Project Milestone

Introduction

The goal of this project is to autotag an article or short sentences. In this project, we use Wikipedia articles and created sentences to train and evaluate the models. At the current stage, we have implemented two main algorithms that are able evaluate the relevance between two Wikipedia articles and between two short sentences.

Methods

Step 1: Obtain the training set

We have two sources of training data.
1) Wikipedia articles (of which relevance to each other will be evaluated)
2) Word corpora to create word vectors (see detail in algorithm section) Our training set is the text8.zip file from http://mattmahoney.net/dc/, which is the resource that Google releases for creating the vector representation of words on the latest research tool *TensorFlow*™. This file contains many articles, having in total above 17 million words, which is sufficient to create a vector representation of almost all English words.

Step 2: Algorithm

We implemented two main algorithms to evaluate relevance of two Wikipedia articles or two short sentences (which will be extended to autotagging in the future). Both algorithms base on constructing article vectors which represent word count in each article. The simple article simply calculates the relevance of two articles by finding the cosine similarity (see pseudo-code) between two vectors. This similarity will range from 0 to 1.

The more complex algorithm leverages the knowledge that different words can have similar meaning. We apply the gensim library, which internally applies the deep learning algorithm, especially hierarchical softmax skip-gram algorithm to the word corpora. Given a training set which is a collections of sentences (each sentence is a collection of words), word2vec function in gensim library will produce word vectors (in 128 dimensions) corresponded to each word input. Now given the articles vectors, we can redefine the function that find the similarity between two articles based on the word vectors. We in fact have implemented two algorithms to do so.

Simple Algorithm

```
relevance = dotProduct(counter1, counter2)/(‖counter1‖_2×‖counter2‖_2)
```

1st Algorithm: We iterate through all combinations of pairs of word where one word is in the counter of the first article and another word is in the counter of the second article. Then, we calculate the summation of the product of occurrences of both words and the similarity relevance we have from the model (where we can compute similarity(word$_1$, word$_2$) from the model obtained from the training data). Then we normalize the value we obtain by dividing the product of L1-norm for both articles.

```
relevance = 0
for word1 in counter1:
    for word2 in counter2:
        relevance += counter1[word1]*counter2[word2]*\
                     similarity(word1, word2)

relevance = relevance/($\|counter1\|_1 \times \|counter2\|_1$)
```

2nd Algorithm: This algorithm is similar to the first version, but the only different is we will calculate the summation of the product of occurrences of both word only in case that similarity(word$_1$, word$_2$) is equal to and greater than the threshold (we currently use threshold = 0.5.)

```
relevance = 0
threshold = 0.5 // Can be other values
for word1 in counter1:
    for word2 in counter2:
        if similarity(word1, word2) >= threshold:
            relevance += counter1[word1]*\
                         counter2[word2]*\
```

`similarity(word1, word2)`

`relevance = relevance/(`$\|counter1\|_1 \times \|counter2\|_1$`)`

Preliminary Results

We ran the new algorithm and old algorithm on Wikipedia articles and short sentences. Results are summarized in the tables below.

Wikipedia Pages

Page Title		Similarity		
First Page	**Second Page**	**Old Algorithm**	**First Algorithm**	**Second Algorithm**
Thailand	Bangkok	1.00	1.00	1.00
Thailand	Country	0.48	1.03	0.97
New York City	Statue of Liberty	1.02	0.87	0.98
Japan	Sushi	0.63	0.63	0.73
Computer	Alan Turing	0.72	0.90	1.06
Pad Thai	Sushi	0.58	1.76	6.00
Stanford University	University	0.73	0.95	1.78
Electricity	Electric Car	1.1	1.46	1.45
Elephant	Cat	0.93	1.67	2.20
Petroleum	Diesel Engine	0.62	1.90	3.00
Logic	Set Theory	0.64	2.25	4.22
Bee	Metamorphosis	0.48	1.94	3.70
Paper	Tree	0.59	1.733	2.10
Thailand	Axiom	0.40	0.36	0.33
Bee	New York City	0.24	0.60	0.543
Car	Alan Turing	0.55	0.65	0.65
Bicycle	Tree	0.75	1.66	1.22

Green: words are relevant; words are proper nouns
Yellow: words are relevant; words are generic nouns
Red: words are irrelevant
Note that this categorization is evaluated by humans. The values in the table are normalized by the similarities between "Thailand" and "Bangkok" obtained from three different algorithms.

Short Sentences

Short Sentences		Similarity		
1st	**2nd**	**Old Algorithm**	**First Algorithm**	**Second Algorithm**
I love dog	Dogs are lovely	0	1.7	2.49
I love dog	This puppy is cute	0	0.98	1.66
I love dog	This kitten is cute	0	0.86	0
I love cat	This kitten is cute	0	0.82	1.66
I witnessed a car incident yesterday	My father works at an insurance company	0	0.36	0.60
I will go to the beach this weekend	I really need to take a vacation	1.0	1.0	1

I love Japanese people	I love Japan because it is a beautiful country	2.5	1.33	1.24
Donald Trump is great	I love Republican party	0	0.38	0
It is really hot here	I hate summer	0	0.155	0

The values in the table are normalized by the similarities between "I will go to the beach this weekend" and "I really need to take a vacation" obtained from three different algorithms.

We can see that when evaluating Wikipedia pages relevance, all three algorithms do a fine job. The two complex algorithms however perform better than the simple one. I.e. they give a wider range of similarity value. In addition, while the simple algorithm fails to recognize the similarity between two articles that do not seem very obvious, such as "Logic" and "Set Theory"; "Diesel Engine" and "Petroleum", the two complex algorithms seem to be able to recognize the similarity well. This is because the two complex algorithms use the knowledge that words can have similar meaning although they are not the same, while the old algorithm only look for the same words in both articles. It is also worth noting that the two complex algorithms seem to yield lower similarity value when comparing articles with proper noun title. This is possible because when we trained word2vec, there are not many proper nouns in our word corpora.

A difference between simple and complex algorithm becomes more apparent when we test them on short sentences. The simple algorithm seems not to be able to recognise the similarity between two sentences with no overlapping words. The complex algorithm however seems to recognise the similarity between words in the two sentences and produce non-zero relevance.

The main difference between the first and second algorithm is that the first algorithm seems to produce similarity values that are less extreme, while the second algorithm sometimes give a really high similarity value. This is mainly because we set a threshold at 0.5 (which is quite high), and the second algorithm will count two words as similar when they are in fact very similar while the first algorithm will add the product of similarities of any two words, however low that may be.

Challenges and future work

Article vectors

Our "article vectors" are currently based on frequency counts of words. However, we are considering training large number of sentences to make a simple program to map locus of word vectors to a higher dimensional vector that represent the whole articles. This can be done by using a parser to take in two sentence from Wikipedia article as one token, where we approximate the relevance of the two sentences by 1{the two sentences are from the same article} or approximatedSimilarity{article1, article2}.

Autotagging and modified Classification

We will find a model to assign a score to each word in the English dictionary and "classify" each articles into these words categories. (Only words with high scores will be considered as a possible category.) We may try to use some other unsupervised learning (k-means cluster), or training actual English dictionary as a prior model.

Citations

[1] Sojka, Petr. "Software Framework for Topic Modelling with Large Corpora." Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks. By Radim Eh Ek. Valletta, Malta: ELRA, 2010. 45-50. Print.

[2] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. In Proceedings of Workshop at ICLR, 2013.

[3] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed Representations of Words and Phrases and their Compositionality. In Proceedings of NIPS, 2013.

[4] "Vector Representations of Words." Tensorflow.org Google Inc. 2015