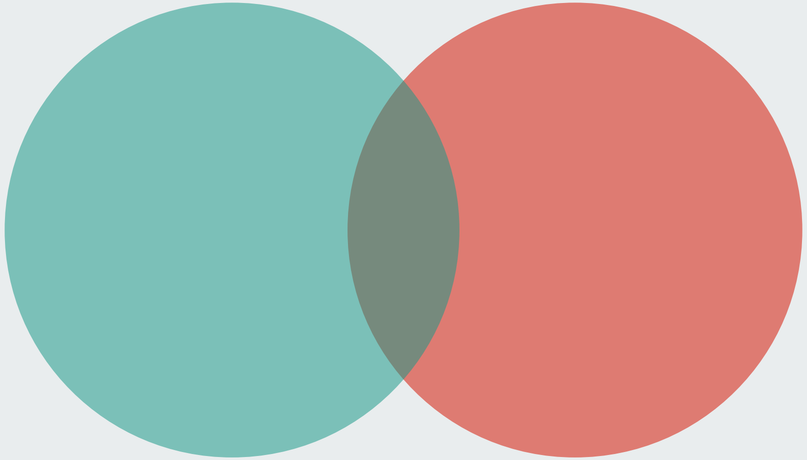
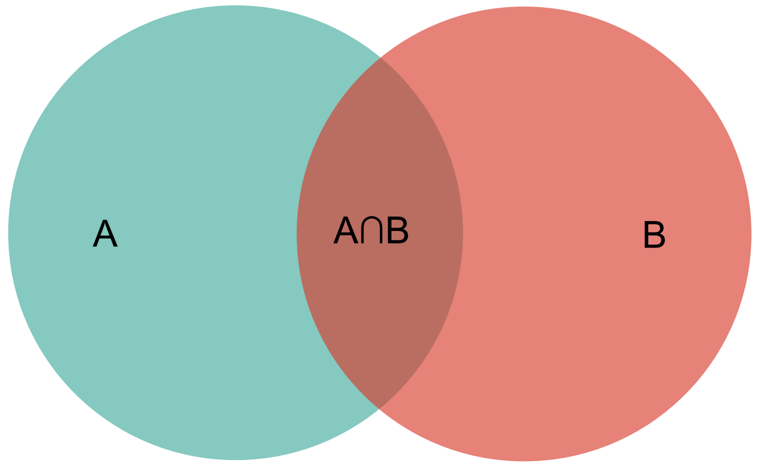
Jaccard Similarity & Minhashing



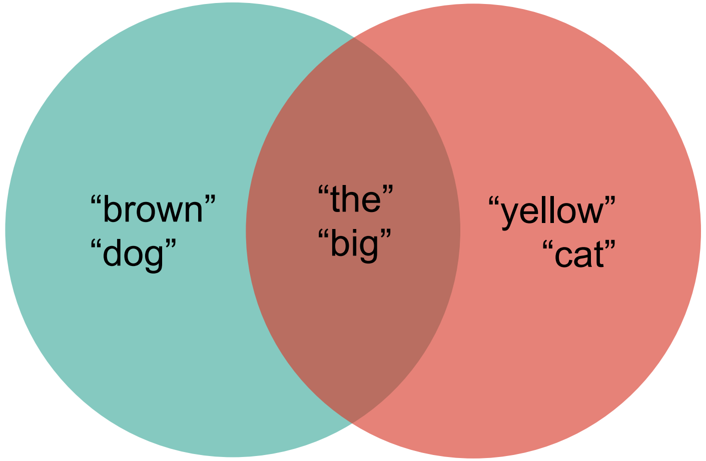
“Jaccard similarity is a statistic for comparing the similarity and diversity of sample sets”



$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}.$$

Jaccard similarity text example

J (“The big brown dog” , “The big yellow cat”) =



$$= \frac{2}{(4 + 4 - 2)} = 0.33$$

Matrix Representation

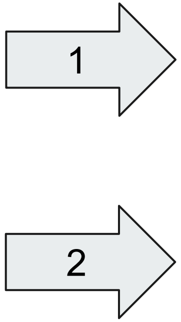
	“The big brown dog”	**“The big yellow cat”**
the	1	1
big	1	1
brown	1	0
dog	1	0
yellow	0	1
cat	0	1

Two types of rows

$$\frac{\text{Type 1}}{\text{Type 1 and Type 2}} = \frac{2}{6} = 0.33$$

It’s the jaccard similarity!

It’s also the P(type 1)



	“The big brown dog”	**“The big yellow cat”**
the	1	1
big	1	1
brown	1	0
dog	1	0
yellow	0	1
cat	0	1

Use sampling to estimate probabilities

1. Shuffle the rows
2. Find first row where word is in document (skip over zeros)
3. Check if first row is the same for document 1 and document 2

```
for permutation in permutations:
     first(d1)==first(d2)
>>> [ 0 1 0 0 1 1 0 0 ]
```

```
P(type 1) ≈ 3/8 = 0.375
```

		d1	d2
Permutation	**Vocab**	**“The big brown dog”**	**“The big yellow cat”**
3	the	1	1
5	big	1	1
2	brown	1	0
6	dog	1	0
1	yellow	0	1
4	cat	0	1

The “min” part of minhashing

1. Encode vocabulary
2. Represent documents using encoding
3. Check if minimum code from each document is the same

```
for permutation in permutations:
     min(d1)==min(d2)
>>> [ 0 1 0 0 1 1 0 0 ]
```

`P(type 1) ≈ 3/8 = 0.375`

Vocab	**Code**
the	4
big	2
brown	3
dog	6
yellow	5
cat	1

$$\left.\right\} \quad \begin{array}{l} d1 = [4,2,3,6] \\ \\ d2 = [4,2,5,1] \end{array}$$

The “hash” part of minhashing

We can use a hash instead of row numbers to encode our vocabulary

If we avoid row numbers, we don’t need to know the total number of words in our vocabulary!

```
for hash in hashes:
     min(d1)==min(d2)
>>> [ ...
```

d1 = [hash(“the”), hash(“big”), hash(“brown”), hash(“dog”)]
    = [4567, 2113, 6571, 6439]

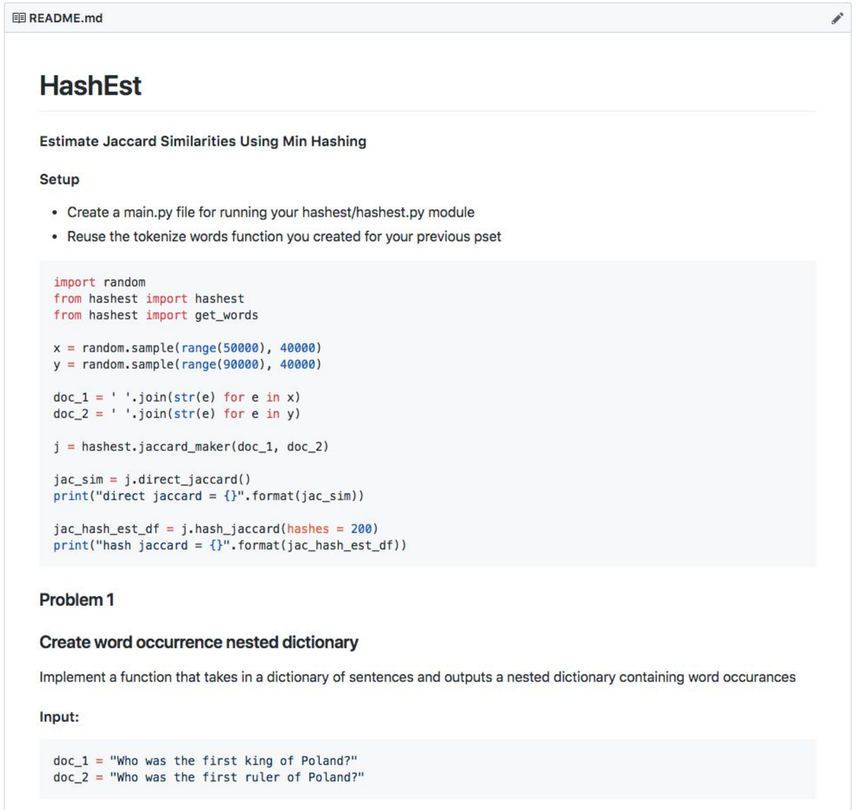
d2 = [hash(“the”), hash(“big”), hash(“yellow”), hash(“cat”)]
    = [4567, 2113, 5665, 1345]

Vocab	Hash
yellow	5665
cat	1345

HashEst Problem Set

1. Create word occurrence “matrix”
    - String tokenization
    - Dictionary comprehension
    - `itertools.chain`
2. Directly calculate jaccard similarity
    - `numpy.intersect1d`
3. Calculate jaccard using minhashing
    - Hashing and salt
    - `map`
    - `functools.partial`

git@github.com:avianap/HashEst.git


README.md

HashEst

Estimate Jaccard Similarities Using Min Hashing

Setup

- Create a main.py file for running your hashest/hashest.py module
- Reuse the tokenize words function you created for your previous pset

```
import random
from hashest import hashest
from hashest import get_words

x = random.sample(range(50000), 40000)
y = random.sample(range(90000), 40000)

doc_1 = ' '.join(str(e) for e in x)
doc_2 = ' '.join(str(e) for e in y)

j = hashest.jaccard_maker(doc_1, doc_2)

jac_sim = j.direct_jaccard()
print("direct jaccard = {}".format(jac_sim))

jac_hash_est_df = j.hash_jaccard(hashes = 200)
print("hash jaccard = {}".format(jac_hash_est_df))
```

Problem 1

Create word occurrence nested dictionary

Implement a function that takes in a dictionary of sentences and outputs a nested dictionary containing word occurances

Input:

```
doc_1 = "Who was the first king of Poland?"
doc_2 = "Who was the first ruler of Poland?"
```

Reference

https://medium.com/engineering-brainly/locality-sensitive-hashing-explained-304eb39291e4

https://en.wikipedia.org/wiki/Jaccard_index

https://docs.python.org/3/library/functools.html

https://docs.python.org/3/library/itertools.html

https://docs.scipy.org/doc/numpy/reference/generated/numpy.intersect1d.html