NLP in Journalism

Yian Shang

Introduction

Yian Shang

Data Engineer at Vox Media

Applications of NLP in Journalism — very broad

This workshop will focus on how to measure document similarity

Setting Up

Clone this repository: https://github.com/shangyian/nlp-journalism-workshop

Data:

TSV file that contains Vox.com articles published prior to March 2017

Load Data:

```
python main.py --load_from ./data/vox_Articles.tsv
```

Storage: SQLite and Redis: https://redis.io/download

Install Redis and start the server with `redis-server`

Cleaning Data

First and crucial component of NLP — dirty data will give bad output

Examine what's in the TSV file and what's in the database

Frequently entails:

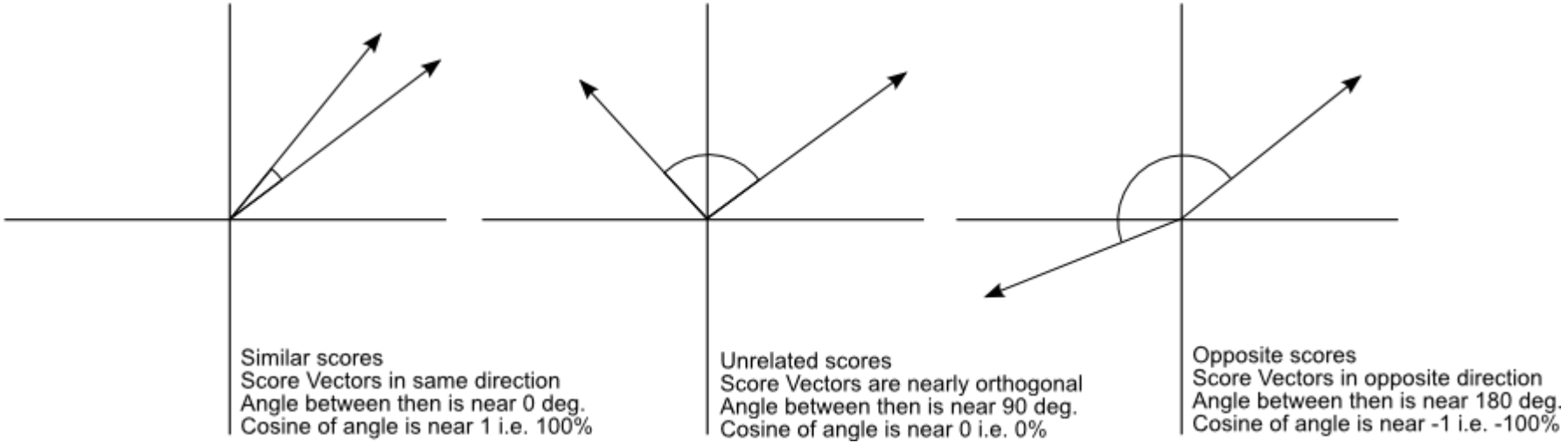
- Removing tags and extraneous characters
- Tokenizing
- Stemming

Vectorizing

Represent articles as vectors — done differently depending on model.

Term-Document Matrix

Cosine Similarity: Measurement of angle between vectors to denote similarity



TF-IDF

Term frequency: number of times each word appears in a document

Inverse document frequency: inverse of the number of documents the word occurs in

= log(total number of documents / number of documents containing term)

Building TF-IDF

Build **document-term matrix** — matrix describing tf-idf weighting of terms for documents

Using Scikit-learn

```
from sklearn.feature_extraction.text  import TfidfVectorizer

model = TfidfVectorizer().fit_transform(data)
```

Building TF-IDF

Build **document-term matrix** — matrix describing tf-idf weighting of terms for documents

Using Scikit-learn

```
from sklearn.feature_extraction.text  import TfidfVectorizer

model = TfidfVectorizer().fit_transform(data)

->

model = TfidfVectorizer(stop_words='english').fit_transform(map(lambda x:
x[1], data))
```

Taking A Closer Look

`fit_transform` yields a **document-term matrix** — weighted using tf-idf

Number of Documents:

```
print model.shape[0], " rows"
```

Terms:

```
print model.shape[1], " columns"
```

Cosine Similarity

Find cosine similarity between two articles using their vector representations

```
from sklearn.metrics.pairwise  import cosine_similarity

for i in range(model.shape[0]):
   similarity_values = cosine_similarity(model[i:i +  1], model)[0]
```

Calculates cosine similarity for article represented by the vector at `model[i]` against each of the other article vectors

Cosine Similarity

Let's save the cosine similarity values in Redis (remember, values between -1 and 1, higher scores = more similar):

```
for i in range(model.shape[0]):
    current_article_id = id2vector[i]
    similarity_values = cosine_similarity(model[i:i + 1], model)[0]
    print 'Adding similarity values for %s using model %s' %
(str(current_article_id), self.model_type)
    for j, value in zip(range(len(similarity_values)), similarity_values):
        redis_db.zadd('%s:%s' % (self.model_type, current_article_id), value,
id2vector[j])
```

Storing Similarity Scores in Redis

```
$ redis-cli

127.0.0.1:6379> zrangebyscore tfidf:5315709 0.5 1.0 WITHSCORES

 1) "5573777"

 2) "0.63665960967164681"

 3) "6613538"

 4) "0.70037538487511741"
```

Running Demo

Try running demo:

```
$ python complete/main.py --build_model tfidf
Adding similarity values for ... using model tfidf
```

When done, there should be a collection of similarity values stored in Redis for all articles in the Vox dataset

Viewing Similar Articles

Run Flask:

```
python complete/api.py
```

Access API:

http://0.0.0.0:8000/articles/5359389
http://0.0.0.0:8000/articles/

Under similar access point:

http://0.0.0.0:8000/similar/5359389/tfidf/10
http://0.0.0.0:8000/similar/<article_id>/<model_type>/<n>

Future

Go implement another model!

In repository, there is already one implemented based on the Word2Vec model called article2vec.

Load using:

```
python complete/main.py --build_model article2vec
```

View using API:

http://0.0.0.0:8000/similar/5359389/article2vec/10