4a-NLP.KNN.clustering

November 8, 2024

[1]:
```
import numpy as np
import pandas as pd
from textblob import TextBlob
from sklearn.feature_extraction.text import CountVectorizer as BagOfWords
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.neighbors import NearestNeighbors
pd.options.display.max_columns = 100
```

1 NLP

If our text data are unlabelled (as is often the case in NLP), we can use KNN to identify documents that are similar to a given document. Let's test this by creating simple documents that are similar and different.

[2]:
```
%%capture
!python -m textblob.download_corpora
```

[3]:
```
sentences = []
sentences += [ 'alpaca ' * 5 + "zebra" ]
sentences += [ 'bird ' * 5 + "zebra" ]
sentences += [ 'cat ' * 5 + "zebra" ]
sentences += [ 'dog ' * 5 + "zebra" ]
sentences += [ "alpaca bird cat dog zebra" ]
sentences
```

[3]:
```
['alpaca alpaca alpaca alpaca alpaca zebra',
 'bird bird bird bird bird zebra',
 'cat cat cat cat cat zebra',
 'dog dog dog dog dog zebra',
 'alpaca bird cat dog zebra']
```

2 Data Cleaning

[4]:
```
# we can create data that needs cleaning
```

2.1 Bag of Words Using CountVectorizer

[5]:
```
# Perform the count transformation
BoW =  BagOfWords(stop_words='english')
vec = BoW.fit_transform(sentences)
vec.toarray()
```

[5]:
```
array([[5, 0, 0, 0, 1],
       [0, 5, 0, 0, 1],
       [0, 0, 5, 0, 1],
       [0, 0, 0, 5, 1],
       [1, 1, 1, 1, 1]])
```

[6]:
```
BoW.get_feature_names_out()
```

[6]:
```
array(['alpaca', 'bird', 'cat', 'dog', 'zebra'], dtype=object)
```

[7]:
```
pd.DataFrame( vec.toarray(), columns = BoW.get_feature_names_out() )
```

[7]:

	alpaca	bird	cat	dog	zebra
0	5	0	0	0	1
1	0	5	0	0	1
2	0	0	5	0	1
3	0	0	0	5	1
4	1	1	1	1	1

2.2 TF-IDF

[8]:
```
# Perform the TF-IDF transformation
tf_idf_vec = TfidfTransformer()
tf_idf_jen = tf_idf_vec.fit_transform(vec)
tf_idf_jen.toarray()
```

[8]:
```
array([[0.99309562, 0.        , 0.        , 0.        , 0.11730765],
       [0.        , 0.99309562, 0.        , 0.        , 0.11730765],
       [0.        , 0.        , 0.99309562, 0.        , 0.11730765],
       [0.        , 0.        , 0.        , 0.99309562, 0.11730765],
       [0.47952794, 0.47952794, 0.47952794, 0.47952794, 0.28321692]])
```

[9]:
```
# Print out results in a dataframe
tf_df = pd.DataFrame(tf_idf_jen.toarray(), columns = BoW.
 ↪get_feature_names_out())
tf_df
```

[9]:

	alpaca	bird	cat	dog	zebra
0	0.993096	0.000000	0.000000	0.000000	0.117308
1	0.000000	0.993096	0.000000	0.000000	0.117308
2	0.000000	0.000000	0.993096	0.000000	0.117308

```
3  0.000000  0.000000  0.000000  0.993096  0.117308
4  0.479528  0.479528  0.479528  0.479528  0.283217
```

2.3 K Nearest Neighbors

[10]:
```
# Fit nearest neighbors
nn = NearestNeighbors().fit(tf_idf_jen)
```

[11]:
```
# Get nearest neighbors distances
sent0 = np.array([tf_df.iloc[0]])
distances, indices = nn.kneighbors(sent0, n_neighbors = len(tf_df))
```

[12]:
```
distances
```

[12]:
```
array([[0.       , 0.9905144, 1.4044493, 1.4044493, 1.4044493]])
```

[13]:
```
indices
```

[13]:
```
array([[0, 4, 3, 2, 1]])
```

[14]:
```
for a,b in zip(distances[0], np.array(sentences)[indices][0]):
  print(f"{a:.4f}: {b}")
```

```
0.0000: alpaca alpaca alpaca alpaca alpaca zebra
0.9905: alpaca bird cat dog zebra
1.4044: dog dog dog dog dog zebra
1.4044: cat cat cat cat cat zebra
1.4044: bird bird bird bird bird zebra
```

[ ]:
```

```