NLP_lab-3_Mahalakshmi.S18

March 21, 2021

0.0.1 Lab3. Computing Document Similarity using VSM

0.0.2 EXERCISE-1: Print TFIDF values

[1]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
import pandas as pd
docs = [
 "good movie", "not a good movie", "did not like",
 "i like it", "good one" ]
# using default tokenizer in TfidfVectorizer
tfidf = TfidfVectorizer(min_df=2, max_df=0.5, ngram_range=(1, 2))
features = tfidf.fit_transform(docs)
print(features)
# Pretty printing
df = pd.DataFrame(
 features.todense(),
 columns=tfidf.get_feature_names())
print(df)
```

```
  (0, 0)        0.7071067811865476
  (0, 2)        0.7071067811865476
  (1, 3)        0.5773502691896257
  (1, 0)        0.5773502691896257
  (1, 2)        0.5773502691896257
  (2, 1)        0.7071067811865476
  (2, 3)        0.7071067811865476
  (3, 1)        1.0
   good movie      like     movie       not
0    0.707107  0.000000  0.707107  0.000000
1    0.577350  0.000000  0.577350  0.577350
2    0.000000  0.707107  0.000000  0.707107
3    0.000000  1.000000  0.000000  0.000000
4    0.000000  0.000000  0.000000  0.000000
```

0.0.3 EXERCISE-2:

1. Change the values of min_df and ngram_range and observe various outputs

[6]:
```
docs = [
 "good movie", "not a good movie", "did not like",
```

```
 "i like it", "good one" ]
# using default tokenizer in TfidfVectorizer
tfidf = TfidfVectorizer(min_df=1, max_df=0.5, ngram_range=(2, 1))
features = tfidf.fit_transform(docs)
print(features)
# Pretty printing
df = pd.DataFrame(
 features.todense(),
 columns=tfidf.get_feature_names())
print(df)
```

```
  (0, 3)        1.0
  (1, 3)        0.7071067811865476
  (1, 4)        0.7071067811865476
  (2, 4)        0.5317722537280788
  (2, 0)        0.6591180018251055
  (2, 2)        0.5317722537280788
  (3, 2)        0.6279137616509933
  (3, 1)        0.7782829228046183
  (4, 5)        1.0
       did        it      like     movie       not  one
0  0.000000  0.000000  0.000000  1.000000  0.000000  0.0
1  0.000000  0.000000  0.000000  0.707107  0.707107  0.0
2  0.659118  0.000000  0.531772  0.000000  0.531772  0.0
3  0.000000  0.778283  0.627914  0.000000  0.000000  0.0
4  0.000000  0.000000  0.000000  0.000000  0.000000  1.0
```

0.0.4 EXERCISE-3: Compute Cosine Similarity between 2 Documents

[2]:
```
from sklearn.metrics.pairwise import linear_kernel
# cosine score between 1st and 2nd doc
doc1 = features[0:1]
doc2 = features[1:2]
score = linear_kernel(doc1, doc2)
print(score)
```

```
[[0.81649658]]
```

[3]:
```
scores = linear_kernel(doc1, features)    # cosine score between 1st and all␣
 ↪other docs
print(scores)
```

```
[[1.         0.81649658 0.         0.         0.        ]]
```

[4]:
```
query = "I like this good movie"        # Cosine Similarity for a new doc
qfeature = tfidf.transform([query])
scores2 = linear_kernel(doc1, features)
print(scores2)
```

```
[[1.         0.81649658 0.         0.         0.        ]]
```

0.0.5 EXERCISE-4: Find Top-N similar documents

Question-1. Consider the following documents and compute TFIDF values

[8]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
import pandas as pd

# this is a very toy example, do not try this at home unless you want to␣
 ↪understand the usage differences
docs=["the house had a tiny little mouse",
"the cat saw the mouse",
"the mouse ran away from the house",
"the cat finally ate the mouse",
"the end of the mouse story"
]
# using default tokenizer in TfidfVectorizer
tfidf = TfidfVectorizer(min_df=1, max_df=0.5, ngram_range=(1, 1))
features = tfidf.fit_transform(docs)
print(features)
# Pretty printing
df = pd.DataFrame(
 features.todense(),
 columns=tfidf.get_feature_names())
print(df)
```

```
  (0, 8)        0.5233582502695435
  (0, 13)       0.5233582502695435
  (0, 6)        0.5233582502695435
  (0, 7)        0.4222421409859579
  (1, 11)       0.7782829228046183
  (1, 2)        0.6279137616509933
  (2, 5)        0.5233582502695435
  (2, 1)        0.5233582502695435
  (2, 10)       0.5233582502695435
  (2, 7)        0.4222421409859579
  (3, 0)        0.6141889663426562
  (3, 4)        0.6141889663426562
  (3, 2)        0.49552379079705033
  (4, 12)       0.5773502691896258
  (4, 9)        0.5773502691896258
  (4, 3)        0.5773502691896258
       ate      away       cat      end   finally      from       had  \
0  0.000000  0.000000  0.000000  0.00000  0.000000  0.000000  0.523358
1  0.000000  0.000000  0.627914  0.00000  0.000000  0.000000  0.000000
2  0.000000  0.523358  0.000000  0.00000  0.000000  0.523358  0.000000
3  0.614189  0.000000  0.495524  0.00000  0.614189  0.000000  0.000000
4  0.000000  0.000000  0.000000  0.57735  0.000000  0.000000  0.000000
```

	house	little	of	ran	saw	story	tiny
0	0.422242	0.523358	0.00000	0.000000	0.000000	0.00000	0.523358
1	0.000000	0.000000	0.00000	0.000000	0.778283	0.00000	0.000000
2	0.422242	0.000000	0.00000	0.523358	0.000000	0.00000	0.000000
3	0.000000	0.000000	0.00000	0.000000	0.000000	0.00000	0.000000
4	0.000000	0.000000	0.57735	0.000000	0.000000	0.57735	0.000000

0.0.6 Question-2. Compute cosine similarity between 3rddocument ("the mouse ran away from the house") with all other documents. Which is the most similar document?.

[14]:
```
docs=["the house had a tiny little mouse",
"the cat saw the mouse",
"the mouse ran away from the house",
"the cat finally ate the mouse",
"the end of the mouse story"
]
```

[16]:
```
tfidf = TfidfVectorizer(min_df=2, max_df=0.5, ngram_range=(1, 2))
features = tfidf.fit_transform(docs)
print(features)
```

```
  (0, 3)        0.7071067811865476
  (0, 1)        0.7071067811865476
  (1, 2)        0.7071067811865476
  (1, 0)        0.7071067811865476
  (2, 3)        0.7071067811865476
  (2, 1)        0.7071067811865476
  (3, 2)        0.7071067811865476
  (3, 0)        0.7071067811865476
```

[17]:
```
docs = features[0:3]
scores = linear_kernel(docs, features)    # cosine score between 1st and all␣
 ↪other docs
print(scores)
```

```
[[1. 0. 1. 0. 0.]
 [0. 1. 0. 1. 0.]
 [1. 0. 1. 0. 0.]]
```