Latent Semantic Indexing with Weighted Cosine and Low Rank Approximation

Rishan Shah

June 2019

1 Introduction

Here we discuss with an example about how to use weighted cosine with latent semantic indexing in low rank approximation. Please note that this is an article documenting experimentation and findings and not a research paper!

Previous understanding of singular value decomposition and cosine similarity is assumed

2 The Example

Lets take for example 5 sentences:

1. the quick brown fox jumped over the lazy dog
2. the brown fox was very quick
3. the dog quickly jumped over the brown wall
4. the dog is quick and brown
5. brown is the colour of my jacket

We create a system in which we wish to establish all relevant sentences based on the semantic similarity. We attempt to establish this similarity with the use of Cosine Similarity with Weightings and reduce the dimensionality of the data set with a low rank approximation using Singular value decomposition.

What this article essential aims to detail is how to

1. weight the important words which we wish to have more distinguishing power and
2. calculate the similarity given certain query elements or data sets of query elements

This can come in handy especially with document record ”fuzzy matching” where records have similar descriptions between data sets.

3 The method

firstly we would need to vectorize the sentences above in some shape or form. The options are either using Word2Vec or Countvectorizer from scikitlearn. It all depends on what you are trying to achieve. If it is a bag of words model then countvectorizer would be sufficient, especially if you don’t expect the words to change however if the words are changing then you might want to consider Word2Vec or continuous bag of words methods (CBOW)
All the unique words within all the sentences above are:
’and’, ’brown’, ’colour’, ’dog’, ’fox’, ’is’, ’jacket’, ’jumped’, ’lazy’, ’my’, ’of’, ’over’, ’quick’, ’quickly’, ’the’, ’very’, ’wall’, ’was’

so if we vectorize the first sentence we would get:

$$\begin{pmatrix} 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 2 & 0 & 0 & 0 \end{pmatrix}$$

$$X = \left(\begin{array}{cccccccccccccccccc} 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 2 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{array}\right) \begin{array}{c} s1 \\ s2 \\ s3 \\ s4 \\ s5 \end{array}$$

We then let
$A = X^T$ and calculate the sigular value decomposition of A

$$A = USV^T \tag{1}$$

This can be accomplished by many python libraries easily although for larger data sets it might be worth using libraries that are used to dealing with sparse matrices such as scipy. To reduce dimensionality we take the first k columns of U, first k rows of V^T and first k rows and columns of S so we obtain

$$A \approx U_k S_k V_k^T \tag{2}$$

We select k based on the elbow method where we see an elbow in the eigen values.

Once we have reached this situation we wish to query the data with a sentence of words to find all possible matched sentences. Bear in mind that our assumption means that we only have to work with the words in the ”bag”.

so lets say we use the following sentences:

- the quick brown fox jumped over the wall
- the dog is brown
- brown wall

This would translate to the following matrix

$$A_q = \begin{array}{c} \left(\begin{array}{cccccccccccccccccc} 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 2 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{array} \right) \end{array} \begin{array}{c} q1 \\ q2 \\ q3 \end{array}$$

We need to convert this query matrix in to the same eigen space so we do the following:

$$A'_q = A_q U_k S_k \tag{3}$$

and we then calculate the cosine similarity between A'_q and V^T using the following python code

```
1-sp.distance.cdist(A_q,vt.T,’cosine’)
```

As a result we would establish the following similarity matrix:

```
array([[ 0.91415118,  0.52130491,  0.65736864,
    -0.21772781,  0.06131547],
        [ 0.20051365, -0.21585225,  0.23433497,
           0.90436882,  0.31913829],
        [ 0.45049518, -0.07268552,  0.77682868,
           0.01197107,  0.67650256]])
```

This is the similarity without the weight vector. We use the following python code to establish a weight vector where we give the words ”fox” and ”dog” 20 times more importance:

```
def create_weight_vector(inputlist, importantfeaturedict):
    indexlist = [i for i, x in enumerate(inputlist) if x in importantfeaturedict.keys()]
    weightvector = np.ones( len(inputlist))
    for i in indexlist:
        weightvector[i]=importantfeaturedict[inputlist[i]]
    return weightvector

dic = {’fox’:20,’dog’:20}

vec = create_weight_vector(vectorizer.get_feature_names(),dic)
vec = vec.dot(u).dot(sigmainv)
for i in range(0, len(vec)):
    if vec[i] < 0:
        vec[i] = 0.001
vec
```

to get a corresponding weight vector in the same eigen space:

```
array([1.18495556e+00, 2.55904237e+00, 1.00000000e-03, 1.00000000e-03])
```

We rather crudely remove all the negative elements of the weight vector and assign a really small weight as we wish to only give some dimensions importance and not take away importance from others. This still works

we then use the following python code to get the similarity using the weightings:

```
1-sp.distance.cdist(A_q,vt.T,’cosine’,w = vec)
```

and we obtain:

```
array([[ 0.53637213,  0.46152368, -0.01964532, -0.90456463,  0.46842397],
       [-0.67684195, -0.61690285,  0.20475949,  0.81274154, -0.29760435],
       [-0.74664195, -0.80416185,  0.98427501, -0.55769161,  0.94411959]])
```

4 Conclusions

We can see that q1 matches heavily with s1 with a cosine distance of 0.91415118 as many of the words are the same. However when we employ the weight vector and increase the importance of the words ”fox” and ”dog” to be 20 times more important than the other words we can see that this significantly reduces the cosine similarity to each other down to 0.51143349. This just allows us to use words we feel are more important in specific contexts that matching has to happen. Obviously this can mutate the data in the eigen space however with correct implementation and well defined weights it can work.

5 Further Improvements

This method can be further improved by using sentence embedding rather than bag of words models and perhaps also looking for the elbow in eigen values to find the best low rank approximation. We can also look to remove stop words and use a validation data set and gradient boosting trees to establish the feature importance (the features being words or a combination or words if we choose to use n-grams)