12 - Text Feature Extraction

Data Science Practicum 2021/22, Lesson 12

Marko Tkalčič

Univerza na Primorskem

Table of Contents

Features in Machine Learning

Document	Label
It was the best of times	C
it was the worst of times	S
it was the age of wisdom	C
it was the age of foolishness	?

Features in Machine Learning

Document	Label
It was the best of times	C
it was the worst of times	S
it was the age of wisdom	C
it was the age of foolishness	?

Document	f1	f2	f3	Label
It was the best of times	1	2	1	C
it was the worst of times	3	3	3	S
it was the age of wisdom	4	1	2	C
it was the age of foolishness	2	2	1	?

Bag-of-Words

- Method for extracting features from text

It was the best of times,
it was the worst of times,
it was the age of wisdom,
it was the age of foolishness,

- Step 1: vocabulary (unique words):
  - it, was, the, best, of, times, worst, age, wisdom, foolishness

Bag-of-Words

- Method for extracting features from text

It was the best of times,
it was the worst of times,
it was the age of wisdom,
it was the age of foolishness,

- Step 1: vocabulary (unique words):
  - it, was, the, best, of, times, worst, age, wisdom, foolishness

- Step 2: document vectors

Document	it	was	the	best	of	times	worst	age	wisdom	foolishness	Label
It was the best of times	1	1	1	1	1	1	0	0	0	0	C
it was the worst of times	1	1	1	0	1	1	1	0	0	0	S
it was the age of wisdom	1	1	1	0	1	0	0	1	1	0	C
it was the age of foolishness	1	1	1	0	1	0	0	1	0	1	S

Bag of Words

```
"it was the worst of times" -> [1, 1, 1, 0, 1, 1, 1, 0, 0, 0]
"it was the age of wisdom" -> [1, 1, 1, 0, 1, 0, 0, 1, 1, 0]
"it was the age of foolishness" -> [1, 1, 1, 0, 1, 0, 0, 1, 0, 1]
```

Bag of Words

```
"it was the worst of times" -> [1, 1, 1, 0, 1, 1, 1, 0, 0, 0]
"it was the age of wisdom" -> [1, 1, 1, 0, 1, 0, 0, 1, 1, 0]
"it was the age of foolishness" -> [1, 1, 1, 0, 1, 0, 0, 1, 0, 1]
```

- Instead of single words we can use N-grams (bi-grams, tri-grams etc.)
  - it was
  - was the
  - the best
  - best of
  - of times

Bag of Words

```
"it was the worst of times" -> [1, 1, 1, 0, 1, 1, 1, 0, 0, 0]
"it was the age of wisdom" -> [1, 1, 1, 0, 1, 0, 0, 1, 1, 0]
"it was the age of foolishness" -> [1, 1, 1, 0, 1, 0, 0, 1, 0, 1]
```

- Instead of single words we can use N-grams (bi-grams, tri-grams etc.)
  - it was
  - was the
  - the best
  - best of
  - of times

- step 3: scoring words
  - instead of a binary vector we can give each N-gram a score
    - Counts: Count the number of times each word appears in a document.
    - Frequencies: Calculate the frequency that each word appears in a document out of all the words in the document.

TF-IDF

- Highly frequent words tend to dominate but might not have information value
  - Rarer words, domain speciofic words may contain more info for the classifier
- Solution?

TF-IDF

- Highly frequent words tend to dominate but might not have information value
  - Rarer words, domain speciofic words may contain more info for the classifier
- Solution?
- TF-Term Frequency: is a scoring of the frequency of the word in the current document.
  - $f_{t,d}$ is the raw count of term t in document d

$$tf(t, d) = f_{t,d}$$

- IDF-Inverse Document Frequency: is a scoring of how rare the word is across documents.
  - N = number of documents in the corpus $N = |D|$
  - the logarithmically scaled inverse fraction of the documents d in corpus D that contain the word t
  - denominator = number of documents where the term t appears

$$idf(t, D) = \log \frac{N}{|\{d \in D : t \in d\}|}$$

TF-IDF

- Highly frequent words tend to dominate but might not have information value
  - Rarer words, domain speciofic words may contain more info for the classifier
- Solution?
- TF-Term Frequency: is a scoring of the frequency of the word in the current document.
  - $f_{t,d}$ is the raw count of term t in document d

$$tf(t,d) = f_{t,d}$$

- IDF-Inverse Document Frequency: is a scoring of how rare the word is across documents.
  - N = number of documents in the corpus $N = |D|$
  - the logarithmically scaled inverse fraction of the documents d in corpus D that contain the word t
  - denominator = number of documents where the term t appears

$$idf(t,D) = \log \frac{N}{|\{d \in D : t \in d\}|}$$

$$tfidf(t,d,D) = tf(t,d) \cdot idf(t,D)$$

TFIDF with sklearn

```
import pandas as pd
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import CountVectorizer

# 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"
]
```

TFIDF with sklearn

```
import pandas as pd
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import CountVectorizer

# 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"
]
```

- First, create a CountVectorizer to count the number of words (term frequency), limit your vocabulary size, apply stop words
- shape should be 5 rows (5 docs) per 16 columns (16 unique words)

```
#instantiate CountVectorizer()
cv=CountVectorizer()

# this steps generates word counts for the words in your docs
word_count_vector=cv.fit_transform(docs)

word_count_vector.shape
```

```
(5, 16)
```

- You can specify a custom stop word list, enforce minimum word count, etc.

TFIDF with sklearn

- Now we are going to compute the IDF values by calling tfidf_transformer.fit(word_count_vector) on the word counts we computed earlier.

```
tfidf_transformer=TfidfTransformer(smooth_idf=True,use_idf=True)
tfidf_transformer.fit(word_count_vector)
```

TFIDF with sklearn

- Now we are going to compute the IDF values by calling tfidf_transformer.fit(word_count_vector) on the word counts we computed earlier.

```
tfidf_transformer=TfidfTransformer(smooth_idf=True,use_idf=True)
tfidf_transformer.fit(word_count_vector)
```

- let's visualize it

```
# print idf values
df_idf = pd.DataFrame(tfidf_transformer.idf_, index=cv.get_feature_names(),
    columns=["idf_weights"])

# sort ascending
df_idf.sort_values(by=['idf_weights'])
```

	idf_weights
mouse	1.000000
the	1.000000
cat	1.693147
house	1.693147
ate	2.098612
away	2.098612
end	2.098612
finally	2.098612
from	2.098612
had	2.098612
little	2.098612
of	2.098612
ran	2.098612
saw	2.098612
story	2.098612
tiny	2.098612

TFIDF

- Once you have the IDF values, you can now compute the tf-idf scores for any document or set of documents.

```
# count matrix
count_vector=cv.transform(docs)
```

- We could have actually used word_count_vector from above.
- However, in practice, you may be computing tf-idf scores on a set of new unseen documents.
- When you do that, you will first have to do cv.transform(your_new_docs) to generate the matrix of word counts.

TFIDF

- Once you have the IDF values, you can now compute the tf-idf scores for any document or set of documents.

```
# count matrix
count_vector=cv.transform(docs)
```

- We could have actually used word_count_vector from above.
- However, in practice, you may be computing tf-idf scores on a set of new unseen documents.
- When you do that, you will first have to do cv.transform(your_new_docs) to generate the matrix of word counts.

- finally compute the tf-idf values

```
# tf-idf scores
tf_idf_vector=tfidf_transformer.transform(count_vector)
```

TFIDF

- let's visualize

```
feature_names = cv.get_feature_names()

#get tfidf vector for first document
first_document_vector=tf_idf_vector[0]

#print the scores
df = pd.DataFrame(first_document_vector.T.todense(), index=feature_names, columns=["tfidf"])
df.sort_values(by=["tfidf"],ascending=False)
```

	tfidf
had	0.493562
little	0.493562
tiny	0.493562
house	0.398203
mouse	0.235185
the	0.235185
ate	0.000000
away	0.000000
cat	0.000000
end	0.000000
finally	0.000000
from	0.000000
of	0.000000
ran	0.000000
saw	0.000000
story	0.000000

Exercise

- File mrtambourineman.txt = multiline text, utf8
- Treat each line as a document
- Generate a dataframe that contains the following columns
  - content_text: value of the line
  - *word*: TFIDF of the *word*
- Hints:
  - `feature_names = cv.get_feature_names()`
  - `tfidf_dense = tf_idf_vector[i].T.todense().tolist()`

Exercise

- File mrtambourineman.txt = multiline text, utf8
- Treat each line as a document
- Generate a dataframe that contains the following columns
  - content_text: value of the line
  - *word*: TFIDF of the *word*
- Hints:
  - `feature_names = cv.get_feature_names()`
  - `tfidf_dense = tf_idf_vector[i].T.todense().tolist()`

```
# imports
import pandas as pd
import numpy as np
import pandas as pd
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import CountVectorizer

# load file and split lines into list
with open("mrtambourineman.txt", encoding="utf8") as file:
    data = file.read()
docs = data.split("\n")
```

Exercise

```
# compute TFIDF

#instantiate CountVectorizer()
cv=CountVectorizer()

# this steps generates word counts for the words in your docs
word_count_vector=cv.fit_transform(docs)

tfidf_transformer=TfidfTransformer(smooth_idf=True,use_idf=True)
tfidf_transformer.fit(word_count_vector)

# count matrix
count_vector=cv.transform(docs)

# tf-idf scores
tf_idf_vector=tfidf_transformer.transform(count_vector)
```

Exercise

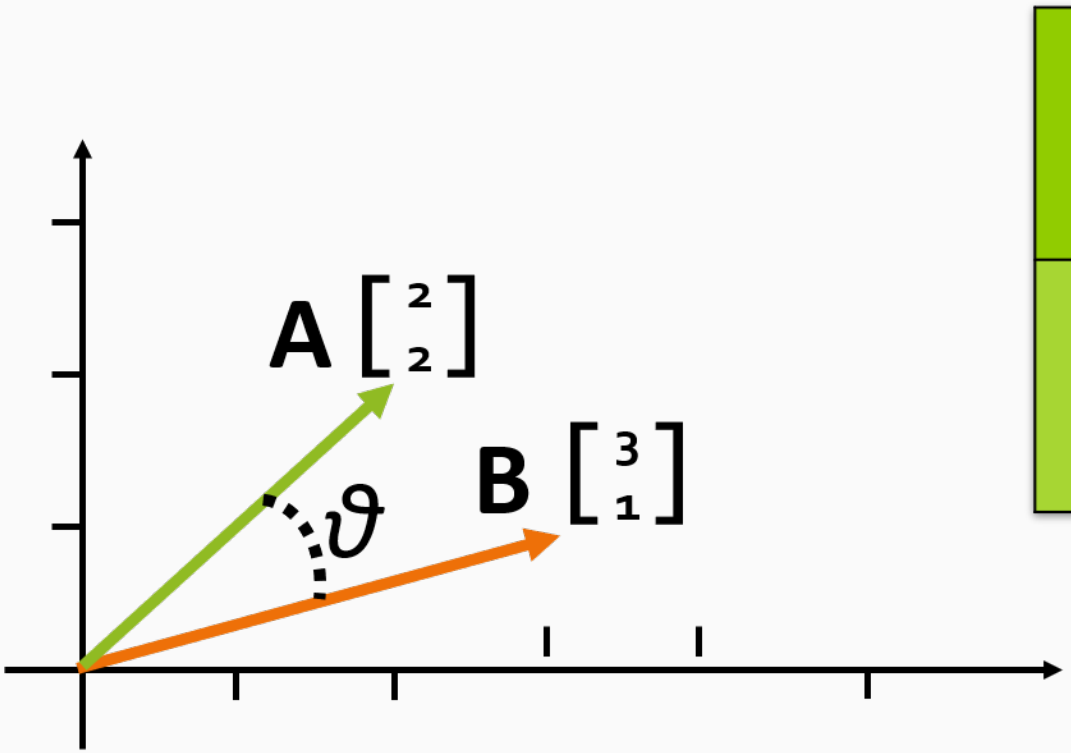
```
# create column names
feature_names = cv.get_feature_names()
cols = ["content_text"]
cols.extend(feature_names)

# create dataframe
df_out = pd.DataFrame(columns=cols)

# iterate through the lines
i = 0
for s in docs:
    row = [s]
    tfidf_dense = tf_idf_vector[i].T.todense().tolist()
    for ti in tfidf_dense:
        row.append(ti[0])
    df_out.loc[len(df_out)] = row
    i += 1

df_out
```

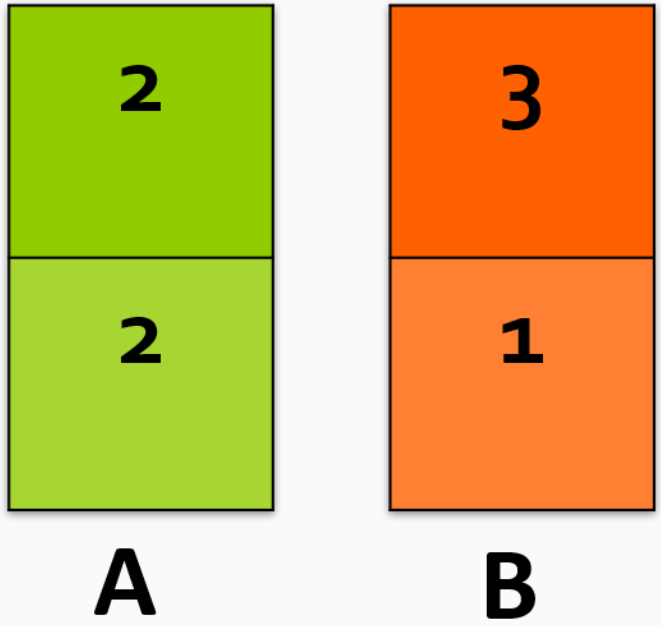
Out[20]:

	content_text	about	across	aimed	all	amazes	ancient	and	any	anyone	...	waves	waving	way	weariness
0	Hey! Mr. Tambourine Man, play a song for me	0.000000	0.000000	0.00000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00000	...	0.000000	0.000000	0.000000	0.000000
1	I'm not sleepy and there is no place I'm going to	0.000000	0.000000	0.00000	0.000000	0.000000	0.000000	0.302437	0.000000	0.00000	...	0.000000	0.000000	0.000000	0.000000
2	Hey! Mr. Tambourine Man, play a song for me	0.000000	0.000000	0.00000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00000	...	0.000000	0.000000	0.000000	0.000000
3	In the jingle jangle morning I'll come followi...	0.000000	0.000000	0.00000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00000	...	0.000000	0.000000	0.000000	0.000000

Table of Contents

Cosine Similarity





$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}},$$

Cosine Similarity

```
from scipy import spatial

v1 = [0,0,1]
v2 = [3,2,1]
s = spatial.distance.cosine(v1,v2)
print(s)
```

```
0.7327387580875756
```

- Other similarities:
  - Pearson's correlation
  - Spearman's correlation
  - Kendall's Tau
  - Cosine similarity
  - Jaccard similarity
  - Manhattan
  - Euclidian

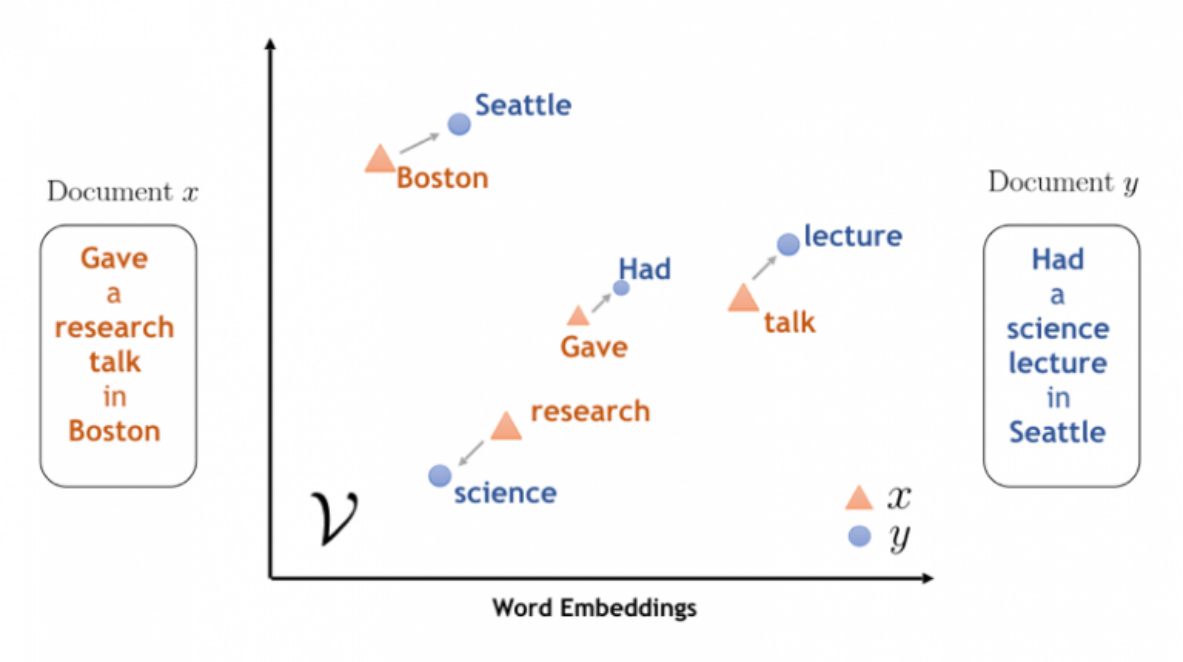
Exercise

- for the dataframe from the previous exercise, calculate pairwise cosine similarities between the lines

Table of Contents

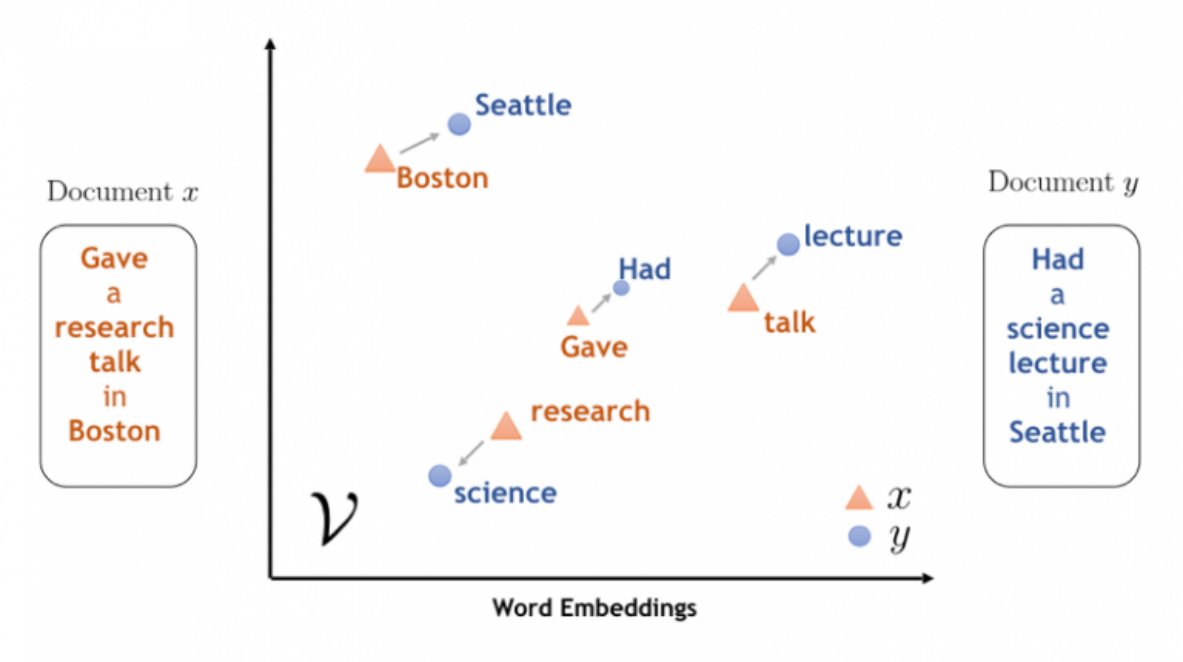
Word Embeddings

- Are a by-product of deep neural networks
- When a DNN is trained on a large corpus of data
  - text units (words) get their unique vectors in low dimensions (several hundreds)
  - each document is represented by an aggregation (sum) of the words' vectors
- Word2vec, fasttext, Bert
- Gensim package in Python



Word Embeddings

- Are a by-product of deep neural networks
- When a DNN is trained on a large corpus of data
  - text units (words) get their unique vectors in low dimensions (several hundreds)
  - each document is represented by an aggregation (sum) of the words' vectors
- Word2vec, fasttext, Bert
- Gensim package in Python



https://colab.research.google.com/drive/1zuq1I_FudtB2W4OSOWff8ODqfqK8d9-G#scrollTo=edQSGcAIh7oh

Table of Contents

References

Part of the material has been taken from the following sources. The usage of the referenced copyrighted work is in line with fair use since it is for nonprofit educational purposes.

- https://machinelearningmastery.com/gentle-introduction-bag-words-model/
- https://en.wikipedia.org/wiki/Tf%E2%80%93idf
- https://kavita-ganesan.com/tfidftransformer-tfidfvectorizer-usage-differences/
- https://www.ibm.com/blogs/research/2018/11/word-movers-embedding/