Analysis of Yelp Dataset for text mining of user Reviews and Rating analysis

Shrikumar Patil - patil.shr@husky.neu.edu
Akshay Shinde - shinde.ak@husky.neu.edu
Information Systems, College of Engineering, Northeastern University - Boston

Abstract

Customer feedback is an essential source of information for improving operations in the service industry, but capturing an accurate and complete picture of the customer experience has always been a challenging task. With the recent rise of social networks and reviews posted on websites and portals, we have access to a large and rapidly growing number of online customer reviews, but most of those reviews consist of unstructured comments that are not amenable to direct analysis using traditional methods. We intend to do text mining and information extraction to analyze these reviews along with the ratings to derive important insights into the collective sentiments of the reviews, performing text classification, and sentiment analysis of a review. Also, recommender systems may be enhanced for customers depending on their positive and negative reviews. Here, we aim to classify the text reviews into positive review and negative review just by analyzing the review text, and testing it against the actual star rating given with that review.

Introduction

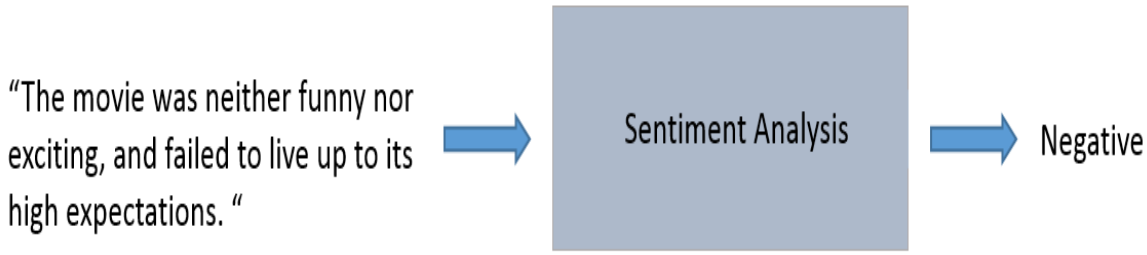
Most text analysis research to date has been on well-formed text documents. Researchers try to discover and thread together topically related material in these documents. However short text mediums are becoming popular in today's societies and provide quite useful information about people's current interests, habits, social behavior and tendencies. Large communities like Yelp are based on user generated content (UGC). Most of the time, the contributions from the users is in the form of short text contributions. Unlike traditional web documents, these text and web segments are usually noisier, less topic–focused, and much shorter. They vary from a dozen words to a few sentences. As a result, it is a challenge to achieve desired accuracy due to the data sparseness.

Sentiment analysis is often used by companies to quantify general social media opinion (for example, using tweets about several brands to compare customer satisfaction). One of the simplest and most common sentiment analysis methods is to classify words as "positive" or "negative", then to average the values of each word to categorize the entire document. We intend to analyze if this approach actually produces good results, can we predict the positivity or negativity of someone's writing by counting words or by understanding the context of the review and understanding rating dimensions using review text.

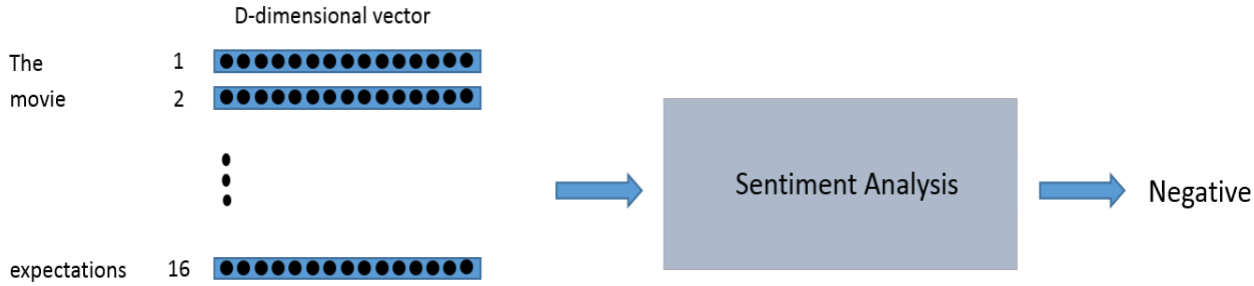
Materials and Methods

The main objective here for us to use and examine the results of various machine learning algorithms and compare it with deep learning method, using LSTM's. We use LSTM here rather than using simple recurrent layers because LSTM's will better preserve the context of the text reviews which will be fed as inputs to the network and assess which perform the best for text classifications. Let us now discuss some background of both LSTM and word vectors.

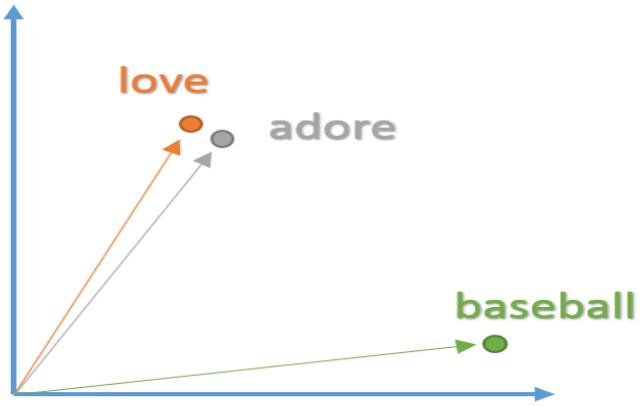
In order to understand how deep learning can be applied, think about all the different forms of data that are used as inputs into machine learning or deep learning models. Convolutional neural networks use arrays of pixel values, logistic regression uses quantifiable features, and reinforcement learning models use reward signals. The common theme is that the inputs need to be scalar values, or matrices of scalar values. When you think of NLP tasks, however, a data pipeline like this may come to mind.



This kind of pipeline is problematic. There is no way for us to do common operations like dot products or backpropagation on a single string. Instead of having a string input, we will need to convert each word in the sentence to a vector.



You can think of the input to the sentiment analysis module as being a 16 x D dimensional matrix.
We want these vectors to be created in such a way that they somehow represent the word and its context, meaning, and semantics. For example, we'd like the vectors for the words "love" and "adore" to reside in relatively the same area in the vector space since they both have similar definitions and are both used in similar contexts. The vector representation of a word is also known as a word embedding.



Word2Vec

In order to create these word embeddings, we'll use a model that's commonly reffered to as "Word2Vec". Without going into too much detail, the model creates word vectors by looking at the context with which words appear in sentences. Words with similar contexts will be placed close together in the vector space. In natural language, the context of words can be very important when trying to determine their meanings. Taking our previous example of the words "adore" and "love", consider the types of sentences we'd find these words in.

I love taking long walks on the beach.
My friends told me that they love popcorn.
⋮
The relatives adore the baby's cute face.
I adore his sense of humor.

From the context of the sentences, we can see that both words are generally used in sentences with positive connotations and generally precede nouns or noun phrases. This is an indication that both words have something in common and can possibly be synonyms. Context is also very important when considering grammatical structure in sentences. Most sentences will follow traditional paradigms of having verbs follow nouns, adjectives precede nouns, and so on. For this reason, the model is more likely to position nouns in the same general area as other nouns. The model takes in a large dataset of sentences (English Wikipedia for example) and outputs vectors for each unique word in the corpus. The output of a Word2Vec model is called an embedding matrix.

This embedding matrix will contain vectors for every distinct word in the training corpus. Traditionally, embedding matrices can contain over 3 million word vectors.

The Word2Vec model is trained by taking each sentence in the dataset, sliding a window of fixed size over it, and trying to predict the center word of the window, given the other words. Using a loss function and optimization procedure, the model generates vectors for each unique word. The specifics of this training procedure can get a little complicated, so we're going to skip over the details for now, but the main

takeaway here is that inputs into any Deep Learning approach to an NLP task will likely have word vectors as input.
For more information on the theory behind Word2Vec and how you create your own embeddings, check out Tensorflow's tutorial

To understand LSTMs, let us first quickly understand the background of Recurrent Neural Networks. As we are dealing with text data, the context of the sentence is very important. The things said earlier in a review, will still make sense and hold true to provide strong context or basis for the text further in the review. RNN's are such networks which have the ability to persist information, because they have a loop and a feedback mechanism within them. Each word in a sentence depends greatly on what came before and comes after it. In order to account for this dependency, we use a recurrent neural network. However, traditional RNN's have a problem of long term dependency.

Let's look at the following example.

Passage: "The first number is 3. The dog ran in the backyard. The second number is 4."

Question: "What is the sum of the 2 numbers?"

Here, we see that the middle sentence had no impact on the question that was asked. However, there is a strong connection between the first and third sentences. With a classic RNN, the hidden state vector at the end of the network might have stored more information about the dog sentence than about the first sentence about the number. Basically, the addition of LSTM units make it possible to determine the correct and useful information that needs to be stored in the hidden state vector.
Again, for example, if we are trying to predict the last word of a sentence, and we have some useful appropriate context in the same sentence, then maybe it's fine, but if you have to predict the last word of this sentence according to the information and context from a sentence or two before that sentence, then RNN's won't be of much help as they do not retain long term information. This problem of RNN's is solved by LSTM's: Long Short Term Memory networks.
Long Short Term Memory networks – usually just called "LSTMs" – are a special kind of RNN, capable of learning long-term dependencies. They work tremendously well on a large variety of problems, and are now widely used.
Long Short Term Memory Units are modules that you can place inside of recurrent neural networks. At a high level, they make sure that the hidden state vector h is able to encapsulate information about long term dependencies in the text.

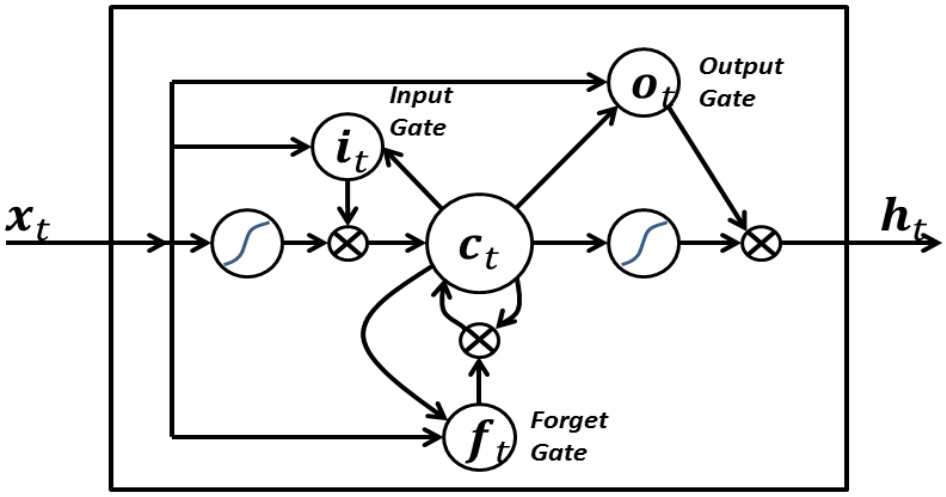
In RNNs, each word in an input sequence will be associated with a specific time step. In effect, the number of time steps will be equal to the max sequence length. RNN hidden state vector equation:

$$h_t = \sigma(W^H h_{t-1} + W^X x_t)$$

Associated with each time step is also a new component called a hidden state vector h_t. From a high level, this vector seeks to encapsulate and summarize all of the information that was seen in the previous time steps. Just like x_t is a vector that encapsulates all the information of a specific word, h_t is a vector that summarizes information from previous time steps. The hidden state is a function of both the current word vector and the hidden state vector at the previous time step. The sigma indicates that the sum of the two terms will be put through an activation function (normally a sigmoid or tanh).

Equations for LSTM:

$i(t) = \sigma(W(i)\, x(t) + U(i)\, h(t-1))$ (Input gate)
$f(t) = \sigma(W(f)\, x(t) + U(f)\, h(t-1))$ (Forget gate)
$o(t) = \sigma(W(o)\, x(t) + U(o)\, h(t-1))$ (Output/Exposure gate)
$\tilde{c}(t) = \tanh(W(c)\, x(t) + U(c)\, h(t-1))$ (New memory cell)
$c(t) = f(t) \circ \tilde{c}(t-1) + i(t) \circ \tilde{c}(t)$ (Final memory cell)
$h(t) = o(t) \circ \tanh(c(t))$



Looking at LSTM units from a more technical viewpoint, the units take in the current word vector x_t and output the hidden state vector h_t. In these units, the formulation for h_t will be a bit more complex than that in a typical RNN. The computation is broken up into 4 components, an input gate, a forget gate, an output gate, and a new memory container.

Each gate will take in x_t and h_{t-1} (not shown in image) as inputs and will perform some computation on them to obtain intermediate states. Each intermediate state gets fed into different pipelines and eventually the information is aggregated to form h_t. For simplicity sake, we won't go into the specific

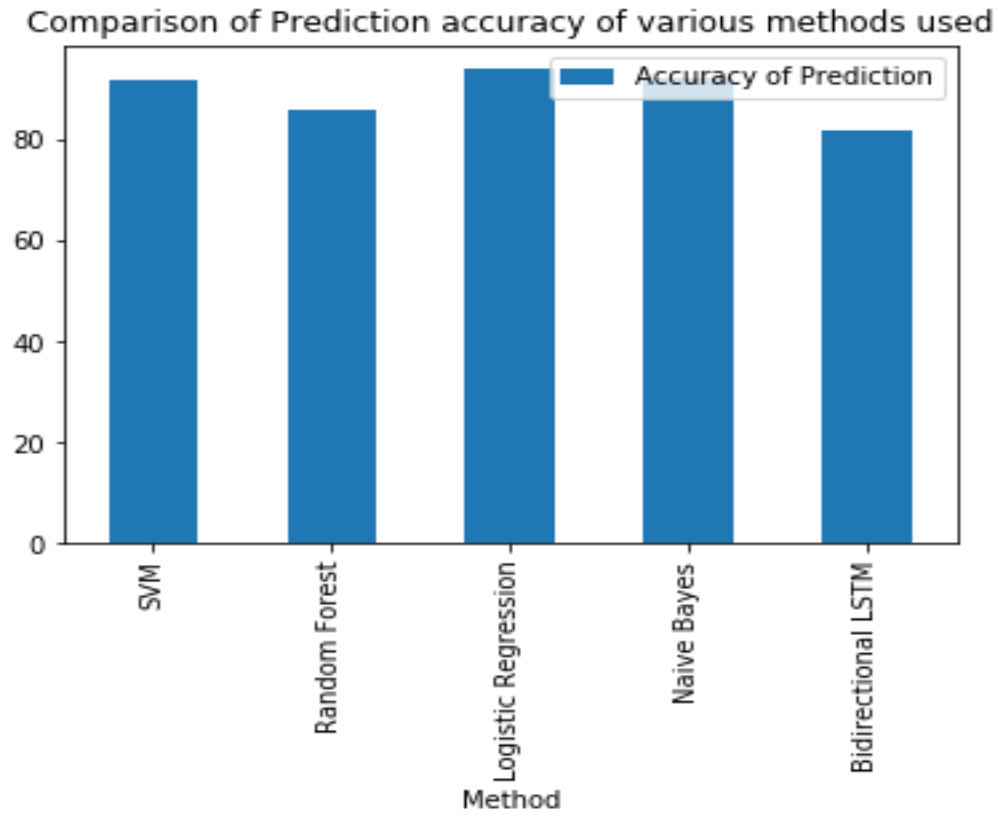
formulations for each gate, but it's worth noting that each of these gates can be thought of as different modules within the LSTM that each have different functions. The input gate determines how much emphasis to put on each of the inputs, the forget gate determines the information that we'll throw away, and the output gate determines the final h_t based on the intermediate states.

Also, we use the concept of item based collaborative filtering. Item-item collaborative filtering, or item-based, or item-to-item, is a form of collaborative filtering for recommender systems based on the similarity between items calculated using people's ratings of those items.

First, the system executes a model-building stage by finding the similarity between all pairs of items. This similarity function can take many forms, such as correlation between ratings or cosine of those rating vectors. As in user-user systems, similarity functions can use normalized ratings (correcting, for instance, for each user's average rating).
Second, the system executes a recommendation stage. It uses the most similar items to a user's already-rated items to generate a list of recommendations. Usually this calculation is a weighted sum or linear regression. This form of recommendation is analogous to "people who rate item X highly, like you, also tend to rate item Y highly, and you haven't rated item Y yet, so you should try it".

Results & Discussion

Code: https://github.com/shrikumarp/ADSSpring18/blob/master/ADSProject_Portfolio.ipynb



We perform basic exploratory data analysis on the yelp dataset, with graphs of review counts and text length distribution etc. Further, we start the process of processing the steps. We prepare the text data to feed to machine learning algorithms and neural networks as we cannot feed text values directly. We first start with tokenization, then we perform count vectorization. Then we split the dataset of generated count vectors from the text reviews and corresponding star rating into training and testing set. What we aim to do here is to train and find the algorithm with maximum accuracy, which can classify a test review correctly into a positive review class or a negative review class. We compared many Algorithms, and also improved performance of some of them using hyper-parameter tuning. The most accurate out of all the algorithms was Logistic Regression after parameter tuning with accuracy of 94%. Bidirectional LSTM neural networks classified the reviews into positive or negative review. The LSTM model performed with accuracy of 81.95.

Method	Accuracy of Prediction
SVM	91.57
Random Forest	86.00
Logistic Regression	94.00
Naive Bayes	92.00
Bidirectional LSTM	81.95

As with all the neural networks, we may get better results with LSTM by training with more variety of data, and are most likely predict better than all other machine learning algorithms with intensive training. However, from our findings for this dataset, logistic regression worked best for text classification based on the corresponding reviews given by the users who wrote those reviews, with accuracy of classification coming upto 94%. This means that just based on the text reviews, our models were able to classify the review into good and bad sentiment, and their accuracy was tested against actual star ratings corresponding to those reviews.

In the last part we build a simple recommender system which uses item based
collaborative filtering that takes into consideration the similarity of restaurants by the user ratings they received and using this as the basis for correlating them and then scaling according to the user reviews to recommend new similar restaurants to the user.

REFERENCES

- https://github.com/adeshpande3/LSTM-Sentiment-Analysis/blob/master/Oriole%20LSTM.ipynb
- https://machinelearningmastery.com/develop-bidirectional-lstm-sequence-classification-python-keras/
- http://colah.github.io/posts/2015-08-Understanding-LSTMs/
- https://link.springer.com/content/pdf/10.1007/978-81-322-1602-5_75.pdf

Namita Mittal: Text Classification Using Machine Learning Methods-A Survey

- https://arxiv.org/pdf/1402.4380.pdf
  Samuel Danso , Eric Atwell and Owen Johnson
  A Comparative Study of Machine Learning Methods for Verbal Autopsy Text Classification

- https://www.cs.cornell.edu/people/tj/publications/joachims_98a.pdf
  Thorsten Joachims
  Text Categorization with Support Vector Machines: Learning with Many Relevant Features

- http://www.cs.utexas.edu/~ml/papers/discotex-melm-03.pdf
  Raymond J. Mooney and Un Yong Nahm
  Text Mining with Information Extraction

- https://scholarship.sha.cornell.edu/cgi/viewcontent.cgi?referer=https://www.google.com/&httpsredir=1&article=1003&context=chrreports
  Hyun Jeong "Spring" Han, Shawn Mankad
  Nagesh Gavirneni
  Rohit Verma
  What Guests Really Think of Your Hotel: Text Analytics of Online Customer Reviews

- http://www.cs.ubc.ca/~nando/540-2013/projects/p9.pdf
  Exploring the Yelp Data Set: Extracting Useful Features with Text Mining and Exploring Regression Techniques for Count Data

- http://www.ics.uci.edu/~vpsaini/files/technical_report.pdf
  Hitesh Sajnani, Vaibhav Saini, Kusum Kumar , Eugenia Gabrielova , Pramit Choudary, Cristina Lopes,
  Classifying Yelp reviews into relevant categories