Use Google's Word2Vec for movie reviews

**Abstract**

In this paper, we dig a little "deeper" into sentiment analysis. [Google's Word2Vec](https://code.google.com/p/word2vec/) is a deep-learning inspired method that focuses on word semantics. This project focuses on Word2Vec for sentiment analysis on the IMDB movie review dataset that was used for the Kaggle competition Bag of Words Meets Bag of Popcorn [1] and aims to categorize the polarity as positive or negative. We also evaluate the Paragraph Vector or doc2vec method [2]. We compare these with the bag of the words technique. Given that real world data is mostly unlabeled, the Word2Vec and Doc2Vec being unsupervised learning techniques are valuable for this problem. We see that hybrid approaches using the above methods with a bag of words model and an ensemble machine learning classifiers can lead to high AUC scores.

**Introduction**

Sentiment analysis is a challenging subject in machine learning. People express their emotions in language that is often obscured by sarcasm, ambiguity, and plays on words, all of which could be very misleading for both humans and computers. Inferring the polarity of this text has been of great business interest to companies in the recent times. Deep Learning techniques have achieved breakthrough results in the areas of computer vision, natural language processing and speech processing and we aim to evaluate the same for sentiment analysis.

The current techniques that are used for sentiment analysis are statistical machine learning approaches that use the bag of words implementation [3]. A vocabulary is created from all the individual sentences and every sentence is represented as a vector that represents the number of times each word in the sentence occurs in the vocabulary. This method however ignores word ordering. Some of these limitations can be overcome by using n-gram [4] based approach where the co-occurrence of words and the spatial information is also preserved. These models are simple and scalable however suffer and the trends are well understood. When the features get too large, dimensions can be reduced using feature selection.

Word2Vec is a deep learning technique that aims to understand the meanings of words and relationships among words. It is based on the fact that similar words tend to occur in similar contexts. It doesn’t exactly have the understand what the words mean but aims to figure out how related the words are. Often people use several words that almost mean the same thing and the challenge is to infer the relationships between the words so we can share parameters.

In the example in Figure 1, by looking at the context words, we can infer cat and kitty mean the same thing.

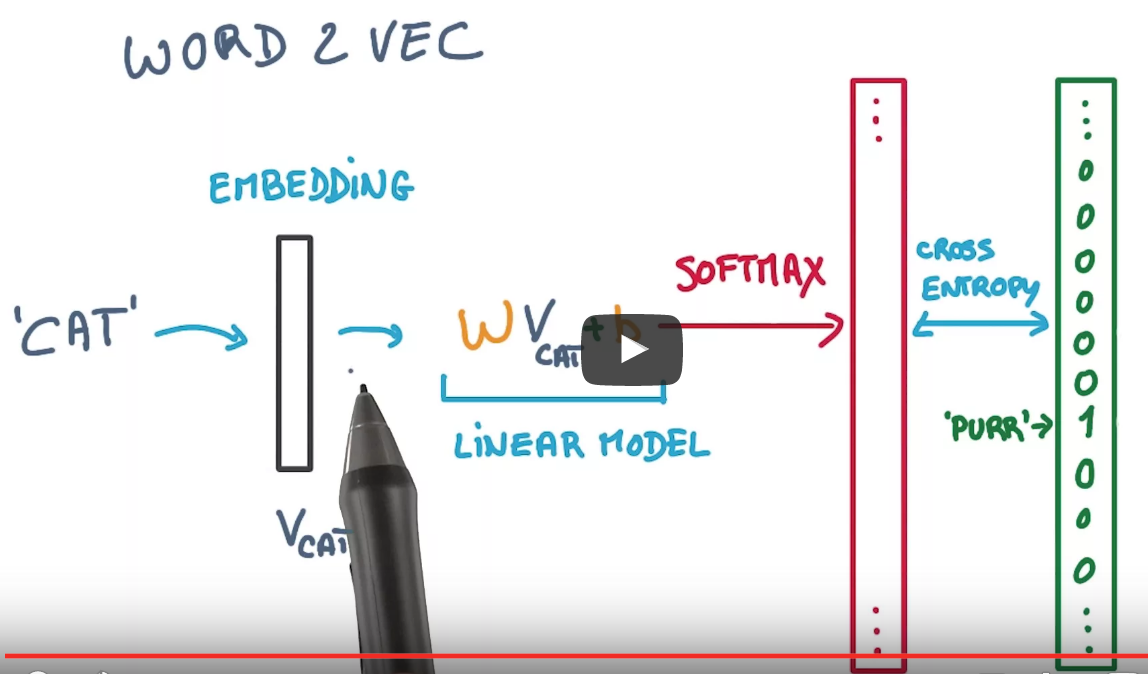
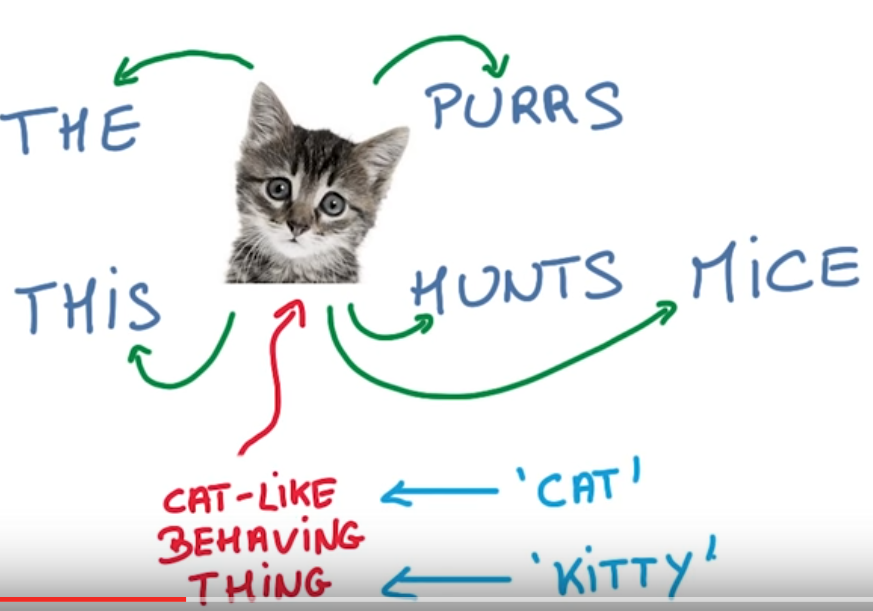


Figure 1. Word context Figure 2. Simple Word2Vec model

As shown in Figure 2, every word is represented in n-dimensional space as a word embedding and this is then fed to a linear model with weights and biases that are learnt over time to reduce the output loss function [stochastic gradient descent]. A softmax probability is derived and cross entropy [hierarchical sampling] is used to predict the output in this case, it is the absence or presence of the context words [6]. This results in a powerful word to vector representation where words with similar meanings end up having almost similar vectors.

The model can also do more than just figure word meanings. It can also infer verb tense, gender, countries vs capitals, etc.

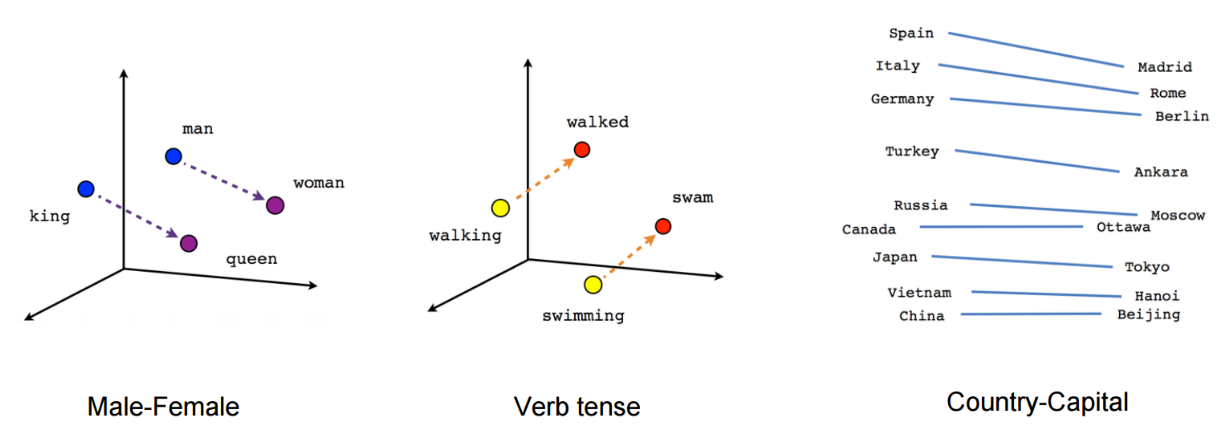


Figure 3. Word2Vec analogies

There are two word2vec architectures [8]. Skip-gram and Continuous Bag of Words. Skip-gram tends to predict the context given the word and Continuous Bag of Words predicts the word given its context. As per Mikolov [9], skip-gram tends to do better with rare words and the other with frequent words, however faster to train.

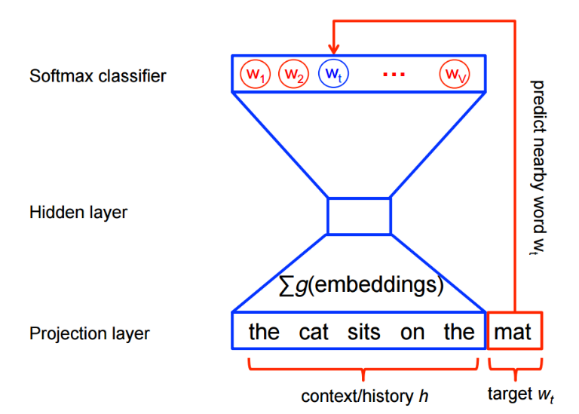
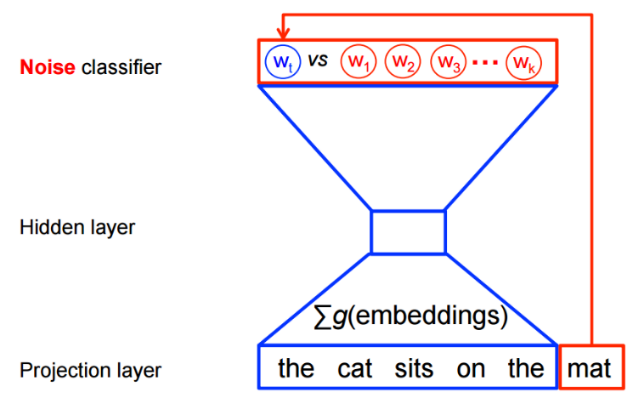


Figure 4 Continuous Bag of Words Figure 5. Skip-gram

The paragraph vector or doc2vec is another technique [10] that also uses a vector representation of the entire paragraph and uses this along with the context words to predict a word. It has 2 architectures Distributed Memory and Distributed bag of words model. In [10], we see Mikolov claiming that this approach outperforms the bag of words model given the unsupervised learning nature.

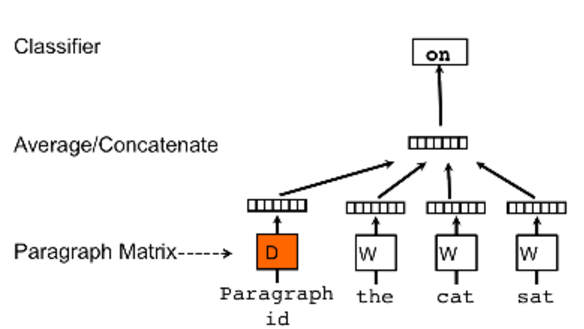
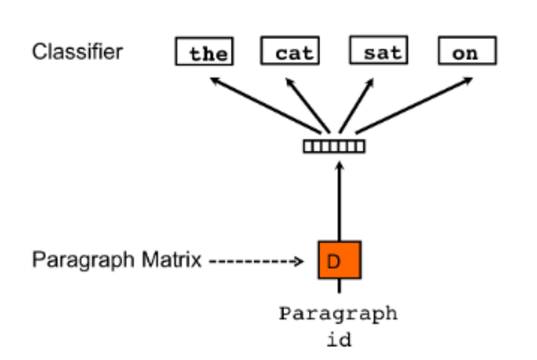


Figure 4 Distributed Memory Model Figure 5. Distributed Bag of Words

**Related Work**

In the above section, we briefly reviewed the bag of words, word2vec and the paragraph vector techniques that have been described and implemented in several papers. Google has a C implementation for word2vec [11]. Genism has an excellent python implementation for word2vec which we plan to use for the paper. It also has a doc2vec implementation, however is not very well tested [12]. Stanford recently published a deep learning solution for sentiment analysis. In this project, we aim to implement and compare these techniques with the bag of words model and also use hybrid approaches to maximize the performance results. We also compare the time and computational complexity of the various techniques.

**Data Exploration and Preprocessing**

The Kaggle IMDB dataset was used for this project. It has 25000 labeled training set [id, review, sentiment label] and a 25000 test set [id, review]. It also has 50000 unlabeled training set [id, review] which can be used to train the Word2Vec and the doc2vec model

Preprocessing of the reviews was done using NLTK techniques. Beautiful Soup package was used to remove html markup. Several modifications were done - convert to lower case, remove non-text using re package, remove stopwords, remove punctuations, use sets for faster search. However, punctuations could be valuable in inferring human emotions. Word2vec needs sentences instead of words [NLTK punkt tokenizer for sentence splitting]. Stemming was done using porter stemmer.

**Algorithms**

Once we have preprocessed the data, we need to do the feature extraction before feeding it into the machine learning classifiers. For this we used the feature\_selection library from scikit-learn.

For the bag of words approach, we used a countvectorizer and a tfidf vectorizer module and then we use a fit\_transform(). The fit\_transform learns the vocabulary and then transforms them into feature vectors, in the end we get a list of strings for each review. Limiting the max\_features limits the vocabulary and hence the model complexity. We iterated over max\_features from 300 to 200000.

vectorizer = CountVectorizer(analyzer = "word", tokenizer = None, preprocessor = None, stop\_words = None, \  
 max\_features = 5000)

vectorizer = TfidfVectorizer( min\_df=2, max\_df=0.95, max\_features = 200000, ngram\_range = ( 1, 4 ),sublinear\_tf = True ). We also varied n-grams over a range 1 to 4.

For the word2vec model, we can play around with the following features from the genism API

* Architecture: Architecture options are skip-gram (default) or continuous bag of words. We didn’t not see much difference in the two models.
* Training algorithm: Hierarchical softmax (default) or negative sampling. The default option worked well.
* Downsampling of frequent words: The Google documentation recommends values between .00001 and .001. Values closer 0.001 seemed to improve the accuracy of the final model.
* Word vector dimensionality: More features result in longer runtimes, and often, but not always, result in better models. Reasonable values can be in the tens to hundreds; we used 5000.
* Context / window size: How many words of context should the training algorithm take into account? 10 seems to work well.
* Worker threads: Number of parallel processes to run. This is computer-specific, but between 4 and 6 should work on most systems.
* Minimum word count: This helps limit the size of the vocabulary to meaningful words. Any word that does not occur at least this many times across all documents is ignored. 10 and 40 worked well.

The doc2vec API has very similar features.

model\_dm = Doc2Vec(min\_count=min\_word\_count, window=context, size=num\_features, sample=downsampling, workers=num\_workers)

Once the model is trained, we can use some built-in functions.

model.doesnt\_match("man woman child kitchen".split()) kitchen

model.most\_similar("queen") [(u'princess', 0.6588068604469299), (u'bride', 0.6255064010620117), (u'stepmother', 0.6235610246658325),

**Using the word2vec model**

Once the word2vec model was trained, we tried the following approaches before feeding it into the machine learning classifiers.

* WordVector Averaging - Average all the word vectors of a review to derive the sentiment
* TF-IDF Vectorizer on the Word Vectors instead of averaging
* Word Clustering - Have n clusters such that each cluster has 5-7 semantically related words gave good results. This is also referred to as bag of centroids model [14]

**System requirements**

We ran experiments on a google cloud platform VM instance with 8CPUs and 30GB memory. The challenge was to train the word2vec and doc2vec models. Word2vec relies on multi-threading so we need to make sure the C libraries are loaded correctly when genism is installed. This can lead to training the model with features=5000 in hours compared to days. There are also dependencies between genism and scipy libraries to ensure this works correctly.

**Performance Metric**

We used AUC – Area under Curve for comparing the results from the different approaches. It is a plot of true positives vs false positives. We used the Kaggle evaluation metric as the test set that was given to participants didn’t have the sentiment label. We also did cross-validation on the training dataset on a couple of approaches to correlate the AUC scores.

**Results**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Model | Classifier | Vocabulary | Max\_features | Time(s) | AUC |
| Bag Of Words - Count Vectorizer | Random Forest | 5000 | 5000 |  | 0.92307 |
| Bag Of Words - TFIDF | SGDClassifier |  | 200000 |  | 0.96817 |
| Bag Of Words - TFIDF | Multinomial Naïve Bayes |  | 200000 |  | 0.9523 |
| **Bag Of Words - TFIDF (n-gram 1-4)** | **SGDClassifier + Multinomial Naïve Bayes** |  | **200000** | **711.84773612(tf-idf vectorizer)** | **0.96907** |
| Word2Vec Average Vectors | Random Forest | 16490 | 5000 | 1540.38(train word2vec) + 69.01137(vector averaging) | 0.91772 |
| Word2Vec TF-IDF | SGDClassifier | 16490 | 5000 |  | 0.95384 |
| **Word2Vec TF-IDF (Skip Gram, CBOW Model)** | **SGDClassifier + Multinomial Naïve Bayes** | **16490** | **5000** |  | **0.95608** |
| Word2Vec Bag of Centroids | K Means for Word clustering, Random Forest | 16490 | 5000 |  | 0.92195 |
| Doc2Vec Average Vectors (Distributed Memory, Distributed Bag Of Words Model) with 10min words | Random Forest | 38664 | 5000 | 501s\*10epochs(train doc2vec) | 0.81412 |
| **Doc2Vec Average Vectors (Distributed Memory) stacked with BOW, 40 min words** | **Logistic regression** | **19271** | **5000 for doc2vec, 50000 for BOW** | **501s\*5epochs** | **0.96355** |

Figure - Experimental results for bag of words, word2vec and doc2vec models.

The Bag of Words with TF-IDF weighting performs surprisingly well on the movie reviews – effect of varying Num\_features is also well understood. The AUC increases as the num\_features increases however if too large, has very slow improvement. If num\_features are too high, then dimensions can be reduced using chi transform. In this configuration, n-gram range was also varied. Best case AUC of 0.96907 was achieved here for 200000 max\_features.

The word2vec average vectors and bag of centroids model performed very similar, however the bag of centroids model took a very long time for K-means clustering. Here we experiment varying num\_clusters and AUC went down if having less clusters and if clusters are increased then run time increases.

The word2vec model with TF-IDF vectorizer performed very similar to the bag of words model with TFIDFvectorizer for the same feature size. The unlabeled training data was also used to train the word2vec model due to the unsupervised learning nature being its biggest advantage. Feature size could also be increased, however we limited this due to time and space limitations. A AUC score of 0.956 was achieved for an ensemble of Naïve Bayes and SGD Classifiers with probabilities of 0.2 and 0.8 respectively. The word2vec model has an initial training time of around 1540 s, but once trained, the time is similar to that of a bag of words model.

The doc2vec model didn’t perform very well stand alone. This is due to a very nascent genism implementation that we had lot of trouble to get it working. The algorithm also trains over epochs and each epoch took around 500s to finish. Since the paragraph vector approach also needs word vectors, there is added time of generating word vectors and the paragraph vector. The space requirements are also very high. A sample snapshot of the space requirements for word2vec and doc2vec models are below

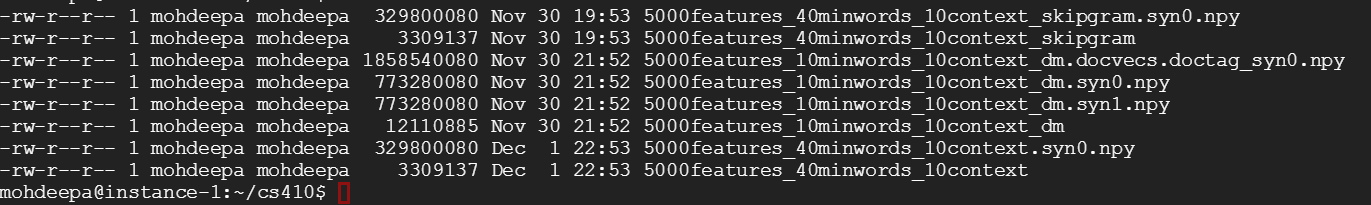
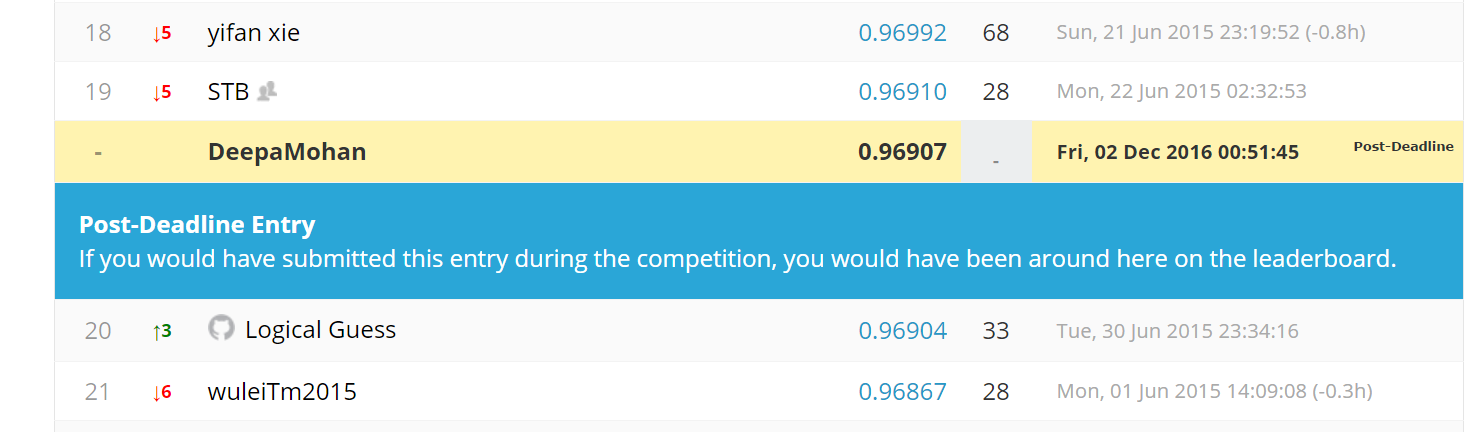


Figure Space requirements

When doc2vec was stacked using np.hstack with the bag of the words model, the hybrid approach did surprisingly well. A AUC of 0.96355 was achieved which was higher than the stand-alone bag of words model for a similar feature set.



**Conclusion**

In terms of time and computational complexity Doc2Vec is the costliest, Word2Vec training time is quite fast because of multi-threading, but once trained, time to tune parameters is similar to BOW. To summarize a high AUC score was achieved when the word2vec and doc2vec techniques were trained with unlabeled and labeled data and the resulting trained model with tf-idf weighting was stacked with a bag of words model and then fed to ensemble machine learning classifiers.

**Future Work**

There are more things we can do to improve the performance results. One thing we could try was using the trained word2vec model with a deep learning network as large datasets which are not ideal for traditional classifiers achieve high results with a deep network. We could also try using a combination word2vec, doc2vec model with different probabilities. Due to system limitations, we could experiment doc2vec with large feature size and is definitely worth experimenting with the distributed memory and the distributed bag of words model. Punctuations were removed from the dataset in preprocessing and it will be great to see how punctuations can infer better polarities from a review. There have also been recent techniques like FastText introduced by Facebook and it will be interesting to see how it does compared to Word2Vec.

**References**

[3] http://www.cs.cornell.edu/home/llee/omsa/omsa.pdf

[4] https://en.wikipedia.org/wiki/N-gram

[5] <http://www.ijser.org/paper/Twitter-Sentiment-Analysis-A-Review.html>

[6] https://www.udacity.com/course/deep-learning--ud730

[7] <https://www.tensorflow.org/versions/r0.12/tutorials/word2vec/index.html>

[8] <http://www-personal.umich.edu/~ronxin/pdf/w2vexp.pdf>

[9] <https://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf>

[10] <http://cs.stanford.edu/~quocle/paragraph_vector.pdf>

[11] <https://code.google.com/archive/p/word2vec/>

[12] <https://radimrehurek.com/gensim/models/word2vec.html>

[13] <http://nlp.stanford.edu/sentiment/>

[14] https://www.kaggle.com/c/word2vec-nlp-tutorial/details/part-2-word-vectors