HomeWork number One

Machine Learning

Professor : Dr Rezvanian

Student : Mahdi Mahdiani 4021334041

Date : ABAN- Azar 1402

Libraries

In [1]:
```
import numpy as np
import pandas as pd
from scipy.stats import zscore
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
```

Import Dataset

In [2]:
```
df = pd.read_excel("HW1_USC_ML_4021.xlsx")
```

EDA

In [3]:
```
df.head()
```

Out[3]:

	زبان	جملات
0	عربی	هل يتسبب شداد في تعطيل مشاركة المريخ عربيا
1	عربی	اتحاد الطائرة يرفض الاستقالة
2	عربی	ادوات تجميل للرجال تشجعهم للتشبه بالنساء
3	عربی	الاتحاد الرياضي يرسل برنامج الدوري العام
4	عربی	الارباب يغادر الاسماعيلية الى جده في مهمة خاصة

In [4]:
```
df.tail()
```

Out[4]:

	زبان	جملات
95	فارسی	...استیلای عرب‌ها به آسیای میانه دگرگونی‌های زیاد
96	فارسی	...هم‌اکنون در کشور تاجیکستان رسماً از خط سیریلیک
97	فارسی	قبل از انقلاب روسیه خط مردم تاجیکستان پارسی بود
98	فارسی	...حرف آ در اسپانیایی در تمام کلمات بدون تغییری ب
99	فارسی	نام کتاب او اسطورشیا به معنای اصول هندسه‌است

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   100   جملات non-null    object
 1   100    زبان non-null    object
dtypes: object(2)
memory usage: 1.7+ KB
```

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

Out[6]:
```
زبان
50      عربی
50     فارسی
dtype: int64
```

Define Persian and Arabic Characters

In [7]:
```
persian_alphabet = ['ی', 'و', 'ه', 'ن', 'م', 'ل', 'گ', 'ک', 'ق', 'ف', 'غ', 'ع', 'ظ', 'ط', 'ض', 'ص', 'ش', 'س', 'ژ'
arabic_alphabet = [ 'و', 'ه', 'ن', 'م', 'ل', 'ك', 'ق', 'ف', 'غ', 'ع', 'ظ', 'ط', 'ض', 'ص', 'ش', 'س', 'ز', 'ر', 'ذ'
combined_alphabet = persian_alphabet + arabic_alphabet
unique_char= list(set(combined_alphabet))
print(unique_char)
```

```
['', 'ض', 'خ', 'ر', 'ب', 'چ', 'ث', 'ج', 'و', 'ک', 'ع', 'آ', 'م', 'ت', 'ك', 'إ', 'ط', 'ق', 'ص', 'ي', 'غ', 'گ', 'ا'
ظ', 'پ', 'ن', 'ش', 'ل', 'ح', 'ز', 'ف', 'ی', 'ه', 'ة', 'ژ', 'د', 'س', 'ذ']
```

binary Bag Of character

Define a Function to set 1 and 0

In [8]:
```
def bag_of_characters_binary(sentence):
    char_binary = {char: 1 if char in sentence else 0 for char in unique_char}
    return char_binary
columns = {char: [] for char in unique_char}
```

use the Function

In [9]:
```
for index, row in df.iterrows():
    sentence = row['جملات']
    sentence = sentence.replace("ى" ,"ي")
    sentence = sentence.replace("ت","ة")
    sentence = sentence.replace("ا","إ")
    sentence = sentence.replace("ا","آ")
    char_binary = bag_of_characters_binary(sentence)

    for char in unique_char:
        columns[char].append(char_binary[char])
```

Creating new data frame for Binary BoW

In [10]:
```
result_df = pd.DataFrame(columns)
result_df['جملات'] = df['جملات']
result_df["زبـان"]=df["زبـان"]
```

Some checks on binary BoW data fram and save it to excel file

In [11]:
```
result_df
```

Out[11]:

	زبان	جملات	ظ	پ	ن	ش	ل	ح	ز	ف	...	ت	ك	إ	ط	ق	ص	ي	غ	گ	ا
0	عربی	هل يتسبب شداد في تعطيل مشاركة المريخ عربيا	0	0	0	1	1	0	0	1	...	1	1	0	1	0	0	0	0	0	1
1	عربی	اتحاد الطائرة يرفض الاستقالة	0	0	0	0	1	1	0	1	...	1	0	0	1	1	0	0	0	0	1
2	عربی	ادوات تجميل للرجال تشجعهم للتشبه بالنساء	0	0	1	1	1	0	0	0	...	1	0	0	0	0	0	0	0	0	1
3	عربی	الاتحاد الرياضي يرسل برنامج الدوري العام	0	0	1	0	1	1	0	0	...	1	0	0	0	0	0	0	0	0	1
4	عربی	الارباب يغادر الاسماعيلية الى جده في مهمة خاصة	0	0	0	0	1	0	0	1	...	1	0	0	0	0	1	0	1	0	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
95	فارسی	...استیلای عرب‌ها به آسیای میانه دگرگونی‌های زیاد	0	0	1	0	1	0	1	0	...	1	0	0	0	0	0	0	0	0	1
96	فارسی	...هم‌اکنون در کشور تاجیکستان رسماً از خط سیریلیک	0	0	1	1	1	0	1	1	...	1	0	0	1	0	0	0	0	0	1
97	فارسی	قبل از انقلاب روسیه خط مردم تاجیکستان پارسی بود	0	1	1	0	1	0	1	0	...	1	0	0	1	1	0	0	0	0	1
98	فارسی	...حرف آ در اسپانیایی در تمام کلمات بدون تغییری ب	0	1	1	1	1	0	1	0	...	1	0	0	0	0	1	0	1	0	1
99	فارسی	نام کتاب او اسطورشیا به معنای اصول هندسه‌است	0	0	1	1	1	0	0	0	...	1	0	0	1	0	0	0	0	0	1

100 rows × 39 columns

In [12]:
```
result_df.head()
```

Out[12]:

	زبان	جملات	ظ	پ	ن	ش	ل	ح	ز	ف	...	ت	ك	إ	ط	ق	ص	ي	غ	گ	ا
0	عربی	هل يتسبب شداد في تعطيل مشاركة المريخ عربيا	0	0	0	1	1	0	0	1	...	1	1	0	1	0	0	0	0	0	1
1	عربی	اتحاد الطائرة يرفض الاستقالة	0	0	0	0	1	1	0	1	...	1	0	0	1	1	0	0	0	0	1
2	عربی	ادوات تجميل للرجال تشجعهم للتشبه بالنساء	0	0	1	1	1	0	0	0	...	1	0	0	0	0	0	0	0	0	1
3	عربی	الاتحاد الرياضي يرسل برنامج الدوري العام	0	0	1	0	1	1	0	0	...	1	0	0	0	0	0	0	0	0	1
4	عربی	الارباب يغادر الاسماعيلية الى جده في مهمة خاصة	0	0	0	0	1	0	0	1	...	1	0	0	0	0	1	0	1	0	1

5 rows × 39 columns

Changing Arabic to 0 and Persian to 1

In [13]:
```
result_df.replace({ 'عربی'} :'زبان':"Arabic" , 'فارسی':"Persian"} } ,inplace=True)
result_df.to_excel("binaryBoW.xlsx",index=False)
```

Weighten Bag of Character

Define Function to calculate count of each char

In [14]:
```
def Weighten_bag_of_characters(sentence):
    char_freq = {}
    for char in sentence:
        char_freq[char] = char_freq.get(char, 0) + 1
    return char_freq
```

Use the function

In [15]:
```
columns = {char: [] for char in unique_char}

for index, row in df.iterrows():
    sentence = row['جملات']
    sentence = sentence.replace("ى" ,"ي")
    sentence = sentence.replace("ت","ة")
    sentence = sentence.replace("ا","إ")
    sentence = sentence.replace("ا","آ")
    char_freq = Weighten_bag_of_characters(sentence)

    for char in unique_char:
        columns[char].append(char_freq.get(char, 0))
```

Creating new data frame for Weighten BoW

In [16]:
```
result_df2 = pd.DataFrame(columns)
result_df2['جملات'] = df['جملات']
result_df2["زبـان"]=df["زبـان"]
```

Some checks on binary BoW data fram and save it to excel file

In [17]:
```
result_df2
```

Out[17]:

	زبان	جملات	ظ	پ	ن	ش	ل	ح	ز	ف	...	ت	ك	إ	ط	ق	ص	ي	غ	گ	ا
0	عربی	هل يتسبب شداد في تعطيل مشاركة المريخ عربيا	0	0	0	2	3	0	0	1	...	3	1	0	1	0	0	0	0	0	4
1	عربی	اتحاد الطائرة يرفض الاستقالة	0	0	0	0	3	1	0	1	...	4	0	0	1	1	0	0	0	0	7
2	عربی	ادوات تجميل للرجال تشجعهم للتشبه بالنساء	0	0	1	2	7	0	0	0	...	4	0	0	0	0	0	0	0	0	5
3	عربی	الاتحاد الرياضي يرسل برنامج الدوري العام	0	0	1	0	5	1	0	0	...	1	0	0	0	0	0	0	0	0	9
4	عربی	الارباب يغادر الاسماعيلية الى جده في مهمة خاصة	0	0	0	0	4	0	0	1	...	3	0	0	0	0	1	0	1	0	9
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
95	فارسی	...استیلای عرب‌ها به آسیای میانه دگرگونی‌های زیاد	0	0	2	0	1	0	1	0	...	1	0	0	0	0	0	0	0	0	9
96	فارسی	...هم‌اکنون در کشور تاجیکستان رسماً از خط سیریلیک	0	0	3	2	1	0	1	1	...	3	0	0	1	0	0	0	0	0	7
97	فارسی	قبل از انقلاب روسیه خط مردم تاجیکستان پارسی بود	0	1	2	0	2	0	1	0	...	2	0	0	1	2	0	0	0	0	6
98	فارسی	...حرف آ در اسپانیایی در تمام کلمات بدون تغییری ب	0	1	4	1	1	1	0	1	...	4	0	0	0	0	1	0	1	0	8
99	فارسی	نام کتاب او اسطورشیا به معنای اصول هندسه‌است	0	0	3	1	1	0	0	0	...	2	0	0	1	0	1	0	0	0	8

100 rows × 39 columns

In [18]:
```
result_df2.head()
```

Out[18]:

	ا	گ	غ	ي	ص	ق	ط	إ	ك	ت	...	ف	ز	ح	ل	ش	ن	پ	ظ	جملات	زبان
0	4	0	0	0	0	0	1	0	1	3	...	1	0	0	3	2	0	0	0	هل يتسبب شداد في تعطيل مشاركة المريخ عربيا	عربى
1	7	0	0	0	0	1	1	0	0	4	...	1	0	1	3	0	0	0	0	اتحاد الطائرة يرفض الاستقالة	عربى
2	5	0	0	0	0	0	0	0	0	4	...	0	0	0	7	2	1	0	0	ادوات تجميل للرجال تشجعهم للتشبه بالنساء	عربى
3	9	0	0	0	0	0	0	0	0	1	...	0	0	1	5	0	1	0	0	الاتحاد الرياضي يرسل برنامج الدوري العام	عربى
4	9	0	1	0	1	0	0	0	0	3	...	1	0	0	4	0	0	0	0	الارباب يغادر الاسماعيلية الى جده في مهمة خاصة	عربى

5 rows × 39 columns

Changing Arabic to 0 and Persian to 1

In [19]:
```
result_df2.replace({ 'عربى'} :'زبان':"Arabic" , 'فارسى':"Persian"} } ,inplace=True)
result_df2.to_excel("WeightenBoW.xlsx",index=False)
```

Length Normalized BoW

defining Function

In [20]:
```
def bag_of_characters_normalized(sentence):
    char_freq = {char: sentence.count(char) / len(sentence) for char in unique_char}
    return char_freq
```

using function

In [21]:
```
columns = {char: [] for char in unique_char}
for index, row in df.iterrows():
    sentence = row['جملات']
    sentence = sentence.replace("ى" ,"ي")
    sentence = sentence.replace("ت","ة")
    sentence = sentence.replace("ا","إ")
    sentence = sentence.replace("ا","آ")
    sentence = sentence.replace(" ","")
    char_freq_normalized = bag_of_characters_normalized(sentence)
    for char in unique_char:
        columns[char].append(char_freq_normalized[char])
```

Creating new data frame for Length Normalized BoW

In [22]:
```
result_df3 = pd.DataFrame(columns)
result_df3['جملات'] = df['جملات']
result_df3["زبـان"]=df["زبـان"]
```

Some checks on Length Normalized BoW data fram and save it to excel file

In [23]:
```
result_df3
```

Out[23]:

	ا	گ	غ	ي	ص	ق	ط	إ	ك	ت	...	ف	ز	ح	
0	0.114286	0.000000	0.000000	0.0	0.000000	0.000000	0.028571	0.0	0.028571	0.085714	...	0.028571	0.000000	0.000000	0.0
1	0.280000	0.000000	0.000000	0.0	0.000000	0.040000	0.040000	0.0	0.000000	0.160000	...	0.040000	0.000000	0.040000	0.12
2	0.142857	0.000000	0.000000	0.0	0.000000	0.000000	0.000000	0.0	0.000000	0.114286	...	0.000000	0.000000	0.000000	0.20
3	0.257143	0.000000	0.000000	0.0	0.000000	0.000000	0.000000	0.0	0.000000	0.028571	...	0.000000	0.000000	0.028571	0.14

4	0.230769	0.000000	0.025641	0.0	0.025641	0.000000	0.000000	0.0	0.000000	0.076923	...	0.025641	0.000000	0.000000	0.1(
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
95	0.200000	0.044444	0.000000	0.0	0.000000	0.000000	0.000000	0.0	0.000000	0.022222	...	0.000000	0.022222	0.000000	0.0
96	0.127273	0.000000	0.000000	0.0	0.000000	0.000000	0.018182	0.0	0.000000	0.054545	...	0.018182	0.018182	0.000000	0.0
97	0.153846	0.000000	0.000000	0.0	0.000000	0.051282	0.025641	0.0	0.000000	0.051282	...	0.000000	0.025641	0.000000	0.0
98	0.137931	0.000000	0.017241	0.0	0.017241	0.000000	0.000000	0.0	0.000000	0.068966	...	0.017241	0.000000	0.017241	0.0
99	0.205128	0.000000	0.000000	0.0	0.025641	0.000000	0.025641	0.0	0.000000	0.051282	...	0.000000	0.000000	0.000000	0.0

100 rows × 39 columns

In [24]:
```
result_df3.head()
```

Out[24]:

	ا	گ	غ	ي	ص	ق	ط	إ	ك	ت	...	ف	ز	ح	ل	ش	
0	0.114286	0.0	0.000000	0.0	0.000000	0.00	0.028571	0.0	0.028571	0.085714	...	0.028571	0.0	0.000000	0.085714	0.057143	0
1	0.280000	0.0	0.000000	0.0	0.000000	0.04	0.040000	0.0	0.000000	0.160000	...	0.040000	0.0	0.040000	0.120000	0.000000	0
2	0.142857	0.0	0.000000	0.0	0.000000	0.00	0.000000	0.0	0.000000	0.114286	...	0.000000	0.0	0.000000	0.200000	0.057143	
3	0.257143	0.0	0.000000	0.0	0.000000	0.00	0.000000	0.0	0.000000	0.028571	...	0.000000	0.0	0.028571	0.142857	0.000000	
4	0.230769	0.0	0.025641	0.0	0.025641	0.00	0.000000	0.0	0.000000	0.076923	...	0.025641	0.0	0.000000	0.102564	0.000000	0

5 rows × 39 columns

Changing Arabic to 0 and Persian to 1

In [25]:
```
result_df3.replace({ 'عربی'} :'زبان':"Arabic" , 'فارسی':"Persian"} } ,inplace=True)
result_df3.to_excel("Length_Normalized_BoW.xlsx",index=False)
```

Z Score Normalized BoW

defining Function

In [26]:
```
def bag_of_characters_z_score(sentence):
    char_count = {char: sentence.count(char) for char in unique_char}
    total_count = sum(char_count.values())
    char_z_scores = zscore(list(char_count.values()))

    char_z_scores_dict = {char: score for char, score in zip(unique_char, char_z_scores)}
    return char_z_scores_dict
```

using function

In [27]:
```
columns = {char: [] for char in unique_char}

for index, row in df.iterrows():
    sentence = row['جملات']
    sentence = sentence.replace("ى" ,"ي")
    sentence = sentence.replace("ت","ة")
    sentence = sentence.replace("ا","إ")
    sentence = sentence.replace("ا","آ")
    sentence = sentence.replace(" ","")
    char_z_scores = bag_of_characters_z_score(sentence)

    for char in unique_char:
        columns[char].append(char_z_scores.get(char, 0))
```

Creating new data frame for Z score Normalized BoW

In [28]:
```
result_df4 = pd.DataFrame(columns)
result_df4['جملات'] = df['جملات']
result_df4["زبـان"]=df["زبـان"]
```

Some checks on Length Normalized BoW data fram and save it to excel file

In [29]:
```
result_df4
```

Out[29]:

	ا	گ	غ	ي	ص	ق	ط	إ	ك	ت	...	ف	
0	2.288556	-0.708845	-0.708845	-0.708845	-0.708845	-0.708845	0.040505	-0.708845	0.040505	1.539206	...	0.040505	-0.7(
1	4.603421	-0.470137	-0.470137	-0.470137	-0.470137	0.254657	0.254657	-0.470137	-0.470137	2.429039	...	0.254657	-0.4
2	2.605773	-0.586730	-0.586730	-0.586730	-0.586730	-0.586730	-0.586730	-0.586730	-0.586730	1.967272	...	-0.586730	-0.5
3	4.437394	-0.521170	-0.521170	-0.521170	-0.521170	-0.521170	-0.521170	-0.521170	-0.521170	0.029781	...	-0.521170	-0.5
4	4.522965	-0.582619	-0.015332	-0.582619	-0.015332	-0.582619	-0.582619	-0.582619	-0.582619	1.119242	...	-0.015332	-0.5
...	...	...	...	...	...	...	...	...	...	...	...	...	
95	3.645564	0.389698	-0.540549	-0.540549	-0.540549	-0.540549	-0.540549	-0.540549	-0.540549	-0.075425	...	-0.540549	-0.0
96	3.203961	-0.747591	-0.747591	-0.747591	-0.747591	-0.747591	-0.183083	-0.747591	-0.747591	0.945931	...	-0.183083	-0.1
97	3.706246	-0.789856	-0.789856	-0.789856	-0.789856	0.708845	-0.040505	-0.789856	-0.789856	0.708845	...	-0.789856	-0.0

98	3.003024	-0.716202	-0.251299	-0.716202	-0.251299	-0.716202	-0.716202	-0.716202	-0.716202	1.143411	...	-0.251299	-0.7
99	4.531515	-0.627441	-0.627441	-0.627441	0.017429	-0.627441	0.017429	-0.627441	-0.627441	0.662298	...	-0.627441	-0.6

100 rows × 39 columns

In [30]:
```
result_df4.head()
```

Out[30]:

	ا	گ	غ	ي	ص	ق	ط	إ	ك	ت	...	ف	
0	2.288556	-0.708845	-0.708845	-0.708845	-0.708845	-0.708845	0.040505	-0.708845	0.040505	1.539206	...	0.040505	-0.7088
1	4.603421	-0.470137	-0.470137	-0.470137	-0.470137	0.254657	0.254657	-0.470137	-0.470137	2.429039	...	0.254657	-0.470
2	2.605773	-0.586730	-0.586730	-0.586730	-0.586730	-0.586730	-0.586730	-0.586730	-0.586730	1.967272	...	-0.586730	-0.5867
3	4.437394	-0.521170	-0.521170	-0.521170	-0.521170	-0.521170	-0.521170	-0.521170	-0.521170	0.029781	...	-0.521170	-0.521
4	4.522965	-0.582619	-0.015332	-0.582619	-0.015332	-0.582619	-0.582619	-0.582619	-0.582619	1.119242	...	-0.015332	-0.5826

5 rows × 39 columns

Changing Arabic to 0 and Persian to 1

In [31]:
```
result_df4.replace({ 'عربی'} :'زبان':"Arabic" , 'فارسی':"Persian"} } ,inplace=True)
result_df4.to_excel("Zscore_Normalized_BoW.xlsx",index=False)
```

Split Data to test And Train

For Binary BoW

In [32]:
```
binary_bow_df = pd.read_excel("binaryBoW.xlsx")
X=binary_bow_df.drop("جملات",axis=1)
y = binary_bow_df['زبان']
X_test = binary_bow_df.iloc[[10, 20, 30, 40, 50, 60, 70, 80, 90]]
y_test = binary_bow_df.iloc[[10, 20, 30, 40, 50, 60, 70, 80, 90]]["زبان"]
X_test=X_test.drop('جملات', axis=1)
X_test=X_test.drop('زبان',axis=1)
X_train = X.drop([10, 20, 30, 40, 50, 60, 70, 80, 90])
y_train=y.drop([10, 20, 30, 40, 50, 60, 70, 80, 90])
X_train=X_train.drop('زبان',axis=1)
```

For Weighten BoW

In [33]:
```
weighten_bow_df = pd.read_excel("WeightenBoW.xlsx")
X2=weighten_bow_df.drop("جملات",axis=1)
y2 = weighten_bow_df['زبان']
X2_test = weighten_bow_df.iloc[[10, 20, 30, 40, 50, 60, 70, 80, 90]]
y2_test = weighten_bow_df.iloc[[10, 20, 30, 40, 50, 60, 70, 80, 90]]["زبان"]
X2_test=X2_test.drop('جملات', axis=1)
X2_test=X2_test.drop('زبان',axis=1)
X2_train = X2.drop([10, 20, 30, 40, 50, 60, 70, 80, 90])
y2_train=y2.drop([10, 20, 30, 40, 50, 60, 70, 80, 90])
X2_train=X2_train.drop('زبان',axis=1)
```

For Length Normalized BoW

In [34]:
```
length_normalized_bow_df = pd.read_excel("Length_Normalized_BoW.xlsx")
X3=length_normalized_bow_df.drop("جملات",axis=1)
y3 = weighten_bow_df['زبان']
X3_test = length_normalized_bow_df.iloc[[10, 20, 30, 40, 50, 60, 70, 80, 90]]
y3_test = length_normalized_bow_df.iloc[[10, 20, 30, 40, 50, 60, 70, 80, 90]]["زبان"]
X3_test=X3_test.drop('جملات', axis=1)
X3_test=X3_test.drop('زبان',axis=1)
X3_train = X3.drop([10, 20, 30, 40, 50, 60, 70, 80, 90])
y3_train=y3.drop([10, 20, 30, 40, 50, 60, 70, 80, 90])
X3_train=X3_train.drop('زبان',axis=1)
```

For ZScore Normalized BoW

In [35]:
```
zscore_bow_df = pd.read_excel("Zscore_Normalized_BoW.xlsx")
X4=zscore_bow_df.drop("جملات",axis=1)
y4 = zscore_bow_df['زبان']
X4_test = zscore_bow_df.iloc[[10, 20, 30, 40, 50, 60, 70, 80, 90]]
y4_test = zscore_bow_df.iloc[[10, 20, 30, 40, 50, 60, 70, 80, 90]]["زبان"]
X4_test=X4_test.drop('جملات', axis=1)
X4_test=X4_test.drop('زبان',axis=1)
X4_train = X4.drop([10, 20, 30, 40, 50, 60, 70, 80, 90])
y4_train=y4.drop([10, 20, 30, 40, 50, 60, 70, 80, 90])
X4_train=X4_train.drop('زبان',axis=1)
```

Train our Model

PART 1 : Binary BoW

In [41]:
```
distance_metrics = ['cosine', 'euclidean']
error_report = []

for k in [1, 3, 5]:
    for distance_metric in distance_metrics:
        knn_model = KNeighborsClassifier(n_neighbors=k, metric=distance_metric)
        knn_model.fit(X_train, y_train)
        y_pred = knn_model.predict(X_test)
        incorrect_predictions = sum(y_pred != y_test)
        error_report.append({
            'k': k,
            'distance_metric': distance_metric,
            'incorrect_predictions': incorrect_predictions,
            'accuracy': accuracy_score(y_test, y_pred)*100
        })
error_df = pd.DataFrame(error_report)
print(error_df)
```

```
   k distance_metric  incorrect_predictions    accuracy
0  1          cosine                      0  100.000000
1  1       euclidean                      1   88.888889
2  3          cosine                      0  100.000000
3  3       euclidean                      0  100.000000
4  5          cosine                      0  100.000000
5  5       euclidean                      1   88.888889
```

PART 2 : Weighten BoW

In [42]:
```
distance_metrics = ['cosine', 'euclidean']
error_report = []

for k in [1, 3, 5]:
    for distance_metric in distance_metrics:
        knn_model = KNeighborsClassifier(n_neighbors=k, metric=distance_metric)
        knn_model.fit(X2_train, y2_train)
        y2_pred = knn_model.predict(X2_test)
        incorrect_predictions = sum(y2_pred != y2_test)
        error_report.append({
            'k': k,
            'distance_metric': distance_metric,
            'incorrect_predictions': incorrect_predictions,
            'accuracy': accuracy_score(y2_test, y2_pred)*100
        })
error_df = pd.DataFrame(error_report)
print(error_df)
```

```
   k distance_metric  incorrect_predictions    accuracy
0  1          cosine                      0  100.000000
1  1       euclidean                      0  100.000000
2  3          cosine                      0  100.000000
3  3       euclidean                      1   88.888889
4  5          cosine                      0  100.000000
5  5       euclidean                      0  100.000000
```

PART 3 : Length Normalized BoW

In [43]:
```
distance_metrics = ['cosine', 'euclidean']
error_report = []

for k in [1, 3, 5]:
    for distance_metric in distance_metrics:
        knn_model = KNeighborsClassifier(n_neighbors=k, metric=distance_metric)
        knn_model.fit(X3_train, y3_train)
        y3_pred = knn_model.predict(X3_test)
        incorrect_predictions = sum(y3_pred != y3_test)
        error_report.append({
            'k': k,
            'distance_metric': distance_metric,
            'incorrect_predictions': incorrect_predictions,
            'accuracy': accuracy_score(y3_test, y3_pred)*100
        })
error_df = pd.DataFrame(error_report)
print(error_df)
```

```
   k distance_metric  incorrect_predictions  accuracy
0  1          cosine                      0     100.0
1  1       euclidean                      0     100.0
2  3          cosine                      0     100.0
3  3       euclidean                      0     100.0
4  5          cosine                      0     100.0
5  5       euclidean                      0     100.0
```

PART 4 : ZScore Normalized BoW

In [44]:
```
distance_metrics = ['cosine', 'euclidean']

error_report = []

for k in [1, 3, 5]:
    for distance_metric in distance_metrics:
        knn_model = KNeighborsClassifier(n_neighbors=k, metric=distance_metric)
        knn_model.fit(X4_train, y4_train)
        y4_pred = knn_model.predict(X4_test)
        incorrect_predictions = sum(y4_pred != y4_test)
        error_report.append({
            'k': k,
            'distance_metric': distance_metric,
            'incorrect_predictions': incorrect_predictions,
            'accuracy': accuracy_score(y4_test, y4_pred)*100
        })
error_df = pd.DataFrame(error_report)
print(error_df)
```

```
   k distance_metric  incorrect_predictions  accuracy
0  1          cosine                      0     100.0
1  1       euclidean                      0     100.0
2  3          cosine                      0     100.0
3  3       euclidean                      0     100.0
4  5          cosine                      0     100.0
5  5       euclidean                      0     100.0
```

After Drop Some Features

Droping This List : ['ة','ي','آ','ژ','گ','پ','چ']

PART 1 : Binary BoW

In [50]:
```
binary_bow_df = pd.read_excel("binaryBoW.xlsx")
X=binary_bow_df.drop(['ة','ي','آ','ژ','گ','پ','چ'],axis=1,inplace=True)
X=binary_bow_df.drop("جملات",axis=1)
y = binary_bow_df['زبان']
X_test = binary_bow_df.iloc[[10, 20, 30, 40, 50, 60, 70, 80, 90]]
y_test = binary_bow_df.iloc[[10, 20, 30, 40, 50, 60, 70, 80, 90]]["زبان"]
X_test=X_test.drop('جملات', axis=1)
X_test=X_test.drop('زبان',axis=1)
X_train = X.drop([10, 20, 30, 40, 50, 60, 70, 80, 90])
y_train=y.drop([10, 20, 30, 40, 50, 60, 70, 80, 90])
X_train=X_train.drop('زبان',axis=1)
```

In [51]:
```
distance_metrics = ['cosine', 'euclidean','minkowski']
error_report = []

for k in [1, 3, 5]:
    for distance_metric in distance_metrics:
        knn_model = KNeighborsClassifier(n_neighbors=k, metric=distance_metric)
        knn_model.fit(X_train, y_train)
        y_pred = knn_model.predict(X_test)
        incorrect_predictions = sum(y_pred != y_test)
        error_report.append({
            'k': k,
            'distance_metric': distance_metric,
            'incorrect_predictions': incorrect_predictions,
            'accuracy': accuracy_score(y_test, y_pred)
        })
error_df = pd.DataFrame(error_report)
print(error_df)
```

```
   k distance_metric  incorrect_predictions  accuracy
0  1          cosine                      2  0.777778
1  1       euclidean                      3  0.666667
2  1       minkowski                      3  0.666667
3  3          cosine                      1  0.888889
4  3       euclidean                      1  0.888889
5  3       minkowski                      1  0.888889
6  5          cosine                      1  0.888889
7  5       euclidean                      1  0.888889
8  5       minkowski                      1  0.888889
```

PART 2 : Weighten BoW

In [54]:
```
weighten_bow_df = pd.read_excel("WeightenBoW.xlsx")
X2=weighten_bow_df.drop(['ة','ي','آ','ژ','گ','پ','چ'],axis=1,inplace=True)
X2=weighten_bow_df.drop("جملات",axis=1)
y2 = weighten_bow_df['زبان']
X2_test = weighten_bow_df.iloc[[10, 20, 30, 40, 50, 60, 70, 80, 90]]
y2_test = weighten_bow_df.iloc[[10, 20, 30, 40, 50, 60, 70, 80, 90]]["زبان"]
X2_test=X2_test.drop('جملات', axis=1)
X2_test=X2_test.drop('زبان',axis=1)
X2_train = X2.drop([10, 20, 30, 40, 50, 60, 70, 80, 90])
y2_train=y2.drop([10, 20, 30, 40, 50, 60, 70, 80, 90])
X2_train=X2_train.drop('زبان',axis=1)
```

In [55]:
```
distance_metrics = ['cosine', 'euclidean','minkowski']
error_report = []

for k in [1, 3, 5]:
    for distance_metric in distance_metrics:
        knn_model = KNeighborsClassifier(n_neighbors=k, metric=distance_metric)
        knn_model.fit(X2_train, y2_train)
        y2_pred = knn_model.predict(X2_test)
        incorrect_predictions = sum(y2_pred != y2_test)
        error_report.append({
            'k': k,
            'distance_metric': distance_metric,
            'incorrect_predictions': incorrect_predictions,
            'accuracy': accuracy_score(y2_test, y2_pred)*100
        })
error_df = pd.DataFrame(error_report)
print(error_df)
```

	k	distance_metric	incorrect_predictions	accuracy
0	1	cosine	0	100.000000
1	1	euclidean	1	88.888889
2	1	minkowski	1	88.888889
3	3	cosine	0	100.000000
4	3	euclidean	1	88.888889
5	3	minkowski	1	88.888889
6	5	cosine	0	100.000000
7	5	euclidean	0	100.000000
8	5	minkowski	0	100.000000

PART 3 : Length Normalized BoW

In [56]:
```
length_normalized_bow_df = pd.read_excel("Length_Normalized_BoW.xlsx")
X3=length_normalized_bow_df.drop(['ة','ي','آ','ژ','گ','پ','چ'],axis=1,inplace=True)
X3=length_normalized_bow_df.drop("جملات",axis=1)
y3 = weighten_bow_df['زبان']
X3_test = length_normalized_bow_df.iloc[[10, 20, 30, 40, 50, 60, 70, 80, 90]]
y3_test = length_normalized_bow_df.iloc[[10, 20, 30, 40, 50, 60, 70, 80, 90]]["زبان"]
X3_test=X3_test.drop('جملات', axis=1)
X3_test=X3_test.drop('زبان',axis=1)
X3_train = X3.drop([10, 20, 30, 40, 50, 60, 70, 80, 90])
y3_train=y3.drop([10, 20, 30, 40, 50, 60, 70, 80, 90])
X3_train=X3_train.drop('زبان',axis=1)
```

In [57]:
```
distance_metrics = ['cosine', 'euclidean','minkowski']
error_report = []

for k in [1, 3, 5]:
    for distance_metric in distance_metrics:
        knn_model = KNeighborsClassifier(n_neighbors=k, metric=distance_metric)
        knn_model.fit(X3_train, y3_train)
        y3_pred = knn_model.predict(X3_test)
        incorrect_predictions = sum(y3_pred != y3_test)
        error_report.append({
            'k': k,
            'distance_metric': distance_metric,
            'incorrect_predictions': incorrect_predictions,
            'accuracy': accuracy_score(y3_test, y3_pred)*100
        })
error_df = pd.DataFrame(error_report)
print(error_df)
```

```
   k distance_metric  incorrect_predictions  accuracy
0  1          cosine                      0     100.0
1  1       euclidean                      0     100.0
2  1       minkowski                      0     100.0
3  3          cosine                      0     100.0
4  3       euclidean                      0     100.0
5  3       minkowski                      0     100.0
6  5          cosine                      0     100.0
7  5       euclidean                      0     100.0
8  5       minkowski                      0     100.0
```

PART 4 : ZScore Normalized BoW

In [58]:
```
zscore_bow_df = pd.read_excel("Zscore_Normalized_BoW.xlsx")
X4=zscore_bow_df.drop(['ة','ي','آ','ژ','گ','پ','چ'],axis=1,inplace=True)
X4=zscore_bow_df.drop("جملات",axis=1)
y4 = zscore_bow_df['زبان']
X4_test = zscore_bow_df.iloc[[10, 20, 30, 40, 50, 60, 70, 80, 90]]
y4_test = zscore_bow_df.iloc[[10, 20, 30, 40, 50, 60, 70, 80, 90]]["زبان"]
X4_test=X4_test.drop('جملات', axis=1)
X4_test=X4_test.drop('زبان',axis=1)
X4_train = X4.drop([10, 20, 30, 40, 50, 60, 70, 80, 90])
y4_train=y4.drop([10, 20, 30, 40, 50, 60, 70, 80, 90])
X4_train=X4_train.drop('زبان',axis=1)
```

In [59]:
```
distance_metrics = ['cosine', 'euclidean','minkowski']

error_report = []

for k in [1, 3, 5]:
    for distance_metric in distance_metrics:
        knn_model = KNeighborsClassifier(n_neighbors=k, metric=distance_metric)
        knn_model.fit(X4_train, y4_train)
        y4_pred = knn_model.predict(X4_test)
        incorrect_predictions = sum(y4_pred != y4_test)
        error_report.append({
            'k': k,
            'distance_metric': distance_metric,
            'incorrect_predictions': incorrect_predictions,
            'accuracy': accuracy_score(y4_test, y4_pred)*100
        })
error_df = pd.DataFrame(error_report)
print(error_df)
```

```
   k distance_metric  incorrect_predictions  accuracy
0  1          cosine                      0     100.0
1  1       euclidean                      0     100.0
2  1       minkowski                      0     100.0
3  3          cosine                      0     100.0
4  3       euclidean                      0     100.0
5  3       minkowski                      0     100.0
6  5          cosine                      0     100.0
7  5       euclidean                      0     100.0
8  5       minkowski                      0     100.0
```

Spliting Train and Test 70 to 30

PART 1 : Binary BoW

In [84]:
```
X5_train, X5_test, y5_train, y5_test = train_test_split(X,y, test_size=0.3)
```

In [85]:
```
X5_test=X5_test.drop("زبان",axis=1)
```

In [86]:
```
X5_train=X5_train.drop("زبان",axis=1)
```

In [91]:
```
distance_metrics = ['cosine', 'euclidean','minkowski']

error_report = []

for k in [1, 3, 5]:
    for distance_metric in distance_metrics:
        knn_model = KNeighborsClassifier(n_neighbors=k, metric=distance_metric)
        knn_model.fit(X5_train, y5_train)
        y5_pred = knn_model.predict(X5_test)
        incorrect_predictions = sum(y5_pred != y5_test)
        error_report.append({
            'k': k,
            'distance_metric': distance_metric,
            'incorrect_predictions': incorrect_predictions,
            'accuracy': accuracy_score(y5_test, y5_pred)*100
        })
error_df = pd.DataFrame(error_report)
print(error_df)
```

```
   k distance_metric  incorrect_predictions   accuracy
0  1          cosine                      9  70.000000
1  1       euclidean                      9  70.000000
2  1       minkowski                      9  70.000000
3  3          cosine                      4  86.666667
4  3       euclidean                      5  83.333333
5  3       minkowski                      5  83.333333
6  5          cosine                      4  86.666667
7  5       euclidean                      4  86.666667
8  5       minkowski                      4  86.666667
```

PART 2 : Weighten BoW

In [100…
```
X6_train, X6_test, y6_train, y6_test = train_test_split(X2,y2, test_size=0.3)
```

In [101…
```
X6_test=X6_test.drop("زبان",axis=1)
X6_train=X6_train.drop("زبان",axis=1)
```

In [102…
```
distance_metrics = ['cosine', 'euclidean','minkowski']

error_report = []

for k in [1, 3, 5]:
    for distance_metric in distance_metrics:
        knn_model = KNeighborsClassifier(n_neighbors=k, metric=distance_metric)
        knn_model.fit(X6_train, y6_train)
        y6_pred = knn_model.predict(X6_test)
        incorrect_predictions = sum(y6_pred != y6_test)
        error_report.append({
            'k': k,
            'distance_metric': distance_metric,
            'incorrect_predictions': incorrect_predictions,
            'accuracy': accuracy_score(y6_test, y6_pred)*100
        })
error_df = pd.DataFrame(error_report)
print(error_df)
```

```
   k distance_metric  incorrect_predictions   accuracy
0  1          cosine                      3  90.000000
1  1       euclidean                      3  90.000000
2  1       minkowski                      3  90.000000
3  3          cosine                      3  90.000000
4  3       euclidean                      2  93.333333
5  3       minkowski                      2  93.333333
6  5          cosine                      2  93.333333
7  5       euclidean                      2  93.333333
8  5       minkowski                      2  93.333333
```

PART 3 : Length Normalized BoW

In [103…
```
X7_train, X7_test, y7_train, y7_test = train_test_split(X3,y3, test_size=0.3)
```

In [104…
```
X7_test=X7_test.drop("زبان",axis=1)
X7_train=X7_train.drop("زبان",axis=1)
```

In [105…
```
distance_metrics = ['cosine', 'euclidean','minkowski']

error_report = []

for k in [1, 3, 5]:
    for distance_metric in distance_metrics:
        knn_model = KNeighborsClassifier(n_neighbors=k, metric=distance_metric)
        knn_model.fit(X7_train, y7_train)
        y7_pred = knn_model.predict(X7_test)
        incorrect_predictions = sum(y7_pred != y7_test)
        error_report.append({
            'k': k,
            'distance_metric': distance_metric,
            'incorrect_predictions': incorrect_predictions,
            'accuracy': accuracy_score(y7_test, y7_pred)*100
        })
error_df = pd.DataFrame(error_report)
print(error_df)
```

```
   k distance_metric  incorrect_predictions   accuracy
0  1          cosine                      4  86.666667
1  1       euclidean                      7  76.666667
2  1       minkowski                      7  76.666667
3  3          cosine                      4  86.666667
4  3       euclidean                      3  90.000000
5  3       minkowski                      3  90.000000
6  5          cosine                      2  93.333333
7  5       euclidean                      2  93.333333
8  5       minkowski                      2  93.333333
```

PART 4 : ZScore Normalized BoW

In [110…
```
X8_train, X8_test, y8_train, y8_test = train_test_split(X4,y4, test_size=0.3)
```

In [111…
```
X8_test=X8_test.drop("زبان",axis=1)
X8_train=X8_train.drop("زبان",axis=1)
```

In [112…
```
distance_metrics = ['cosine', 'euclidean','minkowski']

error_report = []

for k in [1, 3, 5]:
    for distance_metric in distance_metrics:
        knn_model = KNeighborsClassifier(n_neighbors=k, metric=distance_metric)
        knn_model.fit(X8_train, y8_train)
        y8_pred = knn_model.predict(X8_test)
        incorrect_predictions = sum(y8_pred != y8_test)
        error_report.append({
            'k': k,
            'distance_metric': distance_metric,
            'incorrect_predictions': incorrect_predictions,
            'accuracy': accuracy_score(y8_test, y8_pred)*100
        })
error_df = pd.DataFrame(error_report)
print(error_df)
```

	k	distance_metric	incorrect_predictions	accuracy
0	1	cosine	1	96.666667
1	1	euclidean	2	93.333333
2	1	minkowski	2	93.333333
3	3	cosine	1	96.666667
4	3	euclidean	1	96.666667
5	3	minkowski	1	96.666667
6	5	cosine	1	96.666667
7	5	euclidean	1	96.666667
8	5	minkowski	1	96.666667