Extracting keywords from email data using distributed word vectors

Irving Rodriguez
Department of Physics, Stanford University, Stanford, California 94305, USA
(Dated: December 14, 2015)

Current keyword extraction methods often use statistical models to select certain words from a set of documents, which fail to take advantage of the information available in the documents themselves. We propose a model for identifying semantic relationships between words in a document to identify keywords that more accurately capture the meaning of the document. Specifically, we use distributed word vectors to learn hypernymy and test the relationship on the contents of emails to deduce meaningful keywords. We find that this model is capable of producing a list of words that contain such keywords, though more analysis is necessary on learning the hypernymy relationship between words to quantitatively narrow this list to only those keywords.

Introduction

The goal of keyword extraction is to identify the most relevant words within a corpus. By successfully identifying these terms, it becomes easier to condense the conceptual information contained in these documents into a short list of unambiguous and specific words. This is especially useful for extracting information in large sets of documents, like Wikipedia, or very dense, technical corpora, such as a set of scientific papers.

Currently, many methods in keyword extraction rely on purely statistical models to deduce keywords for every document. Many of the leading algorithms use a variation of the term frequency-inverse document frequency measure as a proxy for relevant terms (Eq. 1). For a corpus with D documents, the TF-IDF measure t_d for term $w_i$in document d is

$$t_d = f_{raw}(w_i) \times log(\frac{D}{\sum_{j=1}^{D} \delta_{ij}}) \tag{1}$$

where f is the frequency of the term w_i in document d and $\delta_{ij} = 1$ if w_i is in document j, 0 otherwise. The key assumption made by these models is that, for a given document, its keywords will likely not appear in many other documents within the corpus, but may occur frequently in the document of interest.

For example, in a set of documents describing NBA teams, we expect that "Oakland" will likely occur more frequently under the "Golden State Warriors" article than the "Chicago Bulls" piece, and will likely not appear often in any of the other teams' documents, hence this relevant term describing the location where the Warriors play would have a low value of t in Eq. 1 and would consequently be identified as a keyword.

However, these methods fail to take into account the rich semantic information available within the documents, and such an assumption does not hold very well for less descriptive documents, like emails. An email from a friend stating that he bought tickets to the Warriors game may never explicitly mention "basketball", but such a term certainly captures a lot of the meaning in the message. A person's inbox can contain hundreds or thousands of these messages with information meaningful to the user, and can be useful for tagging future incoming emails by content type.

In order to remedy this disparity in keyword extraction, we use distributed word vectors to learn semantic relationships between words. Hypernymy, the relationship between a word and a more general term, is particularly suited for this task. We use a large subset of the Wikipedia corpus to learn vectors for a large vocabulary of words using a neural network implementation of word2vec[1], the publicly available toolkit, and we proceed to learn hypernymy by feeding hyponym-hypernym pairs ("dog", "animal") into a cluster-projection model that outputs a potential hypernym vector of a given word.

Background

The choice of algorithm for learning word vectors is important for understanding their use in learning semantic relationships. As such, we will discuss their construction in some detail.

One of the first learning algorithms for word vectors came from Bengio et. al.[2] By using a statistical language model, we can predict which word should come next in a sequence based on the conditional probability given the previous words in the sequence; these words were known as the "context" for a given word. That is, to predict the next word w_i, we condition on all words prior for a sequence of T words:

$$P(w_i) = \prod_{t=1}^{T} P(w_i | w_i^{t-1}) \tag{2}$$

Bengio proposed building a neural network in order to capture the grammatical and semantic information contained in the many contexts of a word, using a large text corpus as an input in order to capture this information in a vector. Bengio's neural network consisted of four layers:

Input layer: The input word is represented using a $|V|$-dimensional vector, where V is the vocabulary of distinct

words in the corpus, whose only non-zero entry is the i^{th} entry corresponding to the word's order in the vocabulary.

Projection Layer: The word vector from the input layer is mapped onto a real-valued vector $C(i)$ using a $|V| \times m$- dimensional matrix, C, where m is the number of features used (such that the resulting vectors from training will be m dimensional).

Hidden layer: This layer calculates the unnormalized log-probability, y_i of each output vector $C(i)$using a hyperbolic tangent function:

$$y = tanh(d + HC(i)) \tag{3}$$

where H is the $h \times (n-1)m$ hidden weights matrix, for h the number of hidden units and $(n-1)$ the number of words used in the context.[**?**] The hidden layer serves to compute the probability distribution over the whole of the vocabulary.

Output layer: This layer uses a softmax function to normalize the output of the hidden layer, allowing for a probabilistic interpretation of the vector w_t:

$$P(w_t|w_{t-1}, ..., w_{t-n+1}) = \frac{e^{y_{w_t}}}{\sum_i e^{y_i}} \tag{4}$$

Conceptually, the neural network learns the matrix C which maximizes the average log-probability of each word over the corpus:

$$L = \frac{1}{T}\sum_{t=1}^{T} logP(w_t|w_1^{t-1}; C) \tag{5}$$

Bengio's algorithm is very expensive to compute and required weeks to train vectors of size 60 on a corpus of only 1 million tokens.[2]

Mikolov et. al took this idea further in their first paper and modified Bengio's algorithm to make word vectors a feasible tool for language analysis. The time complexity Q of Bengio's neural network showed that the limiting factor was the hidden layer[**?**] :

$$Q = n \times m + n \times m \times h + h \times |V| \tag{6}$$

where each term is the sum is the projection, hidden, and output layer, respectively. The hidden layer here is highly non-linear.

They provided the **continuous skip-gram algorithm** as an alternative that cut out the hyperbolic tangent layer entirely while preserving the probabilistic assumptions behind Bengio's architecture.[3]

Continuous skip-grams expand the context to include both words that precede and follow the desired word as specified by a window size. The context is also not limited to its immediate context, and can skip a constant number of words. As a result, the algorithm now maximizes the average log-probability of the context given a word:

$$L = \frac{1}{T}\sum_{t=1}^{T}(\sum_{-c \leq j \leq c} logP(w_{t+j}|w_t; C)) \tag{7}$$

for a context range (called the window size) of c.

The input and projection layers remain the same. With these modifications, the neural network could learn 600 dimensional vectors using corpora containing 6 billion tokens in a matter of hours.[3]

Most interestingly, however, the vectors could display knowledge of various relationships using simple vector algebra. Perhaps most famously, the operation v(king) - v(man) + v(queen) yields the vector closest to "woman". Thus, the model could be tested using an analogy task, where the recall of the model could be measured by its ability to successfully return the last word in a string of four tokens, (a, b, c, d) if a is to b as c is to _? and the model can return v(d). In a second paper, Mikolov et. al provide two more additions to their word vector learning models in an effort to improve upon its accuracy.

The first is the ability to subsample from the training corpus. Because there are a multitude of frequent words that appear during learning that do not offer much grammatical or semantic information, like "the" and "in", Mikolov discards the word w_i randomly according to the probability

$$P(w_i) = 1 - \sqrt{\frac{\tau}{f(w_i)}} \tag{8}$$

where τ is a chosen subsampling threshold and f is the frequency of the word in the corpus. [4] Words with higher frequencies will then be subsampled more aggressively. This can decrease the training time significantly as well as bolster the level of information available from contexts since the subsampled tokens will then be replaced by other words that previously fell outside the window size.

Negative sampling also becomes an option during skip-gram learning. Mikolov uses the noise distribution, $U(w)$,to randomly sample tokens from the corpus into a noise set D. They use a unigram distribution to the $\frac{3}{4}th$ power. They weight the conditional probability of each word w_i by $|w_i \cdot w_l|$ with a random word w_l drawn from D, to produce higher-quality word vectors can be produced with less training time.[5] The idea is that if the word pairs well with the context represented by the drawn vector, the dot-product will increase the calculated probability during training, yielding a better vector in less time.

Methodology

Word vectors ability to learn various relationships, like the capital city-country mapping, have motivated attempts to learn semantic mappings through vector algebra. We use similar efforts to learn a hypernymy mapping between a word and a more general counterpart. Many of the previous attempts rely simply on the difference between the hyponym vector, the less general word,

with its hypernym vector, to learn a linear mapping.[6] [7]. Though high levels of recall, often 80% or greater, preliminary tests show that such mappings are not easily applied to a wide variety of hypernyms. We found, for example, that the classifier described in Weeds will output "animal" as a hypernym of "dog", but not "university" for "Stanford" or "vehicle"/"automobile" for "car", among others.

Instead, Fu et. al have shown that these vector differences form many different clusters in vector space (Fig **X**).[8] This motivates an approach of using kmeans clustering in order to identify the various "kinds" of hypernymy before learning a linear mapping between a hyponym to its hypernym.

Kmeans clustering seeks to group together N pairs of (w_o, w_h) hyponym-hypernym examples (respectively) into k clusters. Using the difference between the two vectors as a feature, the algorithm will initialize k centroids in vector space and iterate through each example until they are all grouped into the clusters according to which centroid their difference is closest to.

We will then learn k linear mappings from hyponym to hypernym. More formally, we hypothesize that there exists a transformation H_i for each of the i clusters that maps a vector w_o to its hypernym, w_h. We are looking for an H that minimizes the norm of the difference between the two,

$$H^* = argmin_H \frac{1}{T} \sum^{T} |H(w_o) - w_h|^2 \tag{9}$$

Armed with word vectors and a hypernymy mapping, we can identify the TF-IDF keywords from an email set and map to their hypernyms in order to output a more meaningful set of keywords.

Experimental Set-Up

Word vector training

In order to represent words as vectors, we use half of the English Wikipedia as our training corpus. We use the wikiExtractor, available on the MediaWiki website, to parse the first half of the July 2015 English dump.[9] We iteratively input each sentence into the publicly available word2vec implementation of the continuous skip-gram algorithm.

The word2vec model requires a parameter set specifiying the dimension of the desired vectors, the window size of the context, and the subsampling threshold. We determine the optimal set of parameters by performing the analogy task described above with a provided list of 7910 sets of 4-string tokens.

Hypernymy detector training

Baroni et. al have compiled a list of word pairs for various semantic relationships, including hypernymy, by parsing for specific syntactic patterns. For example, corpora that contain the phrase X, such as Y are often indicative of hypernymy (i.e, fruits, such as apples). This data set contains 1268 hyponym-hypernym word pairs.[10]

We concluded that this was the best collection of semantic relationship examples for our task due to the size, variety, and generality of the provided word pairs. Other sets included pairs that, while technically hypernyms, seemed too specific to capture the general relationship using our model. For comparison, the Baroni set includes such pairs as [swallow, bird] and [baseball, sport][10] while another similar set from Weeds includes pairs like [hurry, necessity] and [emergency, juncture].[6] This seems to approach a level of granularity in hypernyms that would be better explored in later iterations of the model.

We use 10-fold cross validation, using 8 subsets of the samples for training, one subset as a testing sample of true hypernym pairs, and we flip the order of the remaining samples to have negative test samples (since baseball is not a hypernym of sport).

Not every pair of words that will be input into the hypernymy detector model will necessarily be a hyponym-hypernym pair. As such, we only classify pairs whose norm-difference between the potential hypernym and the projection of the word are below some threshold δ. In order to find the ideal combination of clusters and an appropriate threshold, we use the cross-validation to test several pairs of parameters and output the average percent of correct classification. Additionally, we keep the pair in each cluster whose vector difference was closest to the centroid and call it the indicative pair for that type of hypernym.

Application on email data

We use the publicly available Gmail API to pull 100 of the most recent non-spam emails from a personal mailbox.[**?**]. We create a vector t whose i^{th} entry is the TF-IDF value of the i^{th} word in the email vocabulary according to Eq. 1. We normalize the vector so that the length of the email does not artificially affect the relative value of t_d among emails with differing word counts.

We then note that a keyword can have hypernym pairs that appear in different clusters (i.e. dog-animal may lie in one cluster, but dog-canine can be in another). Therefore, for each cluster i, we take indicative pair and use v(knownHyponym) v(knownHypernym) + v(keyword) to generate a potential list of hypernyms for the keyword. For the top 10 results, if $|H_i(keyword) - v(potentialHypernym)|$ is greater than our chosen δ, we classify it as a hypernym.

For testing the accuracy of our hypernyms, we randomly chose 20 of the emails that outputted more than 5 hypernyms and presented them to a human grader. We

Parameter Set	Analogy Task Accuracy
d = 100, c = 10, s = 1e-5	64.10%
d = 200, c = 10, s = 1e-5	75.50
d = 300, c = 10, s = 1e-5	79.10
d = 400, c = 10, s = 1e-5	79.90
d = 500, c = 10, s = 1e-5	81.50
d = 200, c = 8, s = 1e-5	72.50
d = 200, c = 12, s = 1e-5	78.50
d = 200, c = 14, s = 1e-5	81.30
d = 200, c = 10, s = 1e-6	72.60
d = 200, c = 10, s = 2e-5	77.30

TABLE I. **Word Vector Training**. Performance on analogy task with varying parameter sets, where d is the dimensionality of the vector, c the window size, and s the subsampling threshold.

also presented them with S_1, the list of 5 keywords, and S_2, the list of the top 5 hypernyms. We asked them the following questions and tallied the results for each of them.

1) Do you think at least 2 words from S_1 describe the email well?
2) Do you think at least 2 words from S_2 describe the email well?
3) Which set, S_1 or S_2, is more descriptive of the email?

Results and Discussion

For our word vector model, we found the optimal parameter set to be $(300, 14, 1e-5)$. (Table 1) The vocabulary size for this corpus after subsampling was $41,000$ words.

For the dimensionality, we observed that the accuracy of the model only marginally increased with large increases in the number of features used to construct the word vector. As the time complexity suggests, the training time is non-linear with the dimension of the vector. For $d = 300$, total training time surpassed 8 hours, and the run for 500 lasted over 16. This is, of course, due to the exponentially larger number of calculations that the model has to process during each iteration in the projection layer. However, it is likely that there is little new information obtainable by the word vector itself. After all, Wikipedia articles are subject to very strict editing guidelines, and the method of writing is rather constant across all articles. The literature also suggests that $300 - 400$ is a sufficient number of features for a variety of tasks. Given the time constraints, we chose to use 200 to test the remaining two parameters.

We discovered that a larger window size continued to increase the accuracy of the model. While we did not expect the accuracy to plateau, it is surprising that a window size of 14 still saw a significant increase, considering that we were using 200 dimensional vectors and it is hard to believe that the average Wikipedia sentence is longer than 29 words, especially when we already omit most articles and pronouns. The addition of the extra words likely means that the projection output, which averages over all of the skip-gram contexts, is still getting pushed towards a "truer" mean in order to increase the probability of the context happening. This makes sense, as there are more and more "unique" contexts, especially when we are reaching out past 10 words in either direction. There seems to be litlte lost beyond computation time in using a larger window size, so we stuck to 14.

As for the subsample threshold, we decided to use the $1e-5$ used in the literature.[4] Though there is an increase in using a larger threshold, we fear that doubling it may start ommiting words that provide meaning to the contexts they are appearing in. Because we are already using such large vectors, we were conservative with this measure, despite how heuristic the choice may seem.

In the hypernymy detector, the choice of parameters was much more heuristic than we had hoped. The negative classification accuracy offers great insight into the behavior of the threshold, but it is still difficult to gauge the balance. For too low a threshold, the detector is biased toward classifying examples as negative across the number of clusters, given that it forms too strict a distance around the centroids. However, there appears to be more stability for a cluster size of 20, after 25 demonstrated a much larger drop in negative classification between the two tested thresholds. It is likely that, in this case, the high number of centroids began to come arbitrarily close, which affected the classification of some of the examples. Indeed, a few sanity checks showed that it even misclassified an animal as a type of dog. The reverse, however, was not true for $k = 20$, as many of our sanity checks, which consisted of animals and types of foods, were correctly classified with the last set of parameters.

The email results were less convincing. The human judge rendered answered yes to the first question 17 times, yes to the second question only 7 times, and S_1 as the more descriptive set in every single instance save one. (Upon further inspection, she admitted it was solely because she did not want to answer all of the questions with a single answer)

The source is likely the use of TF-IDF as the first screen for keywords. Further tests showed that, an email about a club soccer team's regional tournament identified "region" as the top keyword, with several other examples sharing a similar fate. This is due to the appearance of "club soccer" in almost every other soccer-related email, which is one of the identified drawbacks of using TF-IDF. The model then proceeds to search for hypernyms in a keyword space that is not actually representative of the words in the email which hold the most meaning.

A simple remedy would be to simply identify all of the nouns in the email as the keywords, as these likely carry

Parameter Set	Average Positive Classification	Average Negative Classification
(k = 5, threshold = 3)	55.4%	100%
(k = 5, threshold = 9)	67.3%	95%
(k = 15, threshold = 3)	64.4%	96.4%
(k = 15, threshold = 9)	78.4%	74.3%
(k = 25, threshold = 3)	77.6%	92.3%
(k = 25, threshold = 9)	83.6%	70.1%
(k = 20, threshold = 3)	78.5%	87.7%
(k = 20, threshold = 6)	82.3%	88.5%
(k = 20, threshold = 9)	85.5%	88.5%

TABLE II. **Hypernym Detector Accuracy.** The positive accuracy shows the number of positive samples that were correctly classified, and the same for the negative, respectively.

the most information in school and professional emails. The hypernym detector could then search for better hypernyms instead of being led astray from the start.

The human judge, however, added that the set S_2seemed to contain words that were "broader" than some of those contained in S_1, suggesting that the detector could have indeed succeeded at identifying some hypernyms.

Qualitatively, a browse of the words classified as hypernyms using the email keywords did identify more general terms. For example, several of the soccer related emails contained "sport", "athletics", and "game" in the list of hypernyms, though these did not make the top 5 used to create S_2.

Conclusion

In short, we were able to train high-quality word vectors quickly using the continuous skip-gram implementation offered by word2vec. We showed their ability to handle a variety of natural language-oriented tasks, as evidenced by the high recall rate of the hypernym detector during testing, once the appropriate parameter set was chosen.

Qualitatively, we had mild success in producing a list of more general words than initially extracted from our email data. However, there is evidently a need for a mechanism which can quantitatively identify such words from among the list of false positive that our detector could produce. This might improve upon developing a more rigorous mechanism for choosing the parameter set of our detector in a way that is more consistent with the development of our algorithm and not so reliant on its performance during testing.

In the future, the main work should revolve around exploring the hypernym space further. Similar PCA analysis as Fu et. al could shed more light on the way that hypernymy could be modeled so as to be consistent with its linguistic properties. The training method successfully captured its asymmetry, but as the number of clusters suggests, there may still be more complexity in the hypernym relationship that we do not understand, and thus is not captured by our model.

There may also be more attainable information from running the detector on *pairs* of keywords. There is, for example, a different output by looking at the most similar words of (soccer, tournament) than either word individually.

Eventually, the next natural step would be to build a hypernym generator, as our detector needs a candidate in order to identify hypernymy.

We have demonstrated that word vectors are a reliable way of exploring the semantic properties of words. Nevertheless, a careful consideration of the context in which these words are used is necessary in building successful and linguistically robust models.

[1] Google, .
[2] Y. B. et. al, ().
[3] T. M. et. al, ().
[4] T. M. et. al, ().
[5] Y. Goldberg and O. Levy, .
[6] J. Weeds, .
[7] E. A. et. al, ().
[8] R. F. et. al, ().
[9] MediaWiki, .
[10] M. B. et. al, ().