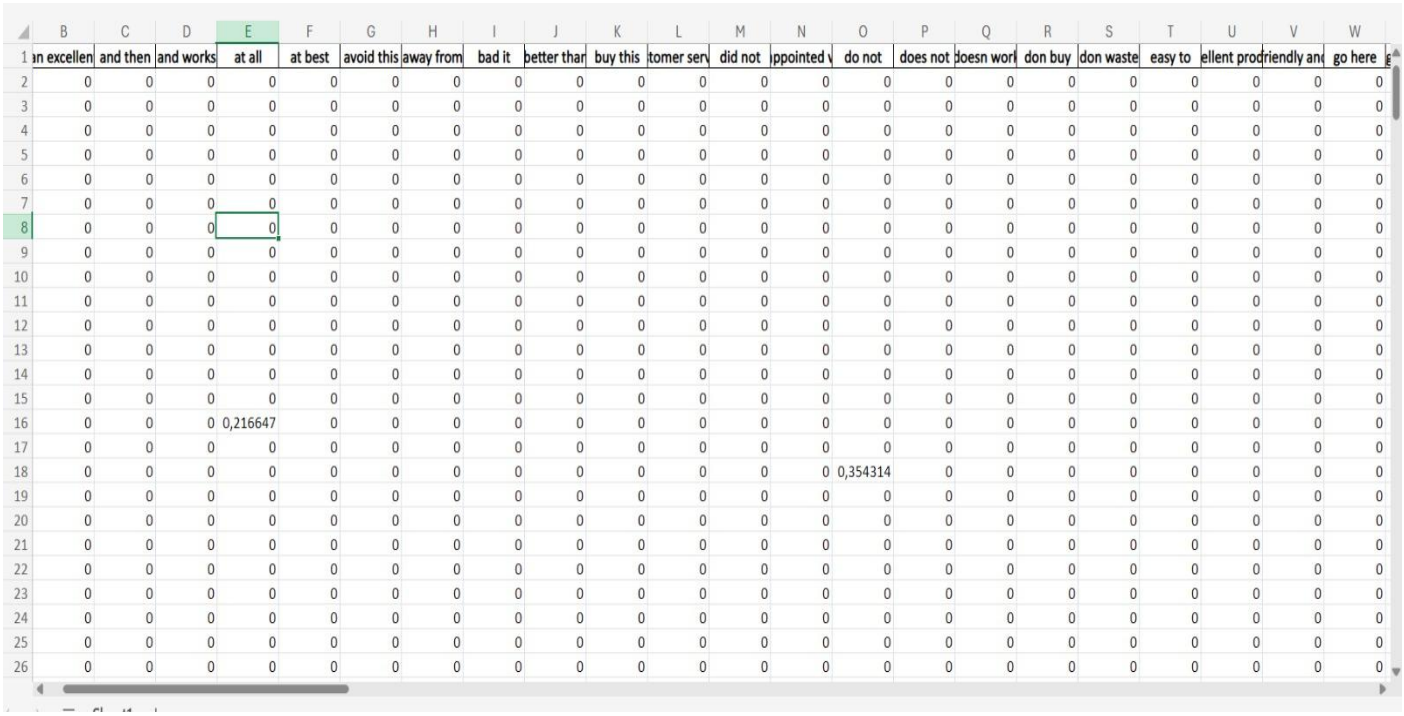
```python
import pandas as pd
from sklearn.feature_selection import SelectKBest, chi2
import os

def apply_chi_square(data, target_col, k, output_file):
    # Hedef klasörün varlığını kontrol et ve yoksa oluştur
    output_dir = os.path.dirname(output_file)
    if not os.path.exists(output_dir):
        os.makedirs(output_dir)

    X = data.drop(columns=[target_col])
    y = data[target_col]

    chi_selector = SelectKBest(score_func=chi2, k=k)
    X_new = chi_selector.fit_transform(X, y)

    selected_features = pd.DataFrame(X_new, columns=data.columns[chi_selector.get_support(indices=True)])
    selected_features[target_col] = y

    selected_features.to_csv(output_file, index=False)
    print(f"Chi-square ile seçilen özellikler kaydedildi: {output_file}")

# Veri dosyaları ve klasörler
tfidf_files = [
    ('tfidf_unigram.csv', 'unigram_results/chi_square.csv'),
    ('tfidf_bigram.csv', 'bigram_results/chi_square.csv')
]

# Her dosya için Chi-Square uygula ve kaydet
for input_file, output_file in tfidf_files:
    data = pd.read_csv(input_file)
    apply_chi_square(data, 'Label', k=100, output_file=output_file)

```

The table after applying Chi-Square to the dataset with unigram TF-IDF



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
1	amazing	and	at	tmospher	average	avoid	awesome	awful	back	bad	beautiful	best	bit	bland	boring	brilliant	clear	omfortabl	cool	crap	customer	delicious
2	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
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0,200733	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0,245556	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0,081762	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0,214305	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0,310311	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
22	0	0,099248	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
23	0	0,123	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25	0	0,095554	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
26	0	0,134625	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
28	0	0,099852	0	0	0	0	0	0	0	0	0	0,217164	0	0	0	0	0	0	0	0	0	0

The table after applying Chi-Square to the dataset with bigram TF-IDF



	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
1	n excellen	and then	and works	at all	at best	avoid this	away from	bad it	better than	buy this	tomer serv	did not	ppointed v	do not	does not	doesn worl	don buy	don waste	easy to	ellent prod	riendly an	go here
2	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
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0,216647	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0,354314	0	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

PCA reduction method :

n is indicates the number that specifies selecting the top 50 principal components with the highest eigenvalues.

Code :

pca.py

```python
import pandas as pd
from sklearn.decomposition import PCA
import os

def apply_pca(data, target_col, n_components, output_file):
    # Hedef klasörün varlığını kontrol et ve yoksa oluştur
    output_dir = os.path.dirname(output_file)
    if not os.path.exists(output_dir):
        os.makedirs(output_dir)

    X = data.drop(columns=[target_col])
    y = data[target_col]

    pca = PCA(n_components=n_components)
    X_pca = pca.fit_transform(X)

    pca_df = pd.DataFrame(X_pca, columns=[f"PC{i+1}" for i in range(n_components)])
    pca_df[target_col] = y

    pca_df.to_csv(output_file, index=False)
    print(f"PCA ile boyutu azaltılmış veri kaydedildi: {output_file}")

# Veri dosyaları ve klasörler
tfidf_files = [
    ('tfidf_unigram.csv', 'unigram_results/pca.csv'),
    ('tfidf_bigram.csv', 'bigram_results/pca.csv')
]

# Her dosya için PCA uygula ve kaydet
for input_file, output_file in tfidf_files:
    data = pd.read_csv(input_file)
    apply_pca(data, 'Label', n_components=50, output_file=output_file)
```

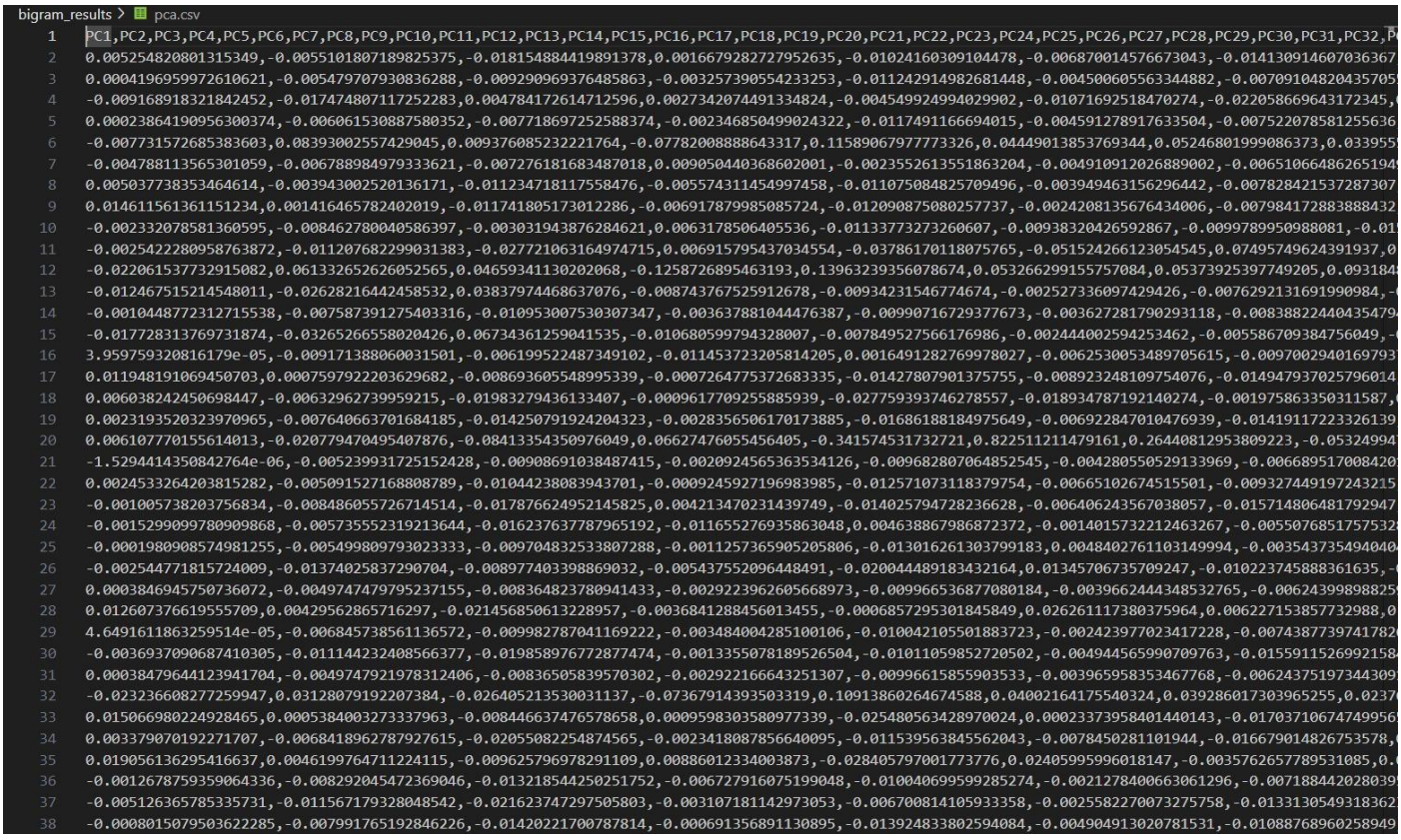
The table after applying PCA to the dataset with unigram TF-IDF

unigram_results > pca.csv

```
PC1,PC2,PC3,PC4,PC5,PC6,PC7,PC8,PC9,PC10,PC11,PC12,PC13,PC14,PC15,PC16,PC17,PC18,PC19,PC20,PC21,PC22,PC23,PC24,PC25,PC26,PC27,PC28,PC29,PC30,PC31,PC32,P
-0.0044723504809777445,-0.09983232123036741,0.04216027747927614,0.08745840539260281,-0.016549152336594136,0.0050567264353501946,0.11191798284927944,0.102
0.0031741106873912957,0.09094890485824063,-0.08276061944885453,-0.1911621493927926,0.15776693858779212,-0.05256137388043954,0.17309340736601916,-0.035777
0.151696520146985,0.2858815547042354,0.03752369338172175,0.08861411104099742,-0.1202389688279117,0.008147333403210739,0.10794007091365912,-0.010498321683
0.03234794170433043,-0.04121827329797374,-0.03839746528100324,-0.022749073670918996,-0.06529721564281213,-0.02973454506299976,0.014733830815610981,0.018
0.18548863877325936,0.3110314208131154,0.18375203833484383,0.10453918644108331,-0.008247912039692567,-0.016526441585135748,0.11463234558348516,0.0487063
-0.0060116315127232865,-0.1292042973972959,-0.030192254897481 76,0.11311736029088854,-0.0612093046573945,-0.0358595532959084,0.058317091382062806,0.07603
-0.005779752072671274,-0.060383949411083666,0.06891705331352892,-0.004648623316527874,-0.09509696221141052,-0.008879300751380978,0.002049659903836 3683,-
0.11524532685593425,-0.11713746004216187,0.032781102154929645,-0.06539718477304948,-0.04520072242134111,0.044728324191756895,0.008050576559603645,0.0786
0.028847840395052757,-0.07348722073905742,-0.0677519442646996,-0.03276519021931791,-0.08831652187274174,-0.03311925291929936,-0.020303684014672384,-0.02
-0.016530482717700735,-0.055653823718868935,-0.010094481966435833,-0.02683096788710705,-0.12224845881521054,-0.05759555337141428,-0.036278643677471044,-
0.1488125279044327,0.3672913153329124,0.2246647691599091 3,0.1044371249944765,0.035670772321432224,-0.09925553332875087,0.14278260561092948,0.12370951770
-0.13686862883467418,0.057156459038546194,0.01991510546387175,-0.03407186269836006,0.002167359453311817,-0.043458235304703975,-0.08201012001591212,-0.054
-0.04740146713073093,0.004482592925621025,0.060363364469744,-0.006943166588326713,-0.07689196325316475,-0.03254539612031017 6,0.013029176095130247,0.0047
-0.09166159469492036,0.13998705499647712,-0.12693117953266786,-0.30696115893693 99,0.3367536023214012,-0.23533488637751368,0.05376642504915509,-0.0512413
-0.09771461786562045,0.05478823388038281,0.1667418611797386,-0.10447639066108683,0.18585311046975272,-0.1851785408834284,-0.012351680051181306,-0.089689
0.12961929873265893,-0.02281441465067822,-0.011251049437046407,0.001040580248668456,-0.025845069998373993,0.012659749256022026,-0.044745291178538 2,-0.08
0.041608676874179316,-0.09392433326737816,-0.10723001153904803,-0.10231829072600293,-0.04616652346421397,0.04497125665228304,0.09123015930148552,-0.0777
-0.06064146148133899,0.023450903095849 77,-0.11603798776831 6,-0.09366500342077258,0.15724445403844 92,0.00049401449779406 87,0.19575342365704 28,0.00156424 16
0.41059834660563427,0.48059457745825407,-0.14306448129415575,0.18149594571226668,-0.10868243861207139,-0.05079155630367532,0.07854667196881708,-0.037078
0.052023213941593226,-0.1204319313016385,-0.020137987219100 07,0.056875470367444 53,0.023682736793544993,0.044245180979274275,0.002659760563207670 8,0.0462
0.004283309719807825,-0.10523452914118 2,-0.09641988205020253,0.0636826924866706,-0.020652940373 97,-0.000936795739609182 6,0.065846480982415 36,-0.045801675
0.05479599854781507,-0.09437841364838971,-0.05061285037656461 4,0.10772816047321898,0.043735728497671746,-0.022069868185990 59,-0.07928993005575 46,-0.0085
-0.09539887477253539,0.03669124030102499,0.13298080715291485,-0.01096594179309866,-0.09286243953087 95,-0.032728853958702155,0.05400475890515 8115,-0.0538
0.056844501724116 71,-0.05185710246665039 4,0.021079694301792 81,0.000943010841215930 9,-0.023613962458311974,0.005259914084832296,-0.043814752160848 43,0.01
-0.04660939386731194,-0.01593094712299063,-0.01643540026005 8745,0.1276214451216442,-0.016927102941706452,-0.0944803052824765,-0.002206929193508 0484,-0.0
0.2215022356137919,0.2094569145868363 2,-0.05696234986230923,0.04995983555250163,-0.08269590067383067,-0.018923691773045655,-0.021336334679053916,-0.0991
0.0972061923840264,0.00205302784359595 03,0.05142963178984104,0.14191112881913562,-0.018822759748823397,0.023370561766652975,-0.056108270385345264,-0.120
-0.049346312030300 05,-0.03386813632898175,0.04228204811796534,0.011203261685303481,-0.09415095929547654,-0.017636997712518 44,0.028067139227597935,0.006
-0.031473895136900 63,0.025837172657820897,0.06303090670748358,0.0714618670082038,-0.10254175421650691,-0.036444206396602 3,0.012055405290931936,-0.054777
0.027349380646141324,-0.021418619884028926,-0.04574201564351659,-0.03700274717522683,-0.046742330490172214,-0.051436365991649295,-0.012004632614317 71,-0
0.12794319055427403,-0.01595537274559091,0.2027868072231544,0.044715011377763625,0.09931696468243256,0.019170113433925277,-0.008923117523673686,0.015249
0.08593678746652272,-0.00933085125298589,0.04861974066886629,-0.10252462792513815,0.11577062447925898,-0.0746072505760411,-0.0636200653323899 1,0.025214 1
0.026012331834885088,-0.04530250243156589,0.011527299131486373,0.06593153967621249,0.05702802731062344,-0.05499099659596979,0.024502853128170 14,0.039473
0.16100123204219394,-0.03255503608675444,0.07533468654746298,-0.00054824774467416 27,0.030758648513489105,0.09895458165320739,-0.138446055249519 45,-0.094
0.011604308790538576,-0.0674398443458514,-0.00869188885114576 8,-0.019867057163625034,-0.07942255057017389,-0.055194530705261 04,0.044582708114104 94,-0.00
0.06307186651406733,-0.024492730105212548,0.11914987911835581,0.047531562138265274,-0.08480443599922094,-0.036882214666060 05,0.048657754815737427,-0.043
-0.028849503402884 2,-0.027768755687018663,-0.13153846537934635,0.05052808420366883,0.16972964244921757,-0.26787200762110375,-0.141434655229229 87,0.005209
```

The table after applying PCA to the dataset with bigram TF-IDF

bigram_results > pca.csv



```
PC1,PC2,PC3,PC4,PC5,PC6,PC7,PC8,PC9,PC10,PC11,PC12,PC13,PC14,PC15,PC16,PC17,PC18,PC19,PC20,PC21,PC22,PC23,PC24,PC25,PC26,PC27,PC28,PC29,PC30,PC31,PC32,P
0.005254820801315349,-0.0055101807189825375,-0.018154884419891378,0.0016679282727952635,-0.01024160309104478,-0.006870014576673043,-0.014130914607036367
0.0004196959972610621,-0.005479707930836288,-0.009290969376485863,-0.0032573905542333253,-0.011242914982681448,-0.004500605563344882,-0.007091048204357057
-0.009168918321842452,-0.017474807117252283,0.004784172614712596,0.0027342074491334824,-0.004549924994029902,-0.01071692518470274,-0.022058669643172345,
0.0002386419095630037 4,-0.006061530887580352,-0.007718697252588374,-0.002346850499024322,-0.0117491166694015,-0.004591278917633504,-0.007522078581255636
-0.007731572685383603,0.08393002557429045,0.009376085232221764,-0.07782008888643317,0.11589067977773326,0.04449013853769344,0.05246801999086373,0.033955
-0.004788113565301059,-0.006788984979333621,-0.007276181683487018,0.009050440368602001,-0.0023552613551863204,-0.004910912026889002,-0.006510664862651949
0.005037738353464614,-0.003943002520136171,-0.011234718117558476,-0.005574311454997458,-0.011075084825709496,-0.003949463156296442,-0.007828421537287307
0.014611561361151234,0.0014164657824020 19,-0.011741805173012286,-0.006917879985085724,-0.012090875080257737,-0.0024208135676434006,-0.007984172883888432
-0.002332078581360595,-0.008462780040586397,-0.003031943876284621,0.0063178506405536,-0.01133773273260607,-0.00938320426592867,-0.0099789950988081,-0.01
-0.0025422280958763872,-0.011207682299031383,-0.027721063164974715,0.006915795437034554,-0.03786170118075765,-0.051524266123054545,0.07495749624391937,0
-0.022061537732915082,0.06133265262605256 5,0.04659341130202068,-0.1258726895463193,0.13963239356078674,0.053266299155757084,0.05373925397749205,0.093184
-0.012467515214548011,-0.02628216442458532,0.03837974468637076,-0.008743767525912678,-0.00934231546774674,-0.002527336097429426,-0.007629213169199098 4,
-0.0010448772312715538,-0.007587391275403316,-0.010953007530307347,-0.003637881044476387,-0.00990716729377673,-0.003627281790293118,-0.008388224404354794
-0.017728313769731874,-0.032652665580204 26,0.06734361259041535,-0.010680599794328007,-0.007849527566176986,-0.002444002594253462,-0.005586709384756049,-
3.959759320816179e-05,-0.009171388060031501,-0.006199522487349102,-0.011453723205814205,0.0016491282769978027,-0.006253005348970561 5,-0.009700294016979 3
0.011948191069450703,0.0007597922203629682,-0.008693605548995339,-0.0007264775372683335,-0.01427807901375755,-0.008923248109754076,-0.014947937025796014
0.006038242450698447,-0.006329627399592 15,-0.01983279436133407,-0.0009617709255885939,-0.027759393746278557,-0.018934787192140274,-0.001975863350311587,
0.0023193520323970965,-0.007640663701684185,-0.014250791924204323,-0.0028356506170173885,-0.016861881849756 49,-0.006922847010476939,-0.01419117223326139
0.006107701551 56614013,-0.020779470495407876,-0.08413354350976049,0.06627476055456405,-0.341574531732721,0.8225112114791 61,0.2644081295380922 3,-0.05324994
-1.5294414350842764e-06,-0.005239931725152428,-0.009086910384874 15,-0.0020924565363534126,-0.009682807064852545,-0.004280550529133969,-0.006689517008420
0.0024533264203815282,-0.005091527168808789,-0.010442380839437 01,-0.0009245927196983985,-0.012571073113837975 4,-0.006651026745155 01,-0.009327449197243215
-0.001005738203756834,-0.008486055726714514,-0.017876624952145825,0.004213470231439749,-0.014025794728236628,-0.006406243567038057,-0.015714806481792947
-0.0015299099780909868,-0.005735552319213644,-0.016237637787965192,-0.011655276935863048,0.004638867986872372,-0.0014015732212463267,-0.005507685175753 25
-0.0001980908574981255,-0.005499809793023333,-0.009704832533807288,-0.0011257365905205806,-0.013016261303799183,0.004840276110314999 4,-0.003543735494040
-0.002544771815724009,-0.013740258372907 04,-0.008977403398869032,-0.005437552096448491,-0.020044489183432164,0.013457067357092 47,-0.010223745888361635,-
0.0003846945750736072,-0.004974747975237155,-0.008364823780941433,-0.0029223962605668973,-0.009966536877080184,-0.003966244434853276 5,-0.006243795197344309
0.012607376619555709,0.004295628657162 97,-0.021456850613228957,-0.0036841288456013455,-0.000685729530184 5849,0.026261117380375964,0.006227153857732988,0
4.6491611863259514e-05,-0.006845738561365 72,-0.009982787041169222,-0.003484004285100106,-0.010042105501883723,-0.002423977023417228,-0.007438773974178 2
-0.0036937090687410305,-0.011144232408566377,-0.019858976772877474,-0.0013355078189526504,-0.010111059852720502,-0.004944565990709763,-0.015591152699215 8
0.0003847964412394170 4,-0.004974792197831 2406,-0.008365058395705 02,-0.002922166643251307,-0.009966158559035 33,-0.003965958353467768,-0.006243751973443099
-0.023236608277259947,0.03128079192207384,-0.026405213530031137,-0.07367914393503319,0.10913860264674588,0.040021641755403 24,0.039286017303965255,0.0237
0.015066980224928465,0.000538400327333963,-0.008446637476578658,0.0009598303580977339,-0.025480563428970024,0.000233739584014401 43,-0.017037106747499567
0.003379070192271707,-0.006841896278792 7615,-0.020550822548745 65,-0.0023418087856640095,-0.011539563845562043,-0.007845028110 1944,-0.016679014826753578,
0.019056136295416637,0.004619976471122 4115,-0.009625796978291109,0.0088601233400387 3,-0.028405797001773776,0.024059959960181 47,-0.003576265778953108 5,0.
-0.0012678759359064336,-0.008292045472369046,-0.013218544250251752,-0.006727916075199048,-0.010040699599285274,-0.002127840066306129 6,-0.007188442028039
-0.005126365785335731,-0.011567179328048542,-0.021623747297505803,-0.003107181142973053,-0.006700814105933358,-0.002558227073275758,-0.013313054931836 2
-0.0008015079503622285,-0.007991765192846226,-0.014202217007878 14,-0.0006913568911308 95,-0.013924833802594084,-0.004904913020781531,-0.010887689602589 49
```

L1 Regularization (Lasso) : The penalty coefficient in L1 is set to its default value of C=1.0.

Code:

l1reg.py > ...

```python
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.feature_selection import SelectFromModel
import os

def apply_l1_regularization(data, target_col, output_file):
    # Hedef klasörün varlığını kontrol et ve yoksa oluştur
    output_dir = os.path.dirname(output_file)
    if not os.path.exists(output_dir):
        os.makedirs(output_dir)

    X = data.drop(columns=[target_col])
    y = data[target_col]

    # L1 Regularization uygulama
    l1_model = LogisticRegression(penalty='l1', solver='liblinear', max_iter=1000)
    l1_model.fit(X, y)

    selector = SelectFromModel(l1_model, prefit=True)
    X_new = selector.transform(X)

    selected_features = pd.DataFrame(X_new, columns=data.columns[selector.get_support(indices=True)])
    selected_features[target_col] = y

    selected_features.to_csv(output_file, index=False)
    print(f"L1 Regularization ile seçilen özellikler kaydedildi: {output_file}")

# Veri dosyaları ve klasörler
tfidf_files = [
    ('tfidf_unigram.csv', 'unigram_results/l1_regularization.csv'),
    ('tfidf_bigram.csv', 'bigram_results/l1_regularization.csv')
]

# Her dosya için L1 Regularization uygula ve kaydet
for input_file, output_file in tfidf_files:
    data = pd.read_csv(input_file)
    apply_l1_regularization(data, 'Label', output_file=output_file)
```

The table after applying L1 Regularization (Lasso) to the dataset with unigram TF-IDF

unigram_results > l1_regularization.csv

```
,amazing,and,at,average,avoid,away,awesome,awful,back,bad,battery,be,beautiful,best,bit,bland,boring,both,brilliant,character,cheap,clear,
,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.
,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.
,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.
,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.
,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.
,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.
,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.
,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.
,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.
27222344956,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0
57401722761,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0
,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.5229854749584628,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0
20672290234,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0
,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.
143046718926814,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.3103109322773043,0.0,0.0,0.0,0.0,0.0,0.0,0.0,
,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.
,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.2731402433064856,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0
,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.
,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.
,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.
.0,0.0,0.0992476597025785,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,
95958464503,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0
,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.
12927403377,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.2278738935639326,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,
15975204292,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.3210471275196541,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,
,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.
19261961561,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.2171644585735455,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,
,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.3273815323498285,0.0,0.0,0.0,0.0,0.0,0.0
33401583301,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0
,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.5127650316907543,0.0
,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.
,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.
323252432943764,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0
29560098831,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0
,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.
520494686055573,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.2016508114326566,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,
,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.
```

The table after applying L1 Regularization (Lasso) to the dataset with bigram TF-IDF

bigram_results > l1_regularization.csv

```
10 10,all the,an excellent,and is,and the,at all,at best,avoid this,away from,better than,customer service,did not,disappointed with,do no
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.44341151
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.
0.0,0.0,0.0,0.0,0.3299896612373952,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.
0.0,0.0,0.0,0.0,0.0,0.2166474963321319,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.2743047647868845,0.0,0.0
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.354314154221361,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.2072400646277779,0.0,0.0,0.0,0.0
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.
```

VARIANCE THRESHOLD : **The threshold value in this reduction method has been set to k=0.001.**

Code :

threshold.py

C:\Users\Semih Zenginoğlu\Desktop\en son data mining\threshold.py (preview)

```python
import pandas as pd
from sklearn.feature_selection import VarianceThreshold
import os

def apply_variance_threshold(data, target_col, threshold, output_file):
    # Hedef klasörün varlığını kontrol et ve yoksa oluştur
    output_dir = os.path.dirname(output_file)
    if not os.path.exists(output_dir):
        os.makedirs(output_dir)

    X = data.drop(columns=[target_col])
    y = data[target_col]

    # Variance Threshold uygulama
    selector = VarianceThreshold(threshold=threshold)
    X_new = selector.fit_transform(X)

    selected_features = pd.DataFrame(X_new, columns=data.columns[selector.get_support(indices=True)])
    selected_features[target_col] = y

    selected_features.to_csv(output_file, index=False)
    print(f"Variance Threshold ile seçilen özellikler kaydedildi: {output_file}")

# Veri dosyaları ve klasörler
tfidf_files = [
    ('tfidf_unigram.csv', 'unigram_results/variance_threshold.csv'),
    ('tfidf_bigram.csv', 'bigram_results/variance_threshold.csv')
]

# Her dosya için Variance Threshold uygula ve kaydet
for input_file, output_file in tfidf_files:
    data = pd.read_csv(input_file)
    apply_variance_threshold(data, 'Label', threshold=0.001, output_file=output_file)
```

The table after applying threshold to the dataset with unigram TF-IDF

```
10,about,absolutely,after,again,all,also,am,amazing,an,and,any,are,as,at,avoid,awesome,awful,back,bad,battery,be,because,been,best,better,
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.2255508817553645,0.0
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.511961350250
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.1661588560877016,0.0
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.2453304908790027,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.2007327222344956,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.2455557401722761,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.5229854749584628,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0817620672290234,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.17890
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.
0.0,0.0,0.0,0.0,0.0,0.2118485094548254,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.2106623032415725,0.2143046718926814,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.2975675268866368,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.2731402433064856,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.
0.0,0.0,0.0,0.0,0.2356817255687914,0.1880208693843011,0.0,0.0,0.0,0.0,0.0992476597025785,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0
0.0,0.0,0.3358274078385818,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.1229995958464503,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.2780578995082287,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0955542927403377,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.2278738935639326,0.0,0.0,0.0,0.0,0.0,0.0,
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.1346245975204292,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.3210471275196541,0.0,0.0,0.0,0.0,0.0,0.0,
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0998519261961561,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.2171644585735455,0.0,0.0,0.0,0.0,0.0,0.0,
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.1854783401583301,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.2125031847208081,0.0,0.0,0.0
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.1941382476376097,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.2323252432943764,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.1230129560098831,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.1395077915240551,0.2904576338952151,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.1620494686055573,0.0,0.0,0.0,0.2016508114326566,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.
```

The table after applying threshold to the dataset with bigram TF-IDF

```
bigram_results > variance_threshold.csv
be back,food was,great phone,great product,is great,it is,it was,love this,not good,of the,service was,the food,this is,this place,very di
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
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
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0
0.0,0.0,0.0,0.0,0.4434115189473218,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0
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
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0
0.0,0.0,0.0,0.0,0.4278680513130835,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.4678389928418552,0.0,1
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0
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
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0
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
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,1
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0
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
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0
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
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0
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
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.1829154025349826,0.0,0.0,0.0,0.0,0.0,0
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1
0.0,0.0,0.0,0.0,0.0,0.2065333495743842,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0
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
```

STEP 3 – CLASSIFICATION :

In the third step, we classified the datasets reduced using four different methods by applying classification models and compared the results using four different metrics. The classification models we used are Naive Bayes, SVM, and Decision Tree.

1 - NAIVE BAYES MODEL :

We are sharing the classification of the operations we performed using the Naive Bayes algorithm and the corresponding code. First, in the Naive Bayes model training, we used 20% of the data as test data and 80% as training data. The Python code for training this model is as follows:

Code:

naive.py > evaluate_model

```python
import os
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Performans değerlendirme fonksiyonu
def evaluate_model(data_file, method_name, results):
    if not os.path.exists(data_file):
        print(f"Dosya bulunamadı: {data_file}")
        return

    data = pd.read_csv(data_file)
    X = data.drop(columns=['Label'])
    y = data['Label']

    # Negatif değerleri sıfıra çevir
    X[X < 0] = 0

    # Eğitim ve test setlerine ayırma
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    # Naive Bayes model eğitimi
    model = MultinomialNB()
    model.fit(X_train, y_train)

    # Test setinde tahmin
    y_pred = model.predict(X_test)

    # Performans metriklerini hesapla
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred, average='weighted')
    recall = recall_score(y_test, y_pred, average='weighted')
    f1 = f1_score(y_test, y_pred, average='weighted')

    # Sonuçları kaydet
    results.append({
        'Method': method_name,
        'Accuracy': accuracy,
        'Precision': precision,
        'Recall': recall,
        'F1 Score': f1
    })

# Sonuçları karşılaştırmak için fonksiyon
def compare_methods(result_folders, output_file):
    results = []
    for folder in result_folders:
        for method_file in ['chi_square.csv', 'pca.csv', 'variance_threshold.csv', 'l1_regularization.csv','tfidf.csv' ]:
            method_path = os.path.join(folder, method_file)
            method_name = f"{folder.split('_')[0]}_{method_file.split('.')[0]}"
            evaluate_model(method_path, method_name, results)

    # Sonuçları bir DataFrame'e dönüştür
    results_df = pd.DataFrame(results)

    # Excel dosyasına kaydet
    results_df.to_excel(output_file, index=False)
    print(f"Sonuçlar kaydedildi: {output_file}")

# Unigram ve Bigram klasörleri
result_folders = ['unigram_results', 'bigram_results']

# Çıktı dosyası
output_excel = 'results_comparison.xlsx'

# Yöntemleri karşılaştır ve Excel'e yaz
compare_methods(result_folders, output_excel)
```

RESULT OF NAIVE BAYES MODEL :

Method	Accuracy	Precision	Recall	F1 Score
unigram_chi_square	0,785454545	0,80555711	0,785454545	0,783856903
unigram_pca	0,598181818	0,632475936	0,598181818	0,583492448
unigram_variance_threshold	0,763636364	0,769391207	0,763636364	0,76358323
unigram_l1_regularization	0,836363636	0,842709562	0,836363636	0,836326852
unigram_tfidf (without reduction)	0,821818182	0,822883567	0,821818182	0,821959736
bigram_chi_square	0,583636364	0,730715584	0,583636364	0,522739868
bigram_pca	0,523636364	0,613042298	0,523636364	0,446510344
bigram_variance_threshold	0,48	0,513195803	0,48	0,379865759
bigram_l1_regularization	0,578181818	0,662935357	0,578181818	0,534841598
bigram_tfidf (without reduction)	0,709090909	0,729702556	0,709090909	0,706084467

2 - SVM MODEL : In this model, we used 20% of the data for testing and 80% for training. A linear kernel SVM model was created and trained using the dataset. We compared the applied TF-IDF and reduction methods based on four metrics and presented the results in an Excel table. The Python code for the SVM model is as follows:

Code :

svm.py > evaluate_model

```python
import os
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Performans değerlendirme fonksiyonu
def evaluate_model(data_file, method_name, results):
    if not os.path.exists(data_file):
        print(f"Dosya bulunamadı: {data_file}")
        return

    data = pd.read_csv(data_file)
    X = data.drop(columns=['Label'])
    y = data['Label']

    # Negatif değerleri sıfıra çevir
    X[X < 0] = 0

    # Eğitim ve test setlerine ayırma
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    # SVM model eğitimi
    model = SVC(kernel='linear')  # SVM with linear kernel
    model.fit(X_train, y_train)

    # Test setinde tahmin
    y_pred = model.predict(X_test)

    # Performans metriklerini hesapla
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred, average='weighted')
    recall = recall_score(y_test, y_pred, average='weighted')
    f1 = f1_score(y_test, y_pred, average='weighted')
```

```python
        # Sonuçları kaydet
        results.append({
            'Method': method_name,
            'Accuracy': accuracy,
            'Precision': precision,
            'Recall': recall,
            'F1 Score': f1
        })

    # Sonuçları karşılaştırmak için fonksiyon
    def compare_methods_svm(result_folders, output_file):
        results = []
        for folder in result_folders:
            for method_file in ['chi_square.csv', 'pca.csv', 'variance_threshold.csv', 'l1_regularization.csv', 'tfidf.csv']:
                method_path = os.path.join(folder, method_file)
                method_name = f"{folder.split('_')[0]}_{method_file.split('.')[0]}"
                evaluate_model(method_path, method_name, results)

        # Sonuçları bir DataFrame'e dönüştür
        results_df = pd.DataFrame(results)

        # Excel dosyasına kaydet
        results_df.to_excel(output_file, index=False)
        print(f"SVM Sonuçları kaydedildi: {output_file}")

    # Unigram ve Bigram klasörleri
    result_folders = ['unigram_results', 'bigram_results']

    # Çıktı dosyası
    output_excel_svm = 'svm_results_comparison.xlsx'

    # Yöntemleri karşılaştır ve Excel'e yaz
    compare_methods_svm(result_folders, output_excel_svm)
```

RESULT OF SVM MODEL :

	Method	Accuracy	Precision	Recall	F1 Score
2	unigram_chi_square	0,783636364	0,78673522	0,783636364	0,781933833
3	unigram_pca	0,681818182	0,690640005	0,681818182	0,673450865
4	unigram_variance_threshold	0,765454545	0,765627885	0,765454545	0,764803506
5	unigram_l1_regularization	0,834545455	0,83447994	0,834545455	0,834487791
6	unigram_tfidf(without reduction)	0,827272727	0,827741088	0,827272727	0,827372859
7	bigram_chi_square	0,585454545	0,749975702	0,585454545	0,521276313
8	bigram_pca	0,516363636	0,666355015	0,516363636	0,410842975
9	bigram_variance_threshold	0,483636364	0,65742621	0,483636364	0,334852604
10	bigram_l1_regularization	0,58	0,676552285	0,58	0,532813988
11	bigram_tfidf(without reduction)	0,709090909	0,716029406	0,709090909	0,708783105

3 – DECISION TREE :

In this study, 80% of the dataset was used for training and 20% for testing. A classic decision tree model, DecisionTreeClassifier, was used. The model utilized the default Gini Impurity criterion for making branching decisions. The Python code for the Decision Tree model is as follows:

Code :

decissiontree.py > ...

```python
import os
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Performans değerlendirme fonksiyonu
def evaluate_model(data_file, method_name, results):
    if not os.path.exists(data_file):
        print(f"Dosya bulunamadı: {data_file}")
        return

    data = pd.read_csv(data_file)
    X = data.drop(columns=['Label'])
    y = data['Label']

    # Negatif değerleri sıfıra çevir
    X[X < 0] = 0

    # Eğitim ve test setlerine ayırma
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    # Decision Tree model eğitimi
    model = DecisionTreeClassifier(random_state=42)  # Karar ağacı modeli
    model.fit(X_train, y_train)

    # Test setinde tahmin
    y_pred = model.predict(X_test)

    # Performans metriklerini hesapla
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred, average='weighted')
    recall = recall_score(y_test, y_pred, average='weighted')
    f1 = f1_score(y_test, y_pred, average='weighted')
```

```python
    # Sonuçları kaydet
    results.append({
        'Method': method_name,
        'Accuracy': accuracy,
        'Precision': precision,
        'Recall': recall,
        'F1 Score': f1
    })

# Sonuçları karşılaştırmak için fonksiyon
def compare_methods_decision_tree(result_folders, output_file):
    results = []
    for folder in result_folders:
        for method_file in ['chi_square.csv', 'pca.csv', 'variance_threshold.csv', 'l1_regularization.csv', 'tfidf.csv' ]:
            method_path = os.path.join(folder, method_file)
            method_name = f"{folder.split('_')[0]}_{method_file.split('.')[0]}"
            evaluate_model(method_path, method_name, results)

    # Sonuçları bir DataFrame'e dönüştür
    results_df = pd.DataFrame(results)

    # Excel dosyasına kaydet
    results_df.to_excel(output_file, index=False)
    print(f"Decision Tree Sonuçları kaydedildi: {output_file}")

# Unigram ve Bigram klasörleri
result_folders = ['unigram_results', 'bigram_results']

# Çıktı dosyası
output_excel_tree = 'decision_tree_results_comparison.xlsx'

# Yöntemleri karşılaştır ve Excel'e yaz
compare_methods_decision_tree(result_folders, output_excel_tree)
```

RESULT OF DECISION TREE MODEL:

	A	B	C	D	E
1	**Method**	**Accuracy**	**Precision**	**Recall**	**F1 Score**
2	unigram_chi_square	0,718182	0,722424242	0,718181818	0,7144022
3	unigram_pca	0,610909	0,610340057	0,610909091	0,610495302
4	unigram_variance_threshold	0,638182	0,638260429	0,638181818	0,638219015
5	unigram_l1_regularization	0,716364	0,716649994	0,716363636	0,716468925
6	unigram_tfidf(without reduction)	0,692727	0,695572476	0,692727273	0,692937595
7	bigram_chi_square	0,576364	0,725913212	0,576363636	0,511514404
8	bigram_pca	0,592727	0,593712024	0,592727273	0,593023946
9	bigram_variance_threshold	0,525455	0,508011698	0,525454545	0,43591626
10	bigram_l1_regularization	0,567273	0,644643806	0,567272727	0,522811639
11	bigram_tfidf(without reduction)	0,578182	0,600193146	0,578181818	0,568061584

RESULT AND COMPARISONS :

In this section, we will compare the methods, models, and types of TF*IDF used. Based on this study, I am sharing the most successful methods, models, and types of TF*IDF.

1) Comparison of TF-IDF Methods

Unigram TF-IDF provided the most effective representation in all models and delivered high accuracy rates.

2) Comparison of Dimensionality Reduction Methods

L1 Regularization has demonstrated the highest performance across all models, making it the most successful dimensionality reduction method. It minimized data loss, enhancing the model's overall performance.

Chi-Square produced effective results, particularly with the SVM model, but it was not as strong as L1 Regularization.

PCA showed the lowest performance, making it an ineffective dimensionality reduction method.

Non-reduced TF-IDF achieved results close to the dimensionality reduction techniques in some cases. Not applying dimensionality reduction can be effective, especially for smaller datasets.

3) Model Comparison

Naive Bayes exhibited the best performance in terms of Accuracy and F1 Score, ranking first overall.

SVM produced results close to Naive Bayes, ranking second. It is competitive in terms of F1 Score.

Decision Tree showed lower performance compared to the other models and was less effective for high-dimensional datasets.

4. Best Combinations

Naive Bayes + L1 Regularization + Unigram TF-IDF

Accuracy: 83.64%, F1 Score: 83.63%

SVM + L1 Regularization + Unigram TF-IDF

Accuracy: 83.45%, F1 Score: 83.44%

SVM + Non-reduced TF-IDF + Unigram

Accuracy: 82.73%, F1 Score: 82.73%

Conclusion

L1 Regularization is the most effective dimensionality reduction method.

Naive Bayes ranks first in overall performance.

Unigram TF-IDF, when combined with dimensionality reduction methods, produces the strongest results.

Non-reduced TF-IDF is an effective alternative for smaller datasets.

AS A RESULT :

We worked on making the given datasets more understandable and simplified by performing specific operations. As a result of these operations, we used certain algorithms to classify the outcomes, trained them, and compared their performance. We aimed to demonstrate the operations, methods, and algorithms we used step by step and shared their results in this report. Everything we did from the start to the end of this project has been explained in this report.

REFERENCES:

-) https://tr.wikipedia.org/wiki/Anasayfa

-) https://archive.ics.uci.edu/dataset/331/sentiment+labelled+sentences

-) https://www.geeksforgeeks.org/

-) https://www.geeksforgeeks.org/understanding-tf-idf-term-frequency-inverse-document-frequency/

-) https://www.ibm.com/think/topics/classification-models

-) ChatGPT