◀ home (../)

Bag of Words (Bow) TF-IDF - Xử lý ngôn ngữ tự nhiên

Bag of Words là một thuật toán hỗ trợ xử lý ngôn ngữ tự nhiên và mục đích của BoW là phân loại text hay văn bản. Ý tưởng của BoW là phân tích và phân nhóm dựa theo "Bag of Words"(corpus). Với test data mới, tiến hành tìm ra số lần từng từ của test data xuất hiện trong "bag". Tuy nhiên BoW vẫn tồn tại khuyết điểm, nên TF-IDF là phương pháp khắc phục. Bạn có thể ứng dụng BoW + TF-IDF vào việc tìm kiếm, phân loại tài liệu, lọc mail spam xác định ý định của người dùng...

BoW hoạt động như thế nào ?

Hình thành vector

Ví dụ bạn có 2 câu như sau:

The quick brown fox jumps over the lazy dog and
Never jump over the lazy dog quickly

Từ 2 câu trên, tiến hành tạo từ điển chứa các từ xuất hiện trong từng câu.

```
{
    'brown': 0,
    'dog': 1,
    'fox': 2,
    'jump': 3,
    'jumps': 4,
    'lazy': 5,
    'never': 6,
    'over': 7,
    'quick': 8,
    'quickly': 9,
    'the': 10,
}
```

Dựa vào từ điển vừa tạo, tiến hành tạo vector lưu trữ số lần xuất hiện của từ trong từ điển ứng với mỗi câu.

Và do từ điển đang có 10 từ nên mỗi vector sẽ có 10 phần tử như sau:

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

Như các bạn thấy, ví dụ câu 1, "the" xuất hiện 2 lần nên phần tử thứ 10 sẽ mang giá trị 2.

Tương tự "brown", "dog", "fox", "jumps", "lazy", "over", "quick" xuất hiện 1 lần. Và vector chỉ mang giá trị 0 khi từ đó không xuất hiện trong câu như "jump", "over", "quickly"

Đánh lại trọng số với TF-IDF

Trong hầu hết các ngôn ngữ, có một số từ có xu hướng xuất hiện thường xuyên như trong tiếng anh có "is", "the"... tương tự tiếng việt có các từ như "là", "của", "cứ"... Chính vì vậy nếu chỉ xét theo tần số xuất hiện của từng từ thì việc phân loại văn bản rất có thể cho kết quả sai dẫn tỷ lệ chính xác sẽ thấp. Vậy giải pháp là gì ?

Phương pháp phổ biến là sử dụng một phương pháp thống kê có tên là **TF-IDF**, giá trị **TF-IDF** của một từ là một con số thu được qua thống kê thể hiện mức độ quan trọng của từ này trong một văn bản, mà bản thân văn bản đang xét nằm trong một tập hợp các văn bản.

Mà tại sao phương pháp lại có tên là **TF-IDF**.

Đầu tiên, TF(Term Frequency) là tần số xuất hiện của 1 từ trong 1 văn bản có cách tính như sau:

$$f_{t,d} \Big/ \sum_{t' \in d} f_{t',d}$$

- **f(t,d)** - số lần xuất hiện từ t trong văn bản d.
- mẫu số là tổng số từ trong văn bản d

Tiếp theo là IDF (Inverse Document Frequency): Tần số nghịch của 1 từ trong tập văn bản (corpus).

Mục đích của việc tính IDF là giảm giá trị của các từ thường xuyên xuất hiện như "is", "the"... Do các từ này không mang nhiều ý nghĩa trong việc phân loại văn bản.

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

* **|D|**: tổng số văn bản trong tập D * mẫu số là số văn bản có chứa từ t . Nếu từ đó không xuất hiện ở bất cứ 1 văn bản nào trong tập thì mẫu số sẽ bằng 0 => phép chia cho không không hợp lệ, vì thế với trường hợp này thường cộng thêm 1 vào mẫu số.
Và cuối cùng TF-IDF bằng:

$$\mathrm{tfidf}(t, d, D) = \mathrm{tf}(t, d) \times \mathrm{idf}(t, D)$$

Những từ có giá trị TF-IDF cao là những từ xuất hiện nhiều trong văn bản này, và xuất hiện ít trong các văn bản khác. Việc này giúp lọc ra những từ phổ biến và giữ lại những từ có giá trị cao (từ khoá của văn bản đó).

Áp dụng

Dưới đây là code thực hiện thuật toán trên, mình sẽ tiến hành làm từng bước cho dễ hiểu.

Đầu tiên là bước tạo vector như trên

```
docA = "the quick brown fox jumps over the lazy dog and"
docB = "never jump over the lazy dog quickly"

bowA = docA.split(" ")
bowB = docB.split(" ")

#Create dictionary
word_dict = set(bowA).union(set(bowB))

wordDictA = dict.fromkeys(word_dict, 0)
wordDictB = dict.fromkeys(word_dict, 0)

#count the word in bads
for word in bowA:
    wordDictA[word]+=1

for word in bowB:
    wordDictB[word]+=1

wordDictA
# {'and': 1,
#  'brown': 1,
#  'dog': 1,
#  'fox': 1,
#  'jump': 0,
#  'jumps': 1,
#  'lazy': 1,
#  'never': 0,
#  'over': 1,
#  'quick': 1,
#  'quickly': 0,
#  'the': 2}
```

Bước 2: Tính TF

```
#calculate TF
def compute_TF(word_dict, bow):
    tf_dict = {}
    bow_count = len(bow)
    for word, count in word_dict.iteritems():
        tf_dict[word] = count/float(bow_count)

    return tf_dict

tf_bowA = compute_TF(wordDictA, bowA)
tf_bowB = compute_TF(wordDictB, bowB)
tf_bowB
# {'and': 0.0,
#  'brown': 0.0,
#  'dog': 0.14285714285714285,
#  'fox': 0.0,
#  'jump': 0.14285714285714285,
#  'jumps': 0.0,
#  'lazy': 0.14285714285714285,
#  'never': 0.14285714285714285,
#  'over': 0.14285714285714285,
#  'quick': 0.0,
#  'quickly': 0.14285714285714285,
#  'the': 0.14285714285714285}
```

Bước 3: Tính IDF

```
def compute_IDF(doc_list):
    import math
    idf_dict = {}
    N = len(doc_list)

    #count number of documents that contain this word
    idf_dict = dict.fromkeys(doc_list[0].keys(), 0)
    for doc in doc_list:
        for word, count in doc.iteritems():
            if count > 0:
                idf_dict[word] += 1

    for word, count in idf_dict.iteritems():
        idf_dict[word] = math.log(N/float(count))

    return idf_dict
```

```
idfs = compute_IDF([wordDictA, wordDictB])
idfs
# {'and': 0.6931471805599453,
#  'brown': 0.6931471805599453,
#  'dog': 0.0,
#  'fox': 0.6931471805599453,
#  'jump': 0.6931471805599453,
#  'jumps': 0.6931471805599453,
#  'lazy': 0.0,
#  'never': 0.6931471805599453,
#  'over': 0.0,
#  'quick': 0.6931471805599453,
#  'quickly': 0.6931471805599453,
#  'the': 0.0}
```

Cuối cùng: Tính TF-IDF

Từ kết quả TF và IDF phía trên chúng ta chỉ cần nhân lại là xong

```
def compute_TFIDF(tf_bow, idfs):
    tfidf = {}
    for word, val in tf_bow.iteritems():
        tfidf[word] = val*idfs[word]
    return tfidf

tfidf_bowA = compute_TFIDF(tf_bowA, idfs)
tfidf_bowB = compute_TFIDF(tf_bowB, idfs)
```

```
df = pd.DataFrame([tfidf_bowA, tfidf_bowB])
df
```

Out[178]:

	and	brown	dog	fox	jump	jumps	lazy	never	over	quick	quickly	the
0	0.069315	0.069315	0.0	0.069315	0.000000	0.069315	0.0	0.000000	0.0	0.069315	0.000000	0.0
1	0.000000	0.000000	0.0	0.000000	0.099021	0.000000	0.0	0.099021	0.0	0.000000	0.099021	0.0

Từ kết quả trên có thể nhìn thấy những từ có trọng số càng cao thì những từ đó càng có giá trị phân loại và ngược lại ví dụ như từ "the" xuất hiện nhiều nên sẽ không có giá trị phân loại các văn bản với nhau. Tuy nhiên bộ train data lần này ít nên hiệu quả không rõ rệt, nếu bạn thử trên lượng data lớn chắc chắn sẽ rất hiệu quả.

Kết luận

Bài viết này mình đã giới thiệu với các bạn thuật toán Bag of Words và TF-IDF, phương pháp này rất hữu ích trong công việc xử lý ngôn ngữ tự nhiên, lọc mail spam, tìm kiếm ... Ngoài cách code như trên, bạn hoàn toàn có thể dùng thư viện của sklearn tại đây (http://scikit-learn.org/stable/modules/generated /sklearn.feature_extraction.text.TfidfVectorizer.html). Hẹn gặp lại các bạn bài viết sau.

1 Comment codetudau Login

Recommend Share Sort by Best

Join the discussion…

LOG IN WITH OR SIGN UP WITH DISQUS ?

Name

Phuc Coi • 3 months ago

hay

• Reply • Share ›

ALSO ON CODETUDAU

Phân tích cây quyết định với scikit-learn | Code từ đâu - Code từ đầu

1 comment • a year ago

Hằng Phan Thúy — đúng thứ đang tìm, cảm ơn anh.

Tìm hiểu phương pháp k-means | Code từ đâu - Code từ đầu

4 comments • 9 months ago

Cao Thinh — Cảm ơn bạn vì bài chia sẻ rất hữu ích. Cho mình hỏi một số parameters của Kmeans1) random state =0 nghĩa là gì?2)

© 2018 Code từ đầu - machine learning (../) All rights reserved.
(index.html#)