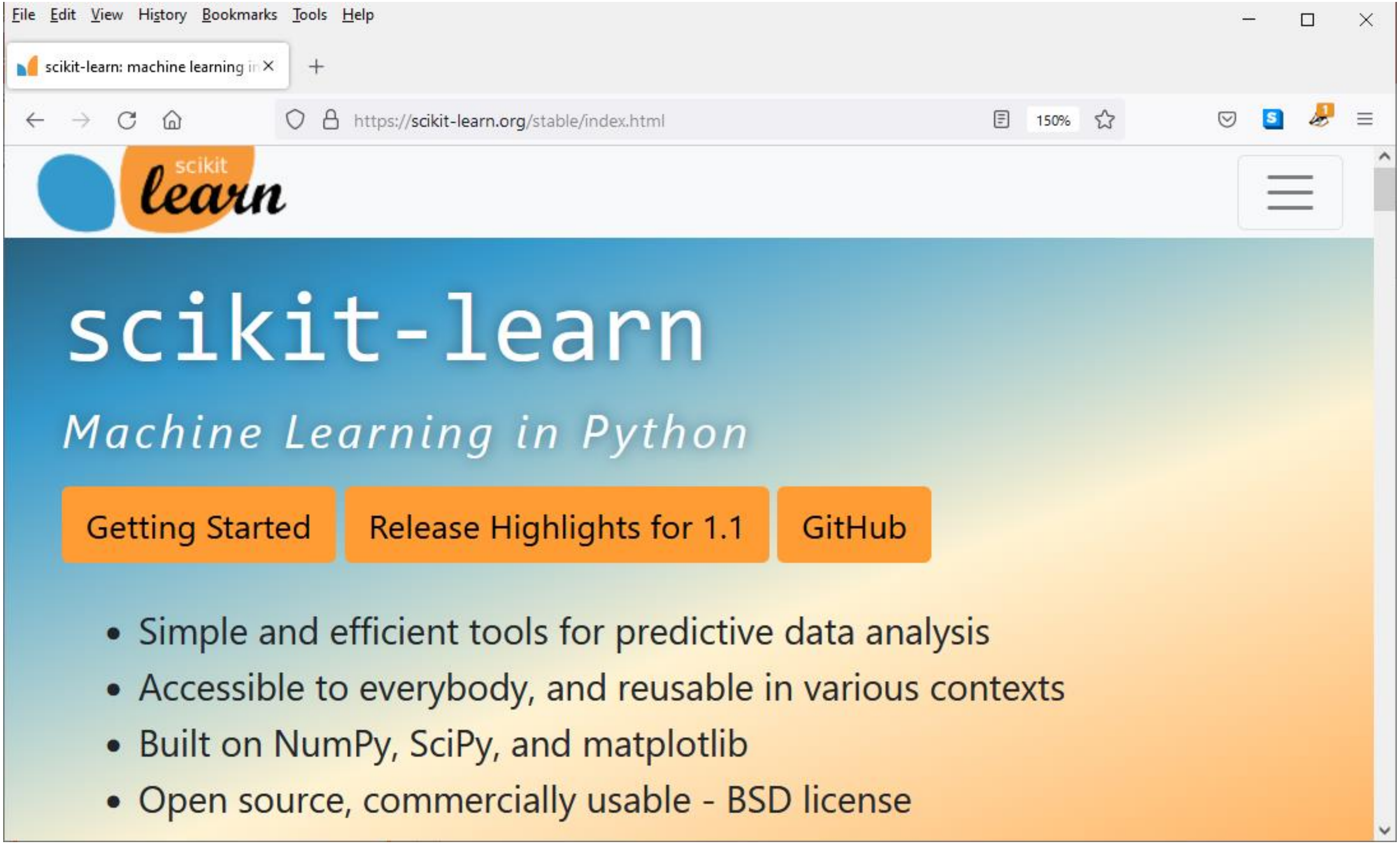
The Vector Space Model

CS5154/6054

Yizong Cheng

9/8/2022


File Edit View History Bookmarks Tools Help
scikit-learn: machine learning in
https://scikit-learn.org/stable/index.html
150%
scikit learn
scikit-learn
Machine Learning in Python
Getting Started
Release Highlights for 1.1
GitHub
• Simple and efficient tools for predictive data analysis
• Accessible to everybody, and reusable in various contexts
• Built on NumPy, SciPy, and matplotlib
• Open source, commercially usable - BSD license

sklearn.preprocessing.MultiLabelBinarizer

class `sklearn.preprocessing.MultiLabelBinarizer`(*, *classes=None*, *sparse_output=False*) ¶ [source]

Transform between iterable of iterables and a multilabel format.

Although a list of sets or tuples is a very intuitive format for multilabel data, it is unwieldy to process. This transformer converts between this intuitive format and the supported multilabel format: a (samples x classes) binary matrix indicating the presence of a class label.

Parameters::

classes : *array-like of shape (n_classes,), default=None*
Indicates an ordering for the class labels. All entries should be unique (cannot contain duplicate classes).

sparse_output : *bool, default=False*
Set to True if output binary array is desired in CSR sparse format.

Attributes::

classes_ : *ndarray of shape (n_classes,)*
A copy of the `classes` parameter when provided. Otherwise it corresponds to the sorted set of classes found when fitting.

File Edit View History Bookmarks Tools Help
sklearn.preprocessing.MultiLab × Python Examples of sklearn.preproc ×
https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MultiLabelBinarizer.h 150%

Methods

`fit`(y)	Fit the label sets binarizer, storing classes_.
`fit_transform`(y)	Fit the label sets binarizer and transform the given label sets.
`get_params`([deep])	Get parameters for this estimator.
`inverse_transform`(yt)	Transform the given indicator matrix into label sets.
`set_params`(**params)	Set the parameters of this estimator.
`transform`(y)	Transform the given label sets.

fit(*y*) [source]

Fit the label sets binarizer, storing classes_.

Parameters:: **y : *iterable of iterables***

A set of labels (any orderable and hashable object) for each sample. If the `classes` parameter is set, `y` will not be iterated.

Returns:: **self : *object***

Fitted estimator.

Toggle Menu

iirexercise1-2.txt

breakthrough drug for schizophrenia
new schizophrenia drug
new approach for treatment of schizophrenia
new hopes for schizophrenia patients

```
import re
import numpy as np
from sklearn.preprocessing import MultiLabelBinarizer

f = open("iirexercise1-2.txt", "r")
docs = f.readlines()
f.close()
tokens = list(map(lambda s: re.findall('\w+', s), docs))
lb = MultiLabelBinarizer()
lb.fit(tokens)
print(lb.classes_)
```

```
['approach' 'breakthrough' 'drug' 'for' 'hopes' 'new' 'of' 'patients'
 'schizophrenia' 'treatment']
```

The Document-Term Incidence Matrix

```
f = open("iirexercise1-2.txt", "r")
docs = f.readlines()
f.close()
tokens = list(map(lambda s: re.findall('\w+', s), docs))
lb = MultiLabelBinarizer()
lb.fit(tokens)
vectors = lb.transform(tokens)
print(vectors)
```

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

From Sets to Vectors

- A set {‘new’, ‘schizophrenia’, ‘drug’} representing Doc2 is now
- A vector [0 0 1 0 0 1 0 0 1 0].
- In a ten-dimensional Euclidean space.
- Two vectors in the same space can have dot product (innerproduct).
  - Defined as sum of element products in all dimensions.
- Dot product = intersection.

Term-Document Matrix

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth	...
ANTHONY	1	1	0	0	0	1	
BRUTUS	1	1	0	1	0	0	
CAESAR	1	1	0	1	1	1	
CALPURNIA	0	1	0	0	0	0	
CLEOPATRA	1	0	0	0	0	0	
MERCY	1	0	1	1	1	1	
WORSER	1	0	1	1	1	0	
...							

Each document is represented as a binary vector $\in \{0,1\}^{|V|}$.

Binary Vector Representation of Documents

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth	...
ANTHONY	1	1	0	0	0	1	
BRUTUS	1	1	0	1	0	0	
CAESAR	1	1	0	1	1	1	
CALPURNIA	0	1	0	0	0	0	
CLEOPATRA	1	0	0	0	0	0	
MERCY	1	0	1	1	1	1	
WORSER	1	0	1	1	1	0	
...							

Each document is represented as a binary vector $\in \{0, 1\}^{|V|}$.

Set Intersection is Vector Dot Product

```
f = open("iirexercise1-2.txt", "r")
docs = f.readlines()
f.close()
tokens = list(map(lambda s: re.findall('\w+', s), docs))
lb = MultiLabelBinarizer()
lb.fit(tokens)
vectors = lb.transform(tokens)
print(vectors)

print(np.dot(vectors[0], vectors[1]))
```

```
[[0 1 1 1 0 0 0 0 1 0]
 [0 0 1 0 0 1 0 0 1 0]
 [1 0 0 1 0 1 1 0 1 1]
 [0 0 0 1 1 1 0 1 1 0]]
2
```

sklearn.metrics.pairwise.cosine_similarity

`sklearn.metrics.pairwise.cosine_similarity`(*X, Y=None, dense_output=True*) [source]

Compute cosine similarity between samples in X and Y.

Cosine similarity, or the cosine kernel, computes similarity as the normalized dot product of X and Y:

K(X, Y) = <X, Y> / (||X||*||Y||)

On L2-normalized data, this function is equivalent to linear_kernel.

Read more in the User Guide.

Parameters::	**X : *{ndarray, sparse matrix} of shape (n_samples_X, n_features)*** Input data. **Y : *{ndarray, sparse matrix} of shape (n_samples_Y, n_features), default=None*** Input data. If `None`, the output will be the pairwise similarities between all samples in `X`.

Toggle Menu

Cosine Similarity |A ∩ B|/√(|A||B|)

```
import re
import numpy as np
from sklearn.preprocessing import MultiLabelBinarizer
from sklearn.metrics.pairwise import cosine_similarity

f = open("iirexercise1-2.txt", "r")
docs = f.readlines()
f.close()
tokens = list(map(lambda s: re.findall('\w+', s), docs))
lb = MultiLabelBinarizer()
lb.fit(tokens)
vectors = lb.transform(tokens)
print(vectors)
print(cosine_similarity(vectors))
```

```
[[0 1 1 1 0 0 0 0 1 0]
 [0 0 1 0 0 1 0 0 1 0]
 [1 0 0 1 0 1 1 0 1 1]
 [0 0 0 1 1 1 0 1 1 0]]
[[1.         0.57735027 0.40824829 0.4472136 ]
 [0.57735027 1.         0.47140452 0.51639778]
 [0.40824829 0.47140452 1.         0.54772256]
 [0.4472136  0.51639778 0.54772256 1.        ]]
```

Cosine Similarity is Normalized Intersection

- Dice coefficient: Dice (A, B) = $|A \cap B|/((|A| + |B|)/2)$
  - Intersection size divided by the arithmetic mean of the sizes of the two sets
- Cosine similarity: $|A \cap B||A|^{-1/2}|B|^{-1/2}$
  - Intersection size divided by the geometric mean of the sizes of the two sets
  - $0.57735027 = (2)4^{-1/2}\ 3^{-1/2}$
- There are other means of two numbers.
  - Harmonic mean: $|A \cap B|(|A|^{-1} + |B|^{-1})$

sklearn.feature_extraction.text.CountVectorizer

class `sklearn.feature_extraction.text.`**`CountVectorizer`**(*, *input='content'*, *encoding='utf-8'*, *decode_error='strict'*, *strip_accents=None*, *lowercase=True*, *preprocessor=None*, *tokenizer=None*, *stop_words=None*, *token_pattern='(?u)\b\w\w+\b'*, *ngram_range=(1, 1)*, *analyzer='word'*, *max_df=1.0*, *min_df=1*, *max_features=None*, *vocabulary=None*, *binary=False*, *dtype=<class 'numpy.int64'>*) [source]

Convert a collection of text documents to a matrix of token counts.

This implementation produces a sparse representation of the counts using scipy.sparse.csr_matrix.

If you do not provide an a-priori dictionary and you do not use an analyzer that does some kind of feature selection then the number of features will be equal to the vocabulary size found by analyzing the data.

n the User Guide

Parameters::

input : *{'filename', 'file', 'content'}, default='content'*

- If `'filename'`, the sequence passed as an argument to fit is expected to be a list of filenames that need reading to fetch the raw content to analyze.
- If `'file'`, the sequence items must have a 'read' method (file-like object) that is called to fetch the bytes in memory.
- If `'content'`, the input is expected to be a sequence of items that can be of type string or byte.

encoding : *str, default='utf-8'*

If bytes or files are given to analyze, this encoding is used to decode.

decode_error : *{'strict', 'ignore', 'replace'}, default='strict'*

Instruction on what to do if a byte sequence is given to analyze that contains characters not of the given `encoding`. By default, it is 'strict', meaning that a UnicodeDecodeError will be raised. Other values are 'ignore' and 'replace'.

strip_accents : *{'ascii', 'unicode'}, default=None*

Remove accents and perform other character normalization during the

token_pattern : *str, default=r"(?u)\b\w\w+\b"*

Regular expression denoting what constitutes a "token", only used if `analyzer == 'word'`. The default regexp select tokens of 2 or more alphanumeric characters (punctuation is completely ignored and always treated as a token separator).

If there is a capturing group in token_pattern then the captured group content, not the entire match, becomes the token. At most one capturing group is permitted.

ngram_range : *tuple (min_n, max_n), default=(1, 1)*

The lower and upper boundary of the range of n-values for different word n-grams or char n-grams to be extracted. All values of n such such that min_n <= n <= max_n will be used. For example an `ngram_range` of `(1, 1)` means only unigrams, `(1, 2)` means unigrams and bigrams, and `(2, 2)` means only bigrams. Only applies if `analyzer` is not callable.

File Edit View History Bookmarks Tools Help

sklearn.feature_extraction.text.C

https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVecto 150%

max_df : *float in range [0.0, 1.0] or int, default=1.0*

When building the vocabulary ignore terms that have a document frequency strictly higher than the given threshold (corpus-specific stop words). If float, the parameter represents a proportion of documents, integer absolute counts. This parameter is ignored if vocabulary is not None.

min_df : *float in range [0.0, 1.0] or int, default=1*

When building the vocabulary ignore terms that have a document frequency strictly lower than the given threshold. This value is also called cut-off in the literature. If float, the parameter represents a proportion of documents, integer absolute counts. This parameter is ignored if vocabulary is not None.

Methods

build_analyzer()	Return a callable to process input data.
build_preprocessor()	Return a function to preprocess the text before tokenization.
build_tokenizer()	Return a function that splits a string into a sequence of tokens.
decode(doc)	Decode the input into a string of unicode symbols.
fit(raw_documents[, y])	Learn a vocabulary dictionary of all tokens in the raw documents.
fit_transform(raw_documents[, y])	Learn the vocabulary dictionary and return document-term matrix.
get_feature_names()	DEPRECATED: get_feature_names is deprecated in 1.0 and will be removed in 1.2.
get_feature_names_out([input_features])	Get output feature names for transformation.
get_params([deep])	Get parameters for this estimator.
get_stop_words()	Build or fetch the effective stop words list.
inverse_transform(X)	Return terms per document with nonzero entries in X.
set_params(**params)	Set the parameters of this estimator.
transform(raw_documents)	Transform documents to document-term matrix.

Toggle Menu

```
# IR6B.py CS5154/6054 cheng 2022
# Usage: python IR6B.py

import re
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics.pairwise import cosine_similarity

f = open("gutprotocol.txt", "r", encoding="utf8")
docs = f.readlines()
f.close()
cv = CountVectorizer()
cv.fit(docs)
print(cv.get_feature_names())
vectors = cv.transform(docs)
print(vectors)
```

Command Prompt

```
C:\classes\6054>python IR6B.py
['000', '000g', '10', '100', '101', '102', '103', '104', '105', '106', '107', '108', '109', '11', '110', '111', '112', '
113', '114', '115', '116', '117', '118', '119', '12', '120', '13', '14', '15', '150', '16', '17', '18', '19', '20', '200
', '21', '22', '23', '24', '25', '26', '27', '28', '29', '30', '300', '31', '32', '33', '34', '35', '36', '37', '38', '3
9', '40', '400g', '41', '42', '43', '44', '45', '450', '46', '47', '48', '488', '49', '50', '500', '500g', '51', '52', '
53', '54', '55', '56', '57', '58', '59', '60', '600', '61', '62', '63', '64', '65', '650', '66', '67', '68', '69', '70',
 '71', '72', '73', '74', '75', '76', '77', '78', '79', '80', '81', '82', '83', '84', '85', '86', '87', '88', '89', '90',
 '900', '91', '92', '93', '94', '95', '96', '97', '98', '99', 'acceptable', 'accommodate', 'according', 'achieve', 'add'
, 'adding', 'addition', 'additional', 'adhered', 'after', 'again', 'against', 'agar', 'air', 'alcohol', 'aliquot', 'aliq
uots', 'all', 'allow', 'allowing', 'already', 'also', 'alternatively', 'aluminum', 'amount', 'amounts', 'an', 'anaerobic
', 'anaerobically', 'and', 'another', 'antifade', 'any', 'apical', 'appropriate', 'are', 'as', 'aside', 'aspirate', 'asp
irating', 'aspiration', 'aspirator', 'assess', 'at', 'avoid', 'away', 'back', 'background', 'bacteria', 'bacterial', 'ba
g', 'base', 'basolateral', 'bath', 'be', 'because', 'become', 'been', 'before', 'behavior', 'bend', 'bent', 'biosafety',
 'block', 'blockaid', 'blue', 'both', 'bottom', 'break', 'bright', 'bring', 'bucket', 'buffer', 'but', 'by', 'cabinet',
'calculate', 'calculated', 'calculations', 'can', 'cap', 'capacity', 'carefully', 'case', 'cases', 'cause', 'cell', 'cel
ls', 'center', 'centrifugation', 'centrifuge', 'cfu', 'chamber', 'change', 'check', 'co2', 'coated', 'coating', 'cocultu
re', 'cold', 'collagen', 'collect', 'collected', 'collection', 'colon', 'colonies', 'colony', 'comparing', 'completely',
 'component', 'concentration', 'confluency', 'conical', 'connect', 'connecting', 'connects', 'containing', 'contains', '
content', 'continue', 'cool', 'cooling', 'corresponding', 'count', 'counted', 'countess', 'counting', 'cover', 'covered'
, 'coverslip', 'critical', 'crs', 'cryotube', 'cryovial', 'culture', 'cup', 'currently', 'curve', 'cut', 'cycles', 'dapi
', 'data', 'day', 'density', 'depending', 'depends', 'described', 'desired', 'determine', 'diameter', 'did', 'differenti
ation', 'dilute', 'diluted', 'diluting', 'dilution', 'dilutions', 'discard', 'dish', 'disinfect', 'disinfectant', 'dislo
dge', 'dispense', 'disposable', 'disrupting', 'dissociate', 'dissociation', 'dissolved', 'disturbing', 'divide', 'dividi
ng', 'donor', 'down', 'dpbs', 'drawing', 'dried', 'drop', 'droplet', 'droplets', 'dry', 'during', 'each', 'easily', 'edg
e', 'edta', 'eight', 'end', 'endohm', 'ends', 'ensure', 'epithelial', 'equivalent', 'estimate', 'ethanol', 'evaluate', '
evaluating', 'evaporate', 'every', 'example', 'excessive', 'eye', 'factors', 'field', 'fig', 'fill', 'finally', 'firm',
'firmly', 'first', 'fit', 'fixative', 'flat', 'flip', 'foil', 'following', 'follows', 'for', 'forceps', 'formation', 'fo
rming', 'formula', 'fragment', 'fragmentation', 'fragmented', 'fragments', 'fresh', 'friday', 'fridays', 'fridge', 'from
', 'full', 'fully', 'further', 'gelation', 'generally', 'generate', 'generated', 'gentle', 'gently', 'get', 'getting', '
```

Command Prompt

```
'trypan', 'trypsin', 'tube', 'tubes', 'turn', 'turned', 'tweezer', 'tweezers', 'two', 'typical', 'typically', 'under', '
unit', 'unstable', 'until', 'up', 'use', 'used', 'using', 'vacuum', 'value', 'values', 'varies', 'vary', 'via', 'viabili
ty', 'vol', 'volume', 'wait', 'warm', 'was', 'wash', 'washing', 'water', 'we', 'well', 'wells', 'when', 'which', 'whole'
, 'wide', 'will', 'wire', 'wires', 'wish', 'with', 'withdraw', 'workstation', 'ycfa', 'you', 'yourself', 'µg', 'µl', 'µm
']
  (0, 28)       1
  (0, 125)      1
  (0, 153)      1
  (0, 214)      1
  (0, 236)      1
  (0, 244)      1
  (0, 296)      1
  (0, 447)      1
  (0, 607)      1
  (0, 668)      1
  (0, 688)      2
  (0, 701)      1
  (0, 713)      1
  (1, 93)       1
  (1, 153)      1
  (1, 186)      1
  (1, 199)      1
  (1, 284)      1
  (1, 317)      1
  (1, 529)      1
  (1, 624)      1
  (1, 639)      1
  (1, 716)      1
  (1, 724)      2
  (2, 195)      1
```

Command Prompt

```
  (2, 195)      1
  :      :
  (121, 668)    1
  (121, 689)    1
  (122, 25)     1
  (122, 28)     1
  (122, 142)    1
  (122, 198)    1
  (122, 251)    2
  (122, 253)    1
  (122, 303)    1
  (122, 307)    1
  (122, 365)    1
  (122, 385)    1
  (122, 442)    1
  (122, 445)    1
  (122, 483)    1
  (122, 488)    1
  (122, 519)    1
  (122, 543)    1
  (122, 595)    1
  (122, 599)    1
  (122, 600)    2
  (122, 621)    2
  (122, 668)    4
  (122, 688)    3
  (122, 726)    1

C:\classes\6054>
```

Count Matrix with Term Frequencies (TFs)

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth	...
ANTHONY	157	73	0	0	0	1	
BRUTUS	4	157	0	2	0	0	
CAESAR	232	227	0	2	1	0	
CALPURNIA	0	10	0	0	0	0	
CLEOPATRA	57	0	0	0	0	0	
MERCY	2	0	3	8	5	8	
WORSER	2	0	1	1	1	5	
...							

Each document is now represented as a count vector $\in \mathbb{N}^{|V|}$.

Count Vector Representation of Documents

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth	...
ANTHONY	157	73	0	0	0	1	
BRUTUS	4	157	0	2	0	0	
CAESAR	232	227	0	2	1	0	
CALPURNIA	0	10	0	0	0	0	
CLEOPATRA	57	0	0	0	0	0	
MERCY	2	0	3	8	5	8	
WORSER	2	0	1	1	1	5	
...							

Each document is now represented as a count vector $\in \mathbb{N}^{|V|}$.

To compensate for the effect of document length, the standard way of quantifying the similarity between two documents d_1 and d_2 is to compute the *cosine similarity* of their vector representations $\vec{V}(d_1)$ and $\vec{V}(d_2)$

COSINE SIMILARITY

$$\text{sim}(d_1, d_2) = \frac{\vec{V}(d_1) \cdot \vec{V}(d_2)}{|\vec{V}(d_1)||\vec{V}(d_2)|}, \tag{6.10}$$

DOT PRODUCT

EUCLIDEAN LENGTH

where the numerator represents the *dot product* (also known as the *inner product*) of the vectors $\vec{V}(d_1)$ and $\vec{V}(d_2)$, while the denominator is the product of their *Euclidean lengths*. The dot product $\vec{x} \cdot \vec{y}$ of two vectors is defined as $\sum_{i=1}^{M} x_i y_i$. Let $\vec{V}(d)$ denote the document vector for d, with M components $\vec{V}_1(d) \ldots \vec{V}_M(d)$. The Euclidean length of d is defined to be $\sqrt{\sum_{i=1}^{M} \vec{V}_i^2(d)}$.

LENGTH-NORMALIZATION

The effect of the denominator of Equation (6.10) is thus to *length-normalize* the vectors $\vec{V}(d_1)$ and $\vec{V}(d_2)$ to unit vectors $\vec{v}(d_1) = \vec{V}(d_1)/|\vec{V}(d_1)|$ and $\vec{v}(d_2) = \vec{V}(d_2)/|\vec{V}(d_2)|$. We can then rewrite (6.10) as

$$\text{sim}(d_1, d_2) = \vec{v}(d_1) \cdot \vec{v}(d_2). \tag{6.11}$$

Example 6.2: Consider the documents in Figure 6.9. We now apply Euclidean normalization to the tf values from the table, for each of the three documents in the table. The quantity $\sqrt{\sum_{i=1}^{M} \vec{V}_i^2(d)}$ has the values 30.56, 46.84 and 41.30 respectively for Doc1, Doc2 and Doc3. The resulting Euclidean normalized tf values for these documents are shown in Figure 6.11.

	Doc1	Doc2	Doc3
car	27	4	24
auto	3	33	0
insurance	0	33	29
best	14	0	17

► **Figure 6.9** Table of tf values for Exercise 6.10.

	Doc1	Doc2	Doc3
car	0.88	0.09	0.58
auto	0.10	0.71	0
insurance	0	0.71	0.70
best	0.46	0	0.41

► **Figure 6.11** Euclidean normalized tf values for documents in Figure 6.9.

File Edit View History Bookmarks Tools Help

sklearn.feature_extraction.text.T

https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfTransfor 150%

scikit learn Install User Guide API Examples Community More Go

sklearn.feature_extraction.text.TfidfTransformer

class `sklearn.feature_extraction.text.`**`TfidfTransformer`**(**, norm='l2', use_idf=True, smooth_idf=True, sublinear_tf=False*) [source]

Transform a count matrix to a normalized tf or tf-idf representation.

Tf means term-frequency while tf-idf means term-frequency times inverse document-frequency. This is a common term weighting scheme in information retrieval, that has also found good use in document classification.

The goal of using tf-idf instead of the raw frequencies of occurrence of a token in a given document is to scale down the impact of tokens that occur very frequently in a given corpus and that are hence ...ess informative than features that occur in a small fraction of the training corpus.

Toggle Menu

File Edit View History Bookmarks Tools Help

sklearn.feature_extraction.text.T

https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectoriz 150%

scikit learn Install User Guide API Examples Community More Go

sklearn.feature_extraction.text.TfidfVectorizer

class `sklearn.feature_extraction.text.`**`TfidfVectorizer`**(**, input='content', encoding='utf-8', decode_error='strict', strip_accents=None, lowercase=True, preprocessor=None, tokenizer=None, analyzer='word', stop_words=None, token_pattern='(?u)\b\w\w+\b', ngram_range=(1, 1), max_df=1.0, min_df=1, max_features=None, vocabulary=None, binary=False, dtype=<class 'numpy.float64'>, norm='l2', use_idf=True, smooth_idf=True, sublinear_tf=False*) [source]

Convert a collection of raw documents to a matrix of TF-IDF features.

Equivalent to `CountVectorizer` followed by `TfidfTransformer`.

Toggle Menu n the User Guide.

```
# IR6C.py CS5154/6054 cheng 2022
# Usage: python IR6C.py

import re
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

f = open("gutprotocol.txt", "r", encoding="utf8")
docs = f.readlines()
f.close()
tv = TfidfVectorizer(max_df=0.4, min_df=3)
tv.fit(docs)
print(tv.get_feature_names())
vectors = tv.transform(docs)
print(vectors)
```

Command Prompt

```
C:\classes\6054>python IR6C.py
['000', '000g', '10', '100', '102', '105', '12', '15', '20', '200', '24', '37', '45', '46', '50', '500', '60', '600', '7
0', '84', '91', '96', '99', 'add', 'additional', 'after', 'again', 'agar', 'aliquot', 'aliquots', 'all', 'allow', 'alter
natively', 'aluminum', 'an', 'anaerobic', 'apical', 'appropriate', 'are', 'as', 'aspirate', 'aspirator', 'at', 'avoid',
'back', 'bacteria', 'bacterial', 'base', 'basolateral', 'bath', 'be', 'before', 'biosafety', 'block', 'both', 'bottom',
'bring', 'bucket', 'by', 'cabinet', 'calculate', 'can', 'cap', 'carefully', 'cell', 'cells', 'centrifugation', 'centrifu
ge', 'cfu', 'chamber', 'check', 'co2', 'coating', 'coculture', 'collagen', 'colon', 'completely', 'conical', 'containing
', 'contains', 'continue', 'cover', 'crs', 'cryovial', 'culture', 'curve', 'day', 'density', 'described', 'desired', 'de
termine', 'dilution', 'dish', 'disinfectant', 'disposable', 'dissociation', 'disturbing', 'down', 'dpbs', 'droplet', 'dr
oplets', 'during', 'each', 'edta', 'end', 'endohm', 'ethanol', 'every', 'example', 'fill', 'first', 'from', 'full', 'gen
tly', 'glass', 'growth', 'have', 'ice', 'if', 'in', 'incubate', 'incubation', 'incubator', 'insert', 'inside', 'inspect'
, 'into', 'inverted', 'is', 'it', 'keep', 'least', 'let', 'make', 'matrigel', 'measurement', 'medium', 'membrane', 'micr
oscope', 'min', 'mix', 'ml', 'monolayer', 'monolayers', 'needed', 'new', 'next', 'number', 'obtain', 'obtained', 'od600'
, 'ohmmeter', 'on', 'once', 'one', 'onto', 'or', 'organoid', 'organoids', 'original', 'outer', 'overnight', 'passaging',
 'pasteur', 'pbs', 'pellet', 'per', 'pipette', 'pipetting', 'place', 'plate', 'plates', 'preempt', 'preparation', 'prepa
re', 'prepared', 'prereduced', 'prewarmed', 'proceed', 'put', 'ratio', 'ready', 'record', 'remove', 'repeat', 'required'
, 'residual', 'respectively', 'resuspend', 'rinse', 'room', 'seed', 'seeding', 'set', 'should', 'side', 'sides', 'single
', 'size', 'small', 'solution', 'spray', 'staining', 'standard', 'step', 'steps', 'sterile', 'sterilize', 'supernatant',
 'supplementary', 'surface', 'suspension', 'table', 'take', 'teer', 'temperature', 'that', 'thaw', 'then', 'there', 'the
y', 'this', 'thoroughly', 'time', 'times', 'tip', 'tissue', 'top', 'transfer', 'transwell', 'transwells', 'trypsin', 'tu
be', 'tubes', 'tweezers', 'under', 'until', 'up', 'use', 'using', 'vacuum', 'values', 'via', 'vol', 'volume', 'washing',
 'water', 'we', 'well', 'wells', 'when', 'which', 'will', 'wire', 'with', 'workstation', 'ycfa', 'µl']
  (0, 237)      0.3651628007763991
  (0, 233)      0.41104229322535957
  (0, 193)      0.3932449402525009
  (0, 141)      0.21306422155313334
  (0, 97)       0.43282449285216673
  (0, 77)       0.3255832481767333
  (0, 67)       0.3651628007763991
```

Weight Matrix

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth	...
ANTHONY	5.25	3.18	0.0	0.0	0.0	0.35	
BRUTUS	1.21	6.10	0.0	1.0	0.0	0.0	
CAESAR	8.59	2.54	0.0	1.51	0.25	0.0	
CALPURNIA	0.0	1.54	0.0	0.0	0.0	0.0	
CLEOPATRA	2.85	0.0	0.0	0.0	0.0	0.0	
MERCY	1.51	0.0	1.90	0.12	5.25	0.88	
WORSER	1.37	0.0	0.11	4.15	0.25	1.95	
...							

Each document is now represented as a real-valued vector of tf-idf weights $\in \mathbb{R}^{|V|}$.

Weight Matrix

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth	...
ANTHONY	5.25	3.18	0.0	0.0	0.0	0.35	
BRUTUS	1.21	6.10	0.0	1.0	0.0	0.0	
CAESAR	8.59	2.54	0.0	1.51	0.25	0.0	
CALPURNIA	0.0	1.54	0.0	0.0	0.0	0.0	
CLEOPATRA	2.85	0.0	0.0	0.0	0.0	0.0	
MERCY	1.51	0.0	1.90	0.12	5.25	0.88	
WORSER	1.37	0.0	0.11	4.15	0.25	1.95	
...							

Each document is now represented as a real-valued vector of tf-idf weights $\in \mathbb{R}^{|V|}$.

COSINESCORE(q)

float $Scores[N] = 0$
Initialize $Length[N]$
for each query term t
do calculate $\mathrm{w}_{t,q}$ and fetch postings list for t
    for each pair(d, $\mathrm{tf}_{t,d}$) in postings list
    do $Scores[d] \mathrel{+}= \mathrm{wf}_{t,d} \times \mathrm{w}_{t,q}$
Read the array $Length[d]$
for each d
do $Scores[d] = Scores[d] / Length[d]$
return Top K components of $Scores[]$

▶ **Figure 6.14** The basic algorithm for computing vector space scores.