Web Search Engine Final Project

Weilun Du

May 7, 2015

1 Project Summary

"Bags of Words Meet Bags of Popcorn" is a Kaggle competition which is a sentiment analysis challenge about binary classification on movie reviews. The problem setting is: The labeled data set consists of 50,000 IMDB movie reviews, specially selected for sentiment analysis. The sentiment of reviews is binary, meaning the IMDB rating < 5 results in a sentiment score of 0, and rating ≥ 7 have a sentiment score of 1. No individual movie has more than 30 reviews. The 25,000 review labeled training set does not include any of the same movies as the 25,000 review test set. In addition, there are another 50,000 IMDB reviews provided without any rating labels. Also, none of the reviews in the test set are given any sentiment label. Thus, evaluation can only be given by the judge running on Kaggle's website. More information can be found at link `https://www.kaggle.com/c/word2vec-nlp-tutorial`.

This project is a mixture of research and implementation. I will show how I derive my approach to the challenge step-by-step by studying the latest research paper. Also, besides using the start-of-art library to train the model, I modified the current Machine Learning library to experiment with new ideas.

Overall, I approached the challenge with two steps. First, feature engineering/learning with the corpus. Second, train the learned features with classical machine learning predictor such as SVM, SGDLogistic Regression. Most of my work and research is about how to learn better features given the corpus. The deliverable of the project is that I achieved 85.7% accuracy in the competition and it can be verified at link `https://www.kaggle.com/c/word2vec-nlp-tutorial/leaderboard` by searching up my name "Weilun

Du". Besides training a model, I have experimented with the gensim which is a open source Python library that has an implementation for the "word2vec" algorithm and "doc2vec" algorithm. I throughly studied their code and tried slightly different implementations. Last but not least, I experimented with multi-tasking learning feature in the "doc2vec" algorithm but it didn't produce good results. I have built a very basic web application that can receive query and return the classification online. More details will be available at Evaluation section.

In this project, I am heavily relying on/owing thanks to open source library. I used Python and the following packages are used for the purposes of preprocessing, machine learning and web development: BeatifulSoup, Cython, gensim, sklearn, tornado.

2 Motivation

Conventional language modelling approaches such as "bag-of-words" and "n-gram" have their limitations when it comes to sentiment analysis. In "bag-of-words" , to represent a given document, we first build a dictionary of words given the corpus, then count the frequency of each word in the document. "bag-of-words" assume that each word in the document occurs independently. In the task of sentiment classification, we often see negative movie review with a lot of positive words. Thus, the approach where the algorithm learns some weights for each word and simply averages the words in a document won't produce very promising results. "N-gram" is another approach which is very similar to "shingles" we covered in class. When n is relatively big, the model will capture the contextual information but it tends to generalize very poorly. The intuition is that, given a very large dictionary of words of size V and some fixed n, there are potentially V^n possible features. So when n gets bigger, either many features from our corpus will never be used again in test set, or features from test set are never seen in our corpus. There are many smoothing tricks to deal with the unseen features, but in general, the dimensionality of the feature will be enormous, So the n-gram model will suffer the so-called"curse of dimensionality".

It is clear from above that we want to find a vector representation for words such that first, it should consider contextual information or "semantic similarity" and second, it should generalize well, more specifically, we don't want the vector to be mostly filled with zero. "Distributed representation" aims

to achieve both goals and it has been widely studied in natural language processing. In this project, I chose to explore how to learn distributed representation for words using neural models. In paper [1], Bengio et al. proposed that we can learn distributed representation using neural network. To be consistent with the notation, we use w_1^t to represent a sequence of words $(w_1, w_2, w_3, \cdots w_t)$ Without loss of generality, we can represent the document with words w using conditional probability

$$P(w_1^T) = \prod_{t=1}^{T} P(w_t|w_1^{t-1})$$

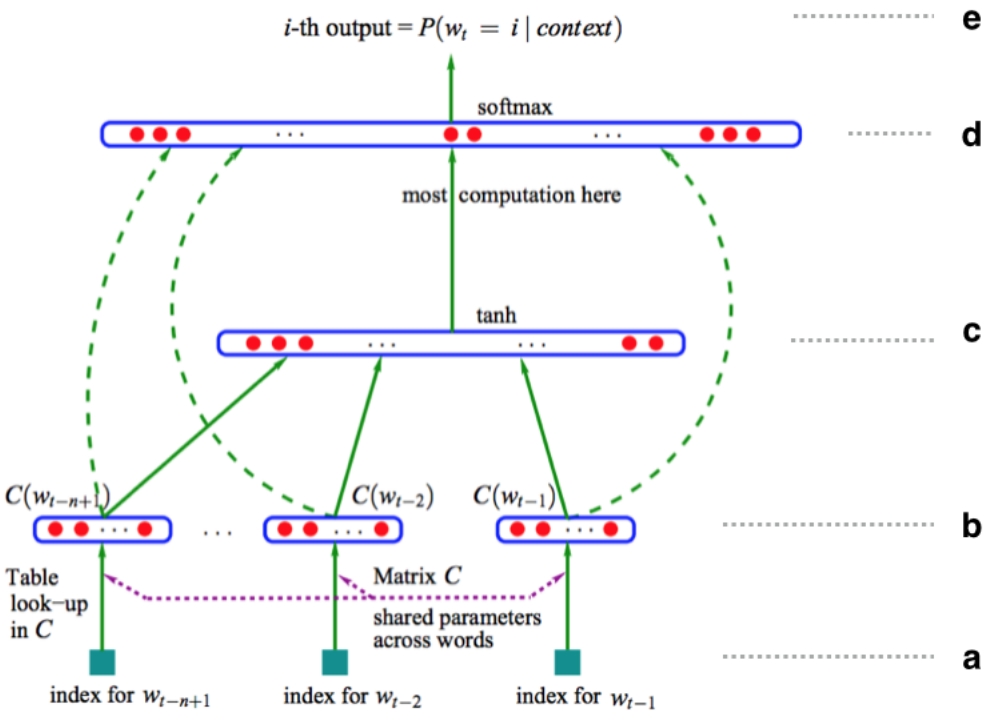
where T is the size of the document. The probability above is hard to estimate and often times, people considers the previous $n-1$ words. n-gram takes this approach and use Maximum Likelihood Estimation to approximate the underlying distribution with different smoothing techniques.

A neural model takes a different approach and approximate the underlying distribution by building a neural network. In figure 1, essentially, the model has three layers: input layer, hidden layer, and output layer. The output of the previous layer serves as the input to the next layer. So each word in a document is projected to a word vector R_k where k is the dimensionality of the distributed representation. The goal is to approximate the probability of $P(w_t|w_1^{t-1})$ which is equivalent to maximize the $\hat{P}(w_t|w_1^{t-1})$ given our corpus. A simplifying view would be: In the input layer, for the current word w_t, we consider the context of word w which is w_{t-n+1}^{t-1}, and concatenate each of the word vector in the context to obtain vector x then feed x into the hidden layer. The hidden layer consists of hidden weights and it uses $tanh$ function to achieve non-linearity. It can be represented as

$$y = b + W * x + U * tanh(H * x + d)$$

where vector y is the output of the hidden layers and other parameters are weights to be learned in the model. Next, we feed the output y into the output layer which is a Softmax function for multi-class classification. This non-convex optimization problem is tackled with "stochastic gradient method" and "backprop". Without getting into the details of the math, there are few limitations in practice when using this model. First, the output layer is computationally intensive because the Softmax function has to be normalized, which means it will compute the probability for each possible outcome(here in our case, each possible word) so as to make sure the sum adds to one. In a dictionary of size V, the output layer has O(V) running

Figure 1: A nueral architecture proposed in [1].

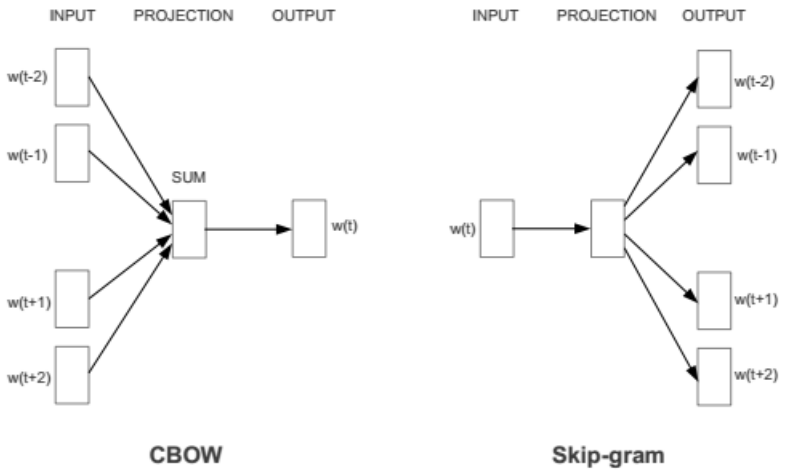


time and V is at least 100,000 in our case. Second, the hidden weights are also enormous because the input layer concatenates all the word vectors. To redeuce the training time for the neural model. In paper [2] and [3], Mikolov et al. proposed more efficient neural models that utilizes "Hierarchical Softmax" which greatly reduces the running time of output layer. More details will be discussed in the next section.

3 Algorithm

Word2Vec has two probabilistic models: Continuous Bag of Words and Skip-gram. In CBOW, we try to infer the probability of a current word w_t using the context of the word which is w_{t-n}^{t+n} where n is the window size. Given some corpus C, to maximize the product of probability in a given corpus is

Figure 2: Two Probabilistic Models in Word2Vec [2][3].



equivalent to the summation of the log probability, thus we have

$$\sum_{w_t \in C} log(P(w_t|w_{t-n}^{t+n})) \tag{1}$$

While in Skip-gram, given the current word w_t, we want to infer its context w_{t-n}^{t+n}. Skip-gram assumes that the words that appears in the current context are conditionally independent. Thus, we have

$$\sum_{w_t \in C} \sum_{w_i \in w_{t-n}^{t+n}} log(P(w_i|w_t)) \tag{2}$$

Essentially, Equations (1) and (2) are the objective function that we try to maximize in CBOW and Skip-gram respectively. In the input layer, each word is mapped to some word vector with k dimension. For the projection layer, in CBOW, the model simply averages the word vector of each word in context which will somehow lose information about word order. In Skip-gram, the model simply take the identity. The true ingenuity lies at how it approximate conditional probability P in the output layer. The goal here is to not only have a good estimation but also achieve time-efficiency. Hkierarchical-Softmax utilizes Huffman encoding to speed up the running time from linear to log. The idea is given a dictionary of words with frequencies, Huffman encoding will assign shorter codes to more frequent words and longer codes to less frequent words. This encoding scheme is equivalent to

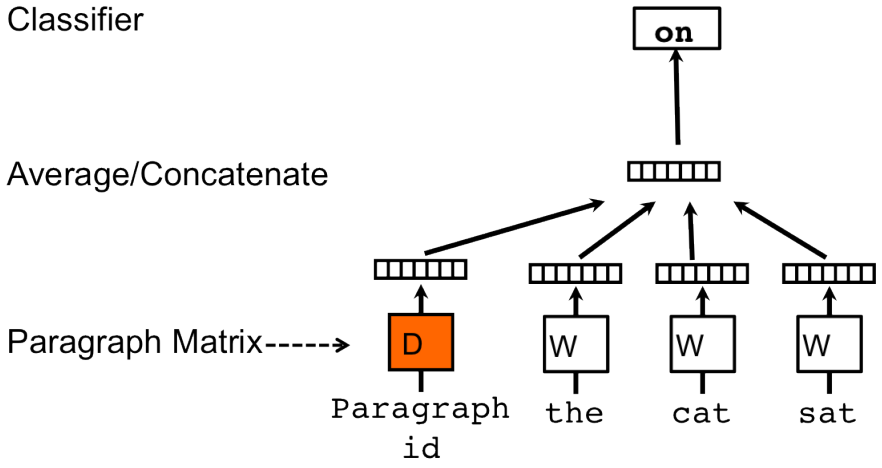
a Huffman tree where each leaf represents a word and we assign each internal node a vector θ to represent hidden weights. A path from root to leaf represents some encoding such that in each edge in path represent number 1 or 0. In word2vec, 0 is right and 1 is left. In the output layer, given the input vector p from previous projection layer, $L(w_t)$ as the path from root to leaf node of word w_t, and d as the Huffman encoding for given word w_t, we have

$$P(w_t|p) = \prod_{j \in L(w_t)} (1 - d_j) * \sigma(\theta_j^T * p)) * d_j * (1 - \sigma(\theta_j^T * p)) \tag{3}$$

where $\sigma(x)$ is the sigmoid function is used in logistic regression for binary classification. It is clear that the probability can be computed within O(log(V)) for a given dictionary of size V. One question the reader might raise is whether $\sum_{w_t \in C} P(w_t|w_i) = 1$ holds true, it can be proved by using induction combined with the property that Huffman tree is a full binary tree, but I will skip the proof. If we plug Equation(3) back into (1) and (2). We can easily derive the objective functions of CBOW and Skip-gram. To optimize the non-convex function in Equation (3), the gensim library uses "stochastic gradient method" and "backprop". I will also skip the derivation of the first order differentials for each parameter but suppose we have word vector with k dimension, we need to learn word vectors matrix $W_{V \times k}$ and hidden weights matrix $H_{V-1 \times k}$. I would not claim that I have really understood the theory behind "stochastic gradient method" and "backprop" as most tutorials are more focused on how to achieve better results by tuning learning rate and scale input data and other techniques. There are also a few details worth noting in the word2vec model. First, we must build the Huffman tree before training the model, if there are unseen words from other test set, word2vec will simply ignore those words. Second, the initial values of the word vectors are generated by random function with deterministic seed, they are values from -0.5 to 0.5.

However, to do useful things with movie reviews, we can't just simply average the word vector in a given movie review as it would just defeat our goal of extracting contextual information. In paper [4], Quoc Le et al. proposed very similar extensions to train document vector to their counterpart in training word vector. The counterpart to CBOW is Distributed Memory, where we treat the document vector D for a given document as a shared memory across the paragraph, while the word vector is shared globally across the corpus. In the projection layer, we can simply either concatenate or average the word vectors in context with the document vector. Thus, the

Figure 3: Distributed Memory Model in Doc2Vec [4].



probability model can represented as

$$\sum_{w_t \in C} log(P(w_t|D_t, w_{t-n}^{t+n})) \tag{4}$$

with the rest of model stays the same. The counterpart to Skip-gram is Distributed Bag of Words where in the projection layer, we try to infer a random word from the current paragraph using the current document vector D.

$$\sum_{D_t \in C} \sum_{w_i \in paragraph} log(P(w_i|D_t)) \tag{5}$$

Both probabilistic models would require that we iterate through the entire the corpus multiple times so as to obtain better approximation of the doc vector. One key difference between doc2vec and word2vec is that we need to construct an identifier for each document but doc vectos should not be taken into account when building a Huffman tree. In the Experiment section, I will explain why the open-source implementation is inconsistent at this point.

So far the business is just about finding better distributed representation to capture semantic similarity. In paper [5], Andrew L. Maas et al. proposed a multi-task learning scheme T capture both semantic similarity and sentiment similarity. The idea is to adjust the objective function such that in paper [5], it maximize the function for "Latent Dirichlet Allocation" which

is a different model for finding distributed representation and a logistic regression for sentiment classification. I have briefly modified the model to incorporate the same idea into doc2vec. Using the example for Distributed Memory model, given sentiment s for given ducoment D and let s = 1 for positive sentiment. We want to optimize

$$\sum_{w_t \in C} log(P(w_t|D_t, w_{t-n}^{t+n})) + \sum_{D_t \in C} s_t * log(\sigma(\theta^T * p + b)) + (1 - s_t) * (1 - log(\sigma(\theta^T * p + b)) \tag{6}$$

where θ and b are the weights for the logistic regression function and they should never be mixed with hidden weights in "Hierarchical Softmax". As usual, we will learn these parameters using the same techniques. One solid questioning about all the objective function here is that none of them regularize the learned weights and certainly it runs of the risk of overfitting.

4 Experiment

Training a neural model is more or less an art then science. I followed the Experiment section of paper [4] to train doc2vec model using both DBOW and Distributed Memory models. More details steps are documented in my code but here are the major steps: 1, Build vocabulary using all data. 2, train unlabelled data. 3, train labeled data in training set. 4, train test set's doc vectors but fixing the word vectors and hidden weights. 5, extract both DBOW and Distributed Memory's doc vecs for training set and test set and concatenate them to obtain the feature. Feed training set's features into a classical linear classifier. I achieved best results with SGD LogisticRegression model. 5, Feed doc vecs of the test set into the classifier. There are a few hyper-parameter worth mentioning: first, the feature dimensionality is chosen to be 400 because doc vector should contain richer information. Second, window size for Distributed Memory is chosen to be 10 and during training, the actual window size is chosen randomly from 1 to 10. Third, I trained 10 epochs with random permutation of each data set. Fourth, learning rate is adjusted according to how much data has been trained with respect to total words. One key difference between my training model and paper [4] is that in Distributed Memory, [4] concatenate the word vector and document vector as the context so as to infer on the outcome in the projection layer, while I simply averages them because of the limitation of the gensim's implementation of doc2vec module. Given the 100,000 movie reviews, our dictionary contains roughly 98,000 words that occurs at least 3 times. In the competition, we are actually allowed to use trained word vec-

tors from Google with roughly 1 million words. I assume that if the model is fed with more data, the word vectors should generalize better.

The doc2vec implementation in gensim is by no means a stable version. I have tried to improved over two places: 1, in the original implementation, the doc2vec object extends word2vec object and build the document label into the Huffman tree. This is certainly inconsistent with the design contract and how the probabilistic model works because in both DBOW and Distributed Memory models, we never try to infer the document label as the outcome. Thus, the total sum of all probability might not add to one! To change this, I overwrite the train() method and hacked their cython code to achieve the result. 2, In order to train online for a single unseen test, I relaxed the condition of the models, and simply skip the unseen words from our vocabulary. This part is relatively easy and mostly of the codes are adapted from word2vec's python implementation for the "backprop".

To train the model with multi-task learning, I need to modify the core algorithm to incorporate the sentiment classification objectives. However, the core algorithm that is used in training is written in highly optimzed Cython code and I was having trouble get it work. I modified the python implementation but it is roughly 100 times slower then Cython implementation. Thus, it is much difficult to have meaningful comparison using the same amount of data because training using Cython takes roughly 30 minutes.

5 Conclusions

In this final project, I have explored new possibilities to train distributed representation for documents. As we show in the Algorithm section, neural model will be able to capture the "semantic similarity" between words and documentations. Even though I failed to get the multi-task learning algorithm to work as expected as paper [5], I think it is potentially a good direction for learning richer information in distributed representation. Gensim as a open source library is great for building my future work and there could have been a lot more improvement to make the doc2vec module better.

references

[1] Yoshua Bengio, Rejean Ducharme, Pascal Vincent, and Christian Janvin. A neural probabilistic language model. The Journal of Machine Learning

Research, 3:1137?1155, 2003.
[2] Mikolov, Tomas, Sutskever, Ilya, Chen, Kai, Corrado, Greg, and Dean, Jeffrey. Distributed representations of phrases and their compositionality. In Advances on Neural Information Processing Systems, 2013c.
[3] Mikolov, Tomas, Chen, Kai, Corrado, Greg, and Dean, Jeffrey. Efficient estimation of word representations in vec- tor space. arXiv preprint arXiv:1301.3781, 2013a.
[4] Quoc Le and Tomas Mikolov. Distributed Representations of Sentences and. Documents. (ICML 2014).
[5] Andrew L. Maas, Raymond E. Daly, Pleaseer T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. (2011). "Learning Word Vectors for Sentiment Analysis." The 49th Annual Meeting of the Association for Computational Linguistics (ACL 2011).