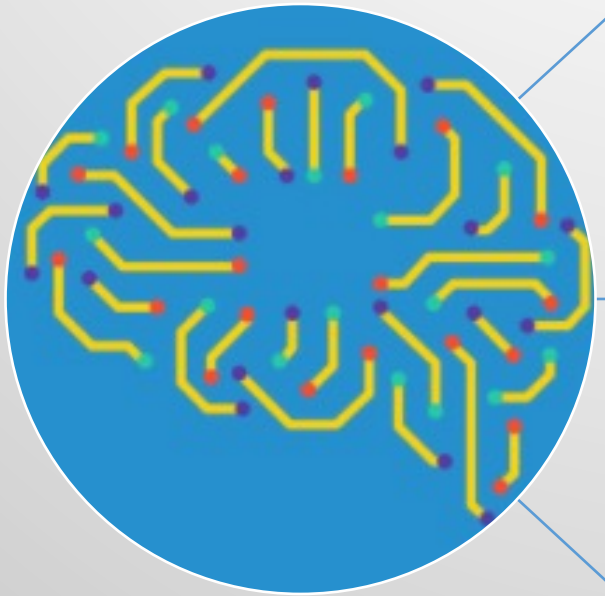
Natural Language Processing

Dr. Karen Mazidi

Part Five:
Machine Learning



Topics

- Text to Data

Quizzes

- Not on this material

Homework

- Homework: not yet

Converting text to numeric data

- Machine learning algorithms expect numeric data
- Convert text to:
  - Count matrices with sklearn CountVectorizer()
  - Tf-idf matrices with sklearn TfidfVectorizer()
  - Both are bag of words approaches
- Corpus:

```
corpus = ['Mary had a little lamb.',
      'The lamb followed Mary to school one day.',
      'The lamb was white.',
      'Mary should not bring a lamb to school.',
      'Mary is a little rebel.']
```

sklearn CountVectorizer

- The function:
  - Tokenizes the text
  - Assigns a unique integer id to each word in the corpus
  - Counts word frequencies

sklearn CountVectorizer

Code 19.1.1 — CountVectorizer. Initial processing.

```
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer()
vectorizer.fit(corpus)
```

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

- analyzer=’word’: use word ngrams, not character
- binary=False: counts instead of 1/0
- ngram_range=(1,1): only unigrams
- Max and min df: ignore terms with df above/below
- Max features: only the top n terms

sklearn CountVectorizer

- Notice that stop words were not removed
- ‘a’ was removed by regex

```
print(vectorizer.vocabulary_)
{'mary': 7, 'had': 3, 'little': 6, 'lamb': 5, 'the': 13, 'followed': 2,
'to': 14, 'school': 11, 'one': 9, 'day': 1, 'was': 15, 'white': 16,
'should': 12, 'not': 8, 'bring': 0, 'is': 4, 'rebel': 10}
```

sklearn CountVectorizer

- Produces a sparse document-term matrix
- One row for each doc; one col for each term

Code 19.1.2 — CountVectorizer. Create a sparse document-term matrix

```
corpus_counts = vectorizer.fit_transform(corpus)
print(corpus_counts.toarray())
print('names:', vectorizer.get_feature_names())
```

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

```
names: ['bring', 'day', 'followed',
'lamb', 'little', 'mary', 'one', 'rebel',
'school', 'white']
```

sklearn TfidfVectorizer

- Similar arguments to CountVectorizer

Code 19.2.1 — TfidfVectorizer. Initial processing

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer()
vectorizer.fit(corpus)
```

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

sklearn TfidfVectorizer

- Similar to CountVectorizer but also computes idf for each term so that tfidf can be calculated

Code 19.2.2 — TfidfVectorizer. Initial processing

```
# look at terms and idf
print('terms:', vectorizer.vocabulary_)
print('idf:', vectorizer.idf_)
```

```
terms: {'mary': 7, 'had': 3, 'little': 6, 'lamb': 5, 'the': 13, 'followed': 2,
'to': 14, 'school': 11, 'one': 9, 'day': 1, 'was': 15, 'white': 16,
'should': 12, 'not': 8, 'bring': 0, 'is': 4, 'rebel': 10}
idf: [2.09861229 2.09861229 2.09861229 2.09861229 2.09861229 1.18232156
 1.69314718 1.18232156 2.09861229 2.09861229 2.09861229 1.69314718
 2.09861229 1.69314718 1.69314718 2.09861229 2.09861229]
```

sklearn TfidfVectorizer

Code 19.2.3 — TfidfVectorizer. Create sparse tfidf matrix

```
# look at the docs
corpus_tfidf = vectorizer.transform(corpus)
print(corpus_tfidf.toarray())
```

```
[[0.         0.         0.         0.6614376  0.         0.3726424
  0.53364369 0.3726424  0.         0.         0.         0.
  0.         0.         0.         0.         0.        ]
 [0.         0.42304773 0.42304773 0.         0.         0.23833771
  0.         0.23833771 0.         0.42304773 0.         0.34131224
  0.         0.34131224 0.34131224 0.         0.         ]
 [0.         0.         0.         0.         0.         0.32700044
  0.         0.         0.         0.         0.         0.
  0.         0.46828197 0.         0.58042343 0.58042343]
 [0.45007472 0.         0.         0.         0.         0.25356425
  0.         0.25356425 0.45007472 0.         0.         0.36311745
  0.45007472 0.         0.36311745 0.         0.        ]
 [0.         0.         0.         0.         0.58042343 0.
  0.46828197 0.32700044 0.         0.         0.58042343 0.
  0.         0.         0.         0.         0.        ]]
```

Isolate the test data

Code 19.3.1 — Processing Train and Test Data. A general approach.

```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split

# read the data
df = pd.read_csv('mydata.csv')

X = df.text     # features
y = df.labels  # targets

# train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y,
        test_size=0.2, train_size=0.8, random_state=1234)
```

Isolate the test data

- fit_transform training data
- Only transform test data

```
# vectorize

X_train = vectorizer.fit_transform(X_train)  # fit and transform
X_test = vectorizer.transform(X_test)         # transform only
```

Add more features

- Adding n-gram features restores some word order to these bag of words approaches
- Max df: feature should occur in at least 2 documents, but not more than 50%

Code 19.4.1 — TFidfVectorizer. Including bigrams

```
vectorizer = TfidfVectorizer(min_df=2, max_df=0.5,
                             ngram_range=(1, 2))
```

sklearn text processing

- Both CountVectorizer and TfidfVectorizer transform raw text to sparse document-term matrices, ready for input to machine learning algorithms
- See the sklearn documentation for full feature list

- Knowing the best parameters can be trial and error
- Example: sarcasm detection performed worse by 5% when stop words were removed

Code Examples

- See text2data notebook in Part 5 Chp 19

Essential points to note

- Raw text needs to be converted from character data to numeric data
- We will learn many ways to do this
- Preprocessing may be done first

Next class

Machine Learning with Text