Machine Learning in python

Text Processing

Introduction

Text Processing

- Mathematical models work with numbers, not text
- Text data need to be converted to numbers before it can be fed into mathematical models
- Convert text to numbers via Text Featurization

Terminologies

<u>Document</u>

- A collection of words
- Article, Email, SMS etc

<u>Corpus</u>

- A collection of documents of a certain theme (e.g. movie reviews, product reviews)
- A machine learning model for textual data usually depends on a corpus to learn the nuances of a subject matter

Terminologies

Term / Token

- Smallest processable unit (e.g. a word within a document)

Vocabulary

- The number of unique terms in a corpus
- The Vocabulary Size is the number of features that our model needs to learn from

Text Preparation

Text Cleansing

- Remove punctuations (e.g. full-stops, commas, exclamations)
- Convert all words to lowercase or uppercase (for consistent comparisons)
- Remove formatting (e.g. HTML tags)

Tokenization

- Tokenization splits a document into tokens or terms
- NLTK, a Natural Language Processing (NLP) tool, can perform tokenization on a string as shown

```
from nltk import word_tokenize

text = 'Hello this is a test.'

print(word_tokenize(text))
```

['Hello', 'this', 'is', 'a', 'test', '.']

Terms

Stemming and Lemmatization

Both Stemming and Lemmatization shorten a word to its root form

Stemming

- Uses rule-based heuristics
- Cuts off prefixes and/or ends of words
- Quicker, but the shorten word might not make sense

Lemmatization

- Uses a vocabulary for its transformation
- Considers the context and return an actual word in the vocabulary

Stemming example

```
from nltk import word_tokenize
from nltk.stem import SnowballStemmer

text = 'he likes cats and dogs, and teaching machines to learn'

stem = SnowballStemmer(language='english')

print([stem.stem(token) for token in word_tokenize(text)])
```

['he', 'like', 'cat', 'and', 'dog', ',', 'and', 'teach', 'machin', 'to', 'learn']

Not a word in the dictionary

Lemmatization example

```
from nltk import word_tokenize
from nltk.stem import WordNetLemmatizer

text = 'he likes cats and dogs, and teaching machines to learn'

lm = WordNetLemmatizer()

print([lm.lemmatize(token, 'v') for token in word_tokenize(text)])
```

['he', 'like', 'cat', 'and', 'dog', ',', 'and', 'teach', 'machine', 'to', 'learn']

An actual word in the dictionary

Stop Words

- Stop Words are words that are very common and deemed as not providing differentiating value when fed into models
- In Text Processing, it is a common practice to remove stop words from our corpus before doing further processing
- However, there is no universal list of stop words; every Natural Language Processing (NLP) tool has its own

Stop Words

- NLTK’s English stop words are shown below

```
from nltk.corpus import stopwords

print(stopwords.words('english'))
```

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'll", "you'd", 'your', 'yours',
'yourself', 'yourselves', 'he', 'him', 'his', 'himself', 'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they',
'them', 'their', 'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those', 'am', 'is',
'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', 'did', 'doing', 'a', 'an', 'the',
'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', 'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into',
'through', 'during', 'before', 'after', 'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under',
'again', 'further', 'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few',
'more', 'most', 'other', 'some', 'such', 'no', 'nor', 'not', 'only', 'own', 'same', 'so', 'than', 'too', 'very', 's', 't', 'can',
'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o', 're', 've', 'y', 'ain', 'aren', "aren't", 'couldn',
"couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn', "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't",
'ma', 'mightn', "mightn't", 'mustn', "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn',
"wasn't", 'weren', "weren't", 'won', "won't", 'wouldn', "wouldn't"]
```

Stop Words

- An example where stop words and punctuations are filtered away using NLTK's list of stop words

```
from nltk import word_tokenize
from nltk.corpus import stopwords
import string

text = 'he likes cats and dogs, and teaching machines to learn'

stops = stopwords.words('english')

punc = str.maketrans('', '', string.punctuation)
text_no_punc = text.translate(punc)

terms = [token for token in word_tokenize(text_no_punc) if token not in stops]

print(terms)
```

['likes', 'cats', 'dogs', 'teaching', 'machines', 'learn']

Information Extraction

Part-of-Speech & Named Entities

Part-of-Speech

- A sentence can have extra grammatical properties tagged to it. Those tagged properties are called Part-of-Speech (POS) tags
- The POS tags of each word in our sentence

```
import nltk
from nltk.tokenize import word_tokenize

text = word_tokenize("Mary has a little lamb")
pos = nltk.pos_tag(text)
print(pos)
```

[('Mary', 'NNP'), ('has', 'VBZ'), ('a', 'DT'), ('little', 'JJ'), ('lamb', 'NN')]

Part–of-Speech (POS)

Part-of-Speech

- Meanings of the POS tags in our sentence

Word	Abbreviation	Meaning
Mary	NNP	Proper noun, Singular (e.g. Sarah)
has	VBZ	Verb, Present Tense
a	DT	Determiner (e.g. a, the)
little	JJ	Adjective (e.g. large, red)
lamb	NN	Noun, Singular (e.g. cat, tree)

✤ A complete table of abbreviation/meaning of NLTK's POS tags can be found here - https://www.guru99.com/pos-tagging-chunking-nltk.html

Named Entity

- Named Entities, such as dates, company names, locations can further be identified within a sentence
- Such add-on data provides your application, e.g. chatbot, with useful context to better understand a sentence

To further elaborate on the geographical trends, North America LOC has procured more than 50% PERCENT of the global share in 2017 DATE and has been leading the regional landscape of AI GPE in the retail market. The U.S. GPE has a significant credit in the regional trends with over 65% PERCENT of investments (including M&As, private equity, and venture capital) in artificial intelligence technology. Additionally, the region is a huge hub for startups in tandem with the presence of tech titans, such as Google ORG , IBM ORG , and Microsoft ORG .

Text Featurization

Bag of Words (BOW)

Text Featurization

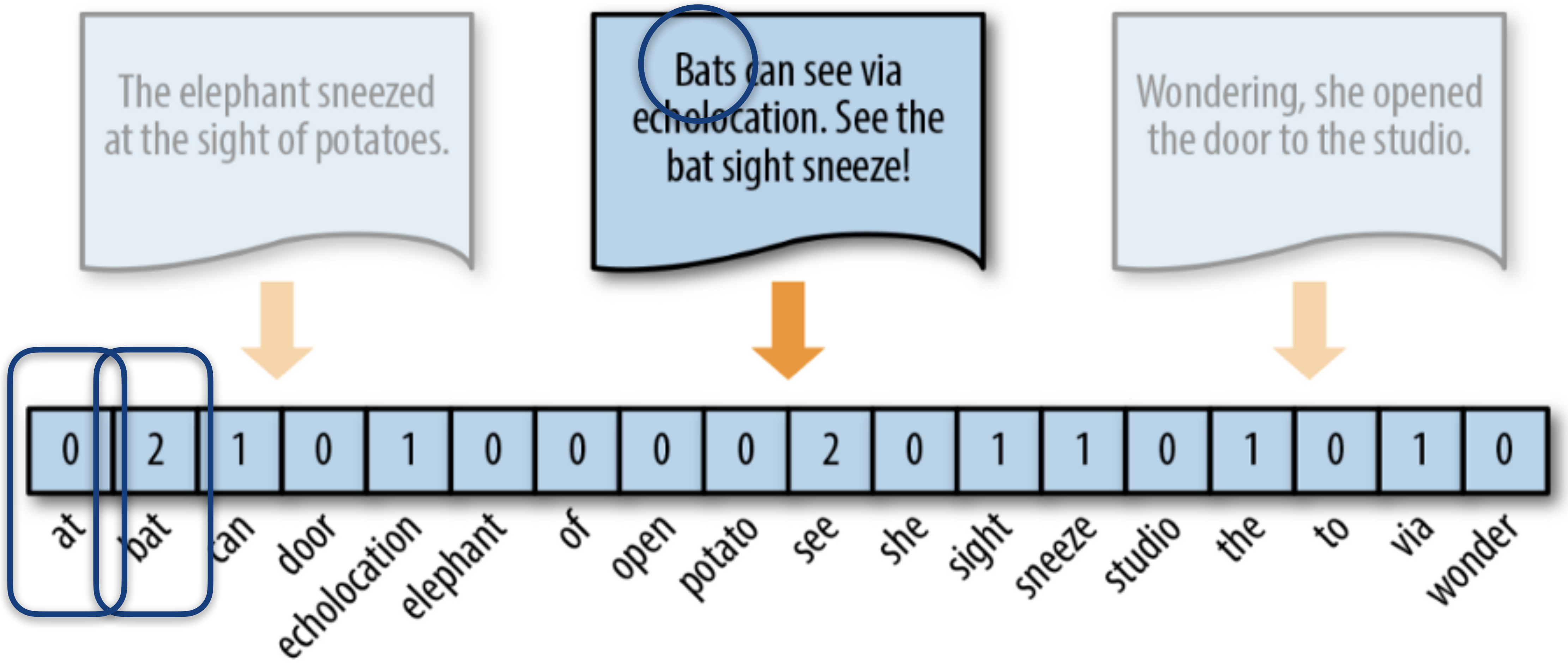
- Mathematical Models only works with numbers
- In order for text to be useful to our models, those text need to be converted to some meaningful numbers (features)
- Common techniques in generating features for text
    - Bag of Words (BOW)
    - TF-IDF

Bag of Words (BOW)

- Bag of Words counts the frequencies of words in a corpus
- BOW discards grammar such as present and past tenses, hence the context might be lost
- BOW also discards word-order, hence understanding the meaning of a sentence is difficult
    - “He genuinely needs to do that.”
    - “He needs to do that genuinely.”

Bag of Words (BOW)

- Each document has a vector to track words found within itself
- Each unique word in the Corpus is mapped to the same position in all the vectors



BOW example

doc1: John has some cats
doc2: Cats, being cats, eat fish
doc3: I ate a big fish

stop-words removal

[has, some, being, I, a]

doc1: John cats
doc2: Cats cats eat fish
doc3: ate big fish

lemmatization

doc1: john cat
doc2: cat cat eat fish
doc3: eat big fish

BOW example

doc1: john cat
doc2: cat cat eat fish
doc3: eat big fish

Vocabulary

[big, cat, eat, fish, john]

Pos-to-Word mapping for Feature Vectors

Index	0	1	2	3	4
Word	big	cat	eat	fish	john

BOW example

doc1: john cat
doc2: cat cat eat fish
doc3: eat big fish

Compute Bag-of-Words (BOW)

	big	cat	eat	fish	john
doc1	0	1	0	0	1
doc2	0	2	1	1	0
doc3	1	0	1	1	0

Feature Vectors

BOW (in code)

- Download required NLTK data – its list of stop-words and wordnet (database of English words)
- Setup Lemmatizer to shorten words to root form

```
import string
import pandas as pd
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from sklearn.feature_extraction.text import CountVectorizer

# can be removed if resources reside in your system
nltk.download('stopwords')
nltk.download('wordnet')

lemmatizer = WordNetLemmatizer()
stop_words = stopwords.words('english')

docs = [
    'John has some cats.',
    'Cats, being cats, eat fish.',
    'I ate a big fish.'
]
```

BOW (in code)

- Next, we filter out punctuations, stop-words, and perform Lemmatization
- str.maketrans(…) creates a mapping of punctuation-symbols to None, and doc.translate(…) performs substitution using that mapping

```
docs_clean = []
punc = str.maketrans('', '', string.punctuation)

# pre-process each document
for doc in docs:
    doc_no_punc = doc.translate(punc)
    words = doc_no_punc.lower().split()

    words = [lemmatizer.lemmatize(word, 'v')
                for word in words if word not in stop_words]

    docs_clean.append(' '.join(words))
```

BOW (in code)

- Finally, we generate feature vectors for all documents and align them with the vocabulary in our corpus

```
# generate feature vectors using BOW
bow = CountVectorizer()

feature_vectors = bow.fit_transform(docs_clean).toarray()
vocab = bow.get_feature_names()

df = pd.DataFrame(data=feature_vectors,
                  index=['doc1', 'doc2', 'doc3'],
                  columns=vocab)

print(df)
```

	big	cat	eat	fish	john
doc1	0	1	0	0	1
doc2	0	2	1	1	0
doc3	1	0	1	1	0

Entire code

```
import string
import pandas as pd
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from sklearn.feature_extraction.text import CountVectorizer

# can be removed if resources reside in your system
nltk.download('stopwords')
nltk.download('wordnet')

lemmatizer = WordNetLemmatizer()
stop_words = stopwords.words('english')

docs = [
    'John has some cats.',
    'Cats, being cats, eat fish.',
    'I ate a big fish.'
]

# data cleansing
docs_clean = []
punc = str.maketrans('', '', string.punctuation)
for doc in docs:
    doc_no_punc = doc.translate(punc)
    words = doc_no_punc.lower().split()

    words = [lemmatizer.lemmatize(word, 'v')
                for word in words if word not in stop_words]

    docs_clean.append(' '.join(words))
```

```
# generate feature vectors using BOW
bow = CountVectorizer()

feature_vectors = bow.fit_transform(docs_clean).toarray()
vocab = bow.get_feature_names()

df = pd.DataFrame(data=feature_vectors,
                  index=['doc1', 'doc2', 'doc3'],
                  columns=vocab)

print(df)
```

Text Featurization

TF-IDF

TF-IDF

- TF-IDF measures the importance of terms with respect to documents within a corpus

$$\text{TF-IDF}(t, d) = \text{TF}(t, d) \times \text{IDF}(t, c)$$

How frequent a term appears in a document

How infrequent the same term appears in our corpus

TF-IDF

- Term Frequency (TF): The number of times a term appears in a document
- TF can easily be implemented using Bag of Words

$$\text{TF-IDF} = \underbrace{\text{TF}(t, d)}_{\text{No. of times a term t appears in a doc d}} \times \text{IDF}(t, c)$$

TF-IDF

- Inverse Document Frequency (IDF): The IDF of a word is the inverse of the number of times it appears in the corpus at least once
- The fewer times a term appears in the corpus, the more important it is considered

$$\text{TF-IDF} = \text{TF}(t, d) \times \text{IDF}(t, c)$$

The inverse of the no. of times term t appears in corpus c at least once

TF-IDF

- There is only one IDF value for each unique term in a corpus
- Compute IDF with this formula

term

corpus

total no. of documents in our corpus

$$IDF(t, c) = \ln \frac{1 + n}{1 + df(t, c)} + 1$$

natural log

document frequency (no. of times the term appears in our corpus at least once

Normalized TF-IDF

- A long document may have high frequencies for certain terms due to its length
- Given a query string, a long document may seem more relevant than a short document if compared using absolute term frequencies
- Normalization gives us relative term frequencies instead of absolute term counts

Normalized TF-IDF

- SKLearn adds normalization as an extra step to our earlier TF-IDF formula by dividing each TF-IDF vector with its Euclidean Norms

TF-IDF vector of a document

$$v_{norm} = \frac{v}{\|v\|^2} = \frac{v}{\sqrt{(v_1)^2 + (v_2)^2 + \ldots + (v_n)^2}}$$

Euclidean distance of the TF-IDF vector

TF-IDF example

doc1: john cat
doc2: cat cat eat fish
doc3: eat big fish

Vocabulary → [big, cat, eat, fish, john]

Term Frequency (TF) = our BOW feature vectors

	big	cat	eat	fish	john
doc1	0	1	0	0	1
doc2	0	2	1	1	0
doc3	1	0	1	1	0

TF-IDF example

- To normalize our Term Frequency (TF) vectors, divide each count with the total number of counts for each document

	big	cat	eat	fish	john	Counts
doc1	0	1	0	0	1	2
doc2	0	2	1	1	0	4
doc3	1	0	1	1	0	3

TF

Normalization

	big	cat	eat	fish	john
doc1	0	1/2	0	0	1/2
doc2	0	2/4	1/4	1/4	0
doc3	1/3	0	1/3	1/3	0

Normalized TF

TF-IDF example

doc1: john cat
doc2: cat cat eat fish
doc3: eat big fish

Vocabulary → [big, cat, eat, fish, john]

Document Frequency (DF) for each term in our corpus

Vocab	Document Frequency (DF)
big	1
cat	2
eat	2
fish	2
john	1

TF-IDF example

- Computing IDF for our vocabulary

term corpus total no. of documents in corpus

$$\mathrm{IDF}(t, c) = \ln \frac{1 + n}{1 + \mathrm{df}(t, c)} + 1$$

natural log

document frequency (no. of time a term t appears in the corpus c at least once)

The IDF formula

Vocab	Document Frequency (DF)	Inverse Document Frequency (IDF)
big	1	$ln(\frac{1+3}{1+1}) + 1 =$ ln(4/2) + 1 = 1.693
cat	2	$ln(\frac{1+3}{1+2}) + 1 =$ ln(4/3) + 1 = 1.288
eat	2	ln(4/3) + 1 = 1.288
fish	2	ln(4/3) + 1 = 1.288
john	1	ln(4/2) + 1 = 1.693

Computed IDF for our corpus

TF-IDF example

- Computing TF-IDF of each word in each document

	big	cat	eat	fish	john
doc1	0	1/2	0	0	1/2
doc2	0	2/4	1/4	1/4	0
doc3	1/3	0	1/3	1/3	0

Normalized TF

×

Vocab	IDF
big	ln(4/2) + 1 = 1.693
cat	ln(4/3) + 1 = 1.287
eat	ln(4/3) + 1 = 1.287
fish	ln(4/3) + 1 = 1.287
john	ln(4/2) + 1 = 1.693

IDF

↓

	big	cat	eat	fish	john
doc1	0	1/2 * 1.287 = 0.644	0	0	1/2 * 1.693 = 0.846
doc2	0	2/4 * 1.287 = 0.644	1/4 * 1.287 = 0.322	1/4 * 1.287 = 0.322	0
doc3	1/3 * 1.693 = 0.564	0	1/3 * 1.287 = 0.429	1/3 * 1.287 = 0.429	0

TF-IDF

TF-IDF example

- Computing Euclidean distance for our TF-IDF feature vectors

TF-IDF vector of a document

$$v_{norm} = \frac{v}{\|v\|^2} = \frac{v}{\boxed{\sqrt{(v_1)^2 + (v_2)^2 + \ldots + (v_n)^2}}}$$

Euclidean distance of the TF-IDF vector

Normalized TF-IDF

	big	cat	eat	fish	john	Euclidean Distance
doc1	0	0.644	0	0	0.846	1.063
doc2	0	0.644	0.322	0.322	0	0.788
doc3	0.564	0	0.429	0.429	0	0.828

$$\sqrt{(0.644)^2 + (0.322)^2 + (0.322)^2}$$

Euclidean Distances of each feature vector

TF-IDF example

	big	cat	eat	fish	john	Euclidean Distance
doc1	0	0.644	0	0	0.846	1.063
doc2	0	0.644	0.322	0.322	0	0.788
doc3	0.564	0	0.429	0.429	0	0.828

TF-IDF

	big	cat	eat	fish	john
doc1	0	$\boxed{0.644/1.063}$ = 0.606	0	0	0.846/1.063 = 0.796
doc2	0	0.644/0.788 = 0.817	0.322/0.788 = 0.409	0.322/0.788 = 0.409	0
doc3	0.564/0.828 = 0.681	0	0.429/0.828 = 0.518	0.429/0.828 = 0.518	0

Normalized TF-IDF

TF-IDF example

- Our corpus has been transformed into numerics to be fed into learning models

doc1: john cat
doc2: cat cat eat fish
doc3: eat big fish

$\xrightarrow{\text{Vocabulary}}$ [big, cat, eat, fish, john]

$\downarrow$ TF-IDF

	big	cat	eat	fish	john
doc1	0	0.606	0	0	0.796
doc2	0	0.817	0.409	0.409	0
doc3	0.681	0	0.518	0.518	0

Normalized TF-IDF

TF-IDF (in code)

- With sklearn’s TF-IDF vectorizer, we can generate each document’s TF-IDF feature vectors via fit_transform(…)

```
tfidf = TfidfVectorizer()
feature_vectors = tfidf.fit_transform(docs_clean).toarray()
```

- Pandas can then be used to visualize the feature vectors by document

```
df = pd.DataFrame(data=feature_vectors,
                  index=['doc1', 'doc2', 'doc3'],
                  columns=tfidf.get_feature_names())

print(df)
```

	big	cat	eat	fish	john
doc1	0	0.606	0	0	0.796
doc2	0	0.817	0.409	0.409	0
doc3	0.681	0	0.518	0.518	0

TF-IDF (in code)

```
import string
import pandas as pd
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from sklearn.feature_extraction.text import CountVectorizer

# can be removed if resources reside in your system
nltk.download('stopwords')
nltk.download('wordnet')

lemmatizer = WordNetLemmatizer()
stop_words = stopwords.words('english')

docs = [
    'John has some cats.',
    'Cats, being cats, eat fish.',
    'I ate a big fish.'
]

# data cleansing
docs_clean = []
punc = str.maketrans('', '', string.punctuation)
for doc in docs:
    doc_no_punc = doc.translate(punc)
    words = doc_no_punc.lower().split()

    words = [lemmatizer.lemmatize(word, 'v')
                for word in words if word not in stop_words]

    docs_clean.append(' '.join(words))
```

```
# generate feature vectors using TF-IDF
tfidf = TfidfVectorizer()
feature_vectors = tfidf.fit_transform(docs_clean).toarray()

df = pd.DataFrame(data=feature_vectors,
                  index=['doc1', 'doc2', 'doc3'],
                  columns=tfidf.get_feature_names())

print(df)
```

Cosine Similarity

Cosine Similarity

- After TF-IDF transforms our documents into vectors, Cosine Similarity can be used to compare the similarity among them
- The Cosine Similarity between two vectors (or two documents) is a measure that calculates the cosine angle between them
- Two documents are considered similar to each other if the cosine angle between their feature vectors is small

Cosine Similarity

- The Dot Product of two vectors is given as

dot product

length of vectors

$$\vec{a} \bullet \vec{b} = \|\vec{a}\|\|\vec{b}\|cos\theta$$

vectors

angle between the two vectors

Cosine Similarity

- The Cosine Similarity of two feature vectors is simply

feature vectors

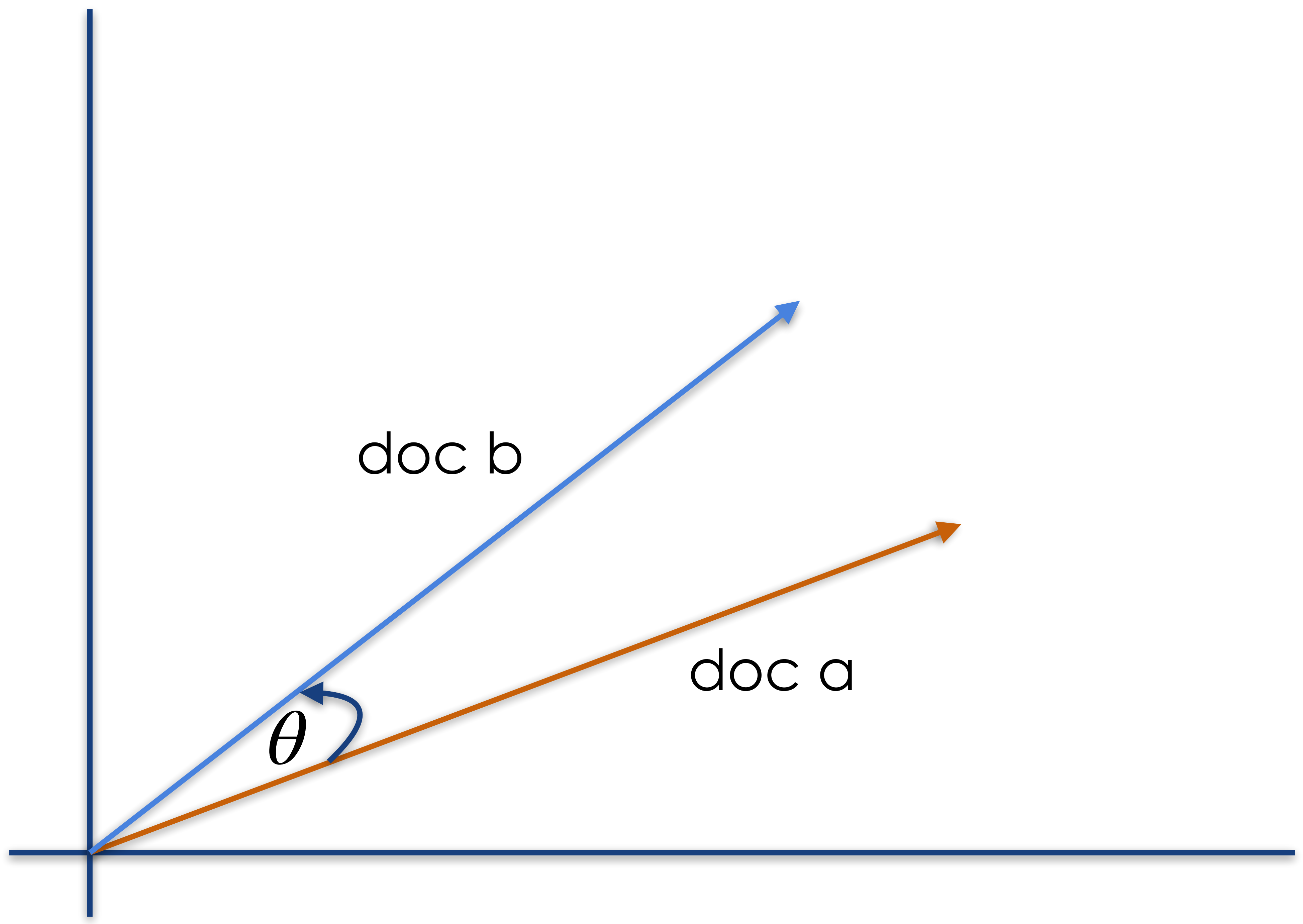
cosine similarity

$$\cos\theta = \frac{\vec{a} \bullet \vec{b}}{\|\vec{a}\|\|\vec{b}\|}$$

length of feature vectors

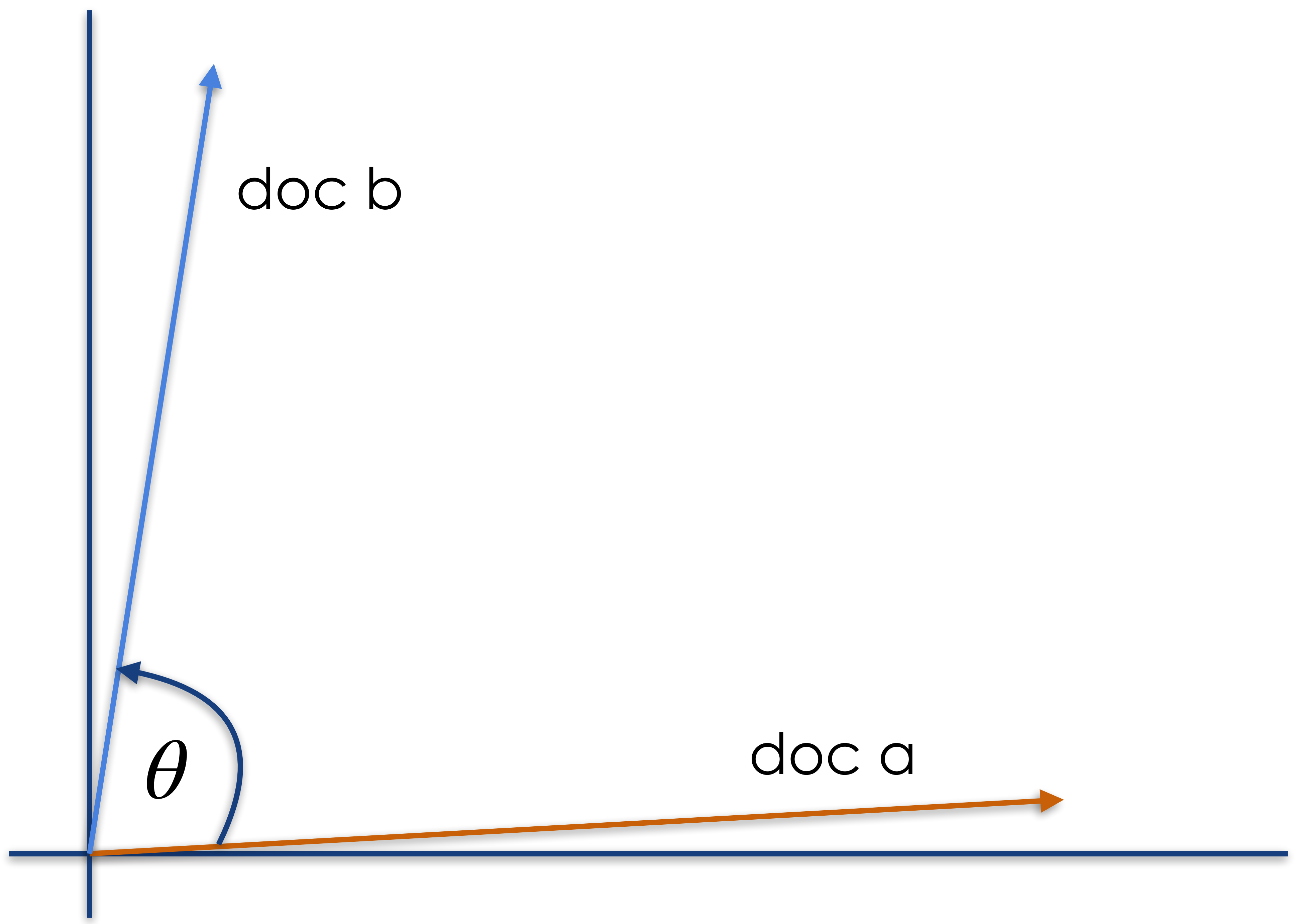
Cosine Similarity

- Two very similar documents yield a cosine similarity that is close to 1 (cos(0) = 1)



Cosine Similarity

- Likewise, two documents that are very distinct from each other yield a cosine similarity that is close to 0 (cos(90) = 0)



Cosine Similarity example

- Cosine Similarity can be for querying for relevant documents given a query string
- Let’s say our query string is “cats and fish”
- First, we treat “cats and fish” as a document and perform TF-IDF on it

Cosine Similarity example

- Computed Normalized TF-IDF vector for “cat fish”

	big	cat	eat	fish	john
TF	0	1	0	1	0
Normalized TF	0	1/2	0	1/2	0
IDF	1.692	1.288	1.288	1.288	1.693
TF * IDF	0	0.644	0	0.644	0
Normalized TF-IDF	0	0.707	0	0.707	0

Cosine Similarity example

- Treating the Normalized TF-IDF vector for the query string "cats and fish" as a document within our corpus

	big	cat	eat	fish	john
doc1	0	0.606	0	0	0.796
doc2	0	0.817	0.409	0.409	0
doc3	0.681	0	0.518	0.518	0
query	0	0.707	0	0.707	0

Normalized TF-IDF

Cosine Similarity example

- How similar is the query string compared to doc2?
- Vector Multiplication of $\overrightarrow{Query} \bullet \overrightarrow{Doc2}$ = $\boxed{(0.707 * 0.817)}$ + (0.707 * 0.409) = 0.866
- Length Multiplication of $\|\overrightarrow{Query}\|\|\overrightarrow{Doc2}\|$ = 1 * 1 = 1
- Cosine Similarity = $cos\theta = \dfrac{\overrightarrow{Query} \bullet \overrightarrow{Doc2}}{\|\overrightarrow{Query}\|\|\overrightarrow{Doc2}\|}$ = 0.866 / 1 = 0.866
- Query and Doc2 have similar words, and that similarity is being reflected with a cosine similarity of 0.866, which is close to 1

	big	cat	eat	fish	john
query	0	0.707	0	0.707	0
doc2	0	0.817	0.409	0.409	0

Cosine Similarity (in code)

- Use sklearn for Cosine Similarity calculation
- Restructure code for reusability

```
import string
import pandas as pd
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

def preprocess(docs):
    cleansed = []
    punc = str.maketrans('', '', string.punctuation)

    for doc in docs:
        doc_no_punc = doc.translate(punc)
        words = doc_no_punc.lower().split()

        words = [lemmatizer.lemmatize(word, 'v')
                    for word in words if word not in stop_words]

        cleansed.append(' '.join(words))

    return cleansed
```

Cosine Similarity (in code)

- Fit the TF-IDF model using the terms found in our corpus
- Used the fitted model to generate feature vectors for the documents in the corpus and the query string

```
lemmatizer = WordNetLemmatizer()
stop_words = stopwords.words('english')

docs = [
    'John has some cats.',
    'Cats, being cats, eat fish.',
    'I ate a big fish'
]

query = ['cats and fish']

docs_clean = preprocess(docs)
query_clean = preprocess(query)

tfidf = TfidfVectorizer()
tfidf.fit(docs_clean)

fv_corpus = tfidf.transform(docs_clean).toarray()
fv_query = tfidf.transform(query_clean).toarray()
```

Cosine Similarity (in code)

- Viewing our query string's TF-IDF feature vector in a tabular format using Pandas

```
fv = pd.DataFrame(data=fv_query,
                  index=['query string'],
                  columns=tfidf.get_feature_names())

print(fv, '\n')
```

	big	cat	eat	fish	john
query	0	0.707	0	0.707	0

Cosine Similarity (in code)

- Performing a Cosine Similarity for the query string against all documents in our corpus

```
similarity = cosine_similarity(fv_query, fv_corpus)

cs = pd.DataFrame(data=similarity,
                  index=['cosine similarity'],
                  columns=['doc1', 'doc2', 'doc3'])

print(cs)
```

	doc1	doc2	doc3
Cosine Similarity	0.428	0.866	0.366

Cosine Similarity (in code)

<u>Corpus (stop-words removed & lemmatized)</u>
- doc1: john cat
- doc2: cat cat eat fish
- doc3: eat big fish

<u>Query String (stop-words removed & lemmatized)</u>
- cat fish

- The Cosine Similarity values indicate that doc2 has the highest similarity with our query string, followed by doc1 and doc3

	doc1	doc2	doc3
Cosine Similarity	0.428	0.866	0.366

Application - Search Engine

- Rank documents by relevance given a query string
- Query String - “cats and fish”
- Returned Results
  - doc2 (rank: 0.866)
  - doc1 (rank: 0.428)
  - doc3 (rank: 0.366)

Entire code

```python
import string
import pandas as pd
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

def preprocess(docs):
    cleansed = []
    punc = str.maketrans('', '', string.punctuation)

    for doc in docs:
        doc_no_punc = doc.translate(punc)
        words = doc_no_punc.lower().split()

        words = [lemmatizer.lemmatize(word, 'v')
                    for word in words if word not in stop_words]

        cleansed.append(' '.join(words))

    return cleansed


lemmatizer = WordNetLemmatizer()
stop_words = stopwords.words('english')
```

```python
docs = [
    'John has some cats.',
    'Cats, being cats, eat fish.',
    'I ate a big fish'
]

query = ['cats and fish']

docs_clean = preprocess(docs)
query_clean = preprocess(query)

# compute normalized TF-IDF
tfidf = TfidfVectorizer()
tfidf.fit(docs_clean)

fv_corpus = tfidf.transform(docs_clean).toarray()
fv_query = tfidf.transform(query_clean).toarray()

fv = pd.DataFrame(data=fv_query,
            index=['query string'],
            columns=tfidf.get_feature_names())

print(fv, '\n')

#compute cosine similarity
similarity = cosine_similarity(fv_query, fv_corpus)

cs = pd.DataFrame(data=similarity,
            index=['cosine similarity'],
            columns=['doc1', 'doc2', 'doc3'])

print(cs)
```

The End