Sentiment prediction on the Yelp review dataset

Yaoyang Zhang
Department of Civil and Environmental Engineering
University of California, Berkeley
yaoyangzhang@berkeley.edu

Sri Krishna Vadlamani
Department of Electrical Engineering and Computer Sciences
University of California, Berkeley
srikv@berkeley.edu

Abstract

Sentiment analysis of text is very helpful to businesses in evaluating customer experience and spotting dissatisfaction in order to improve their services or products. One way to analyze sentiment of sentences is to train a classifier on a large dataset with sentiment labels. In this work, we trained a variety of neural network models to predict the ratings that users awarded restaurants on Yelp, based only on their review text. A vanilla neural network (VNN) and a Gaussian Naive Bayes (GNB) classifier based on the bag-of-words model were trained as baselines. Then, a Convolutional Neural Network (CNN) and a Recurrent Neural Network (RNN) were trained using the word2vec representation of words. The GNB, VNN, CNN, and RNN achieved validation accuracies of 38.8%, 53%, 56.9% and 54.1% respectively.

1 Introduction

Many e-commerce sites such as Yelp and Amazon allow users to write reviews and ratings for various businesses and they are useful references for other customers. A numerical rating (e.g. from 1 to 5) is usually the most straightforward evaluation for a business (or a merchandise). These evaluations are important to both the businesses and the customers. Businesses can use these ratings to quantify their performance and customers can use them as references. However, many businesses lack such direct evaluations from customers. This problem can be ameliorated by predicting ratings based on relevant reviews scraped from the Internet or social media such as Twitter and Facebook, and the predicted ratings can then be used by such companies to improve their products or services. Such predictions can also be incorporated into recommendation systems based on user reviews.

We tackled this problem by training various neural networks on the Yelp Challenge Dataset (YCD) using Google pre-trained word2vec word vectors. Two different flavors of neural networks were trained to predict ratings based on review text and the results were compared with those of baseline vanilla neural net and Naïve Bayes models. The neural networks were also used to extract special review embeddings that could be employed to find sets of similar reviews.

2 Dataset and preprocessing

We worked on the Yelp Challenge Dataset (YCD), which contains 4.1 million reviews by 1 million users for 144,000 businesses and retained only the reviews for restaurants. However, it was noticed that the dataset was highly skewed towards high ratings. In order to create a balanced training set, equal numbers (2500) of samples were picked from all classes to form a training set of size 10,000. A disjoint set of 4,000 samples was also held out from the original Yelp dataset as the validation set for the sake of model evaluation. Random sampling was performed to form the

validation set, and not equal sampling from all classes, as we wished to test our models on data representative of the actual distribution of reviews in the full YCD.

Only the text was extracted from the dataset and all the punctuations, digits, and stop words were removed before generating vector representations. We used two text representation models, bag-of-words and word2vec, both of which are explained in greater detail in the following section. Due to RAM limitations on our personal laptops, the maximum length of each review was set to 100 in cases where word2vec was used and the shorter reviews were zero-padded. The neural networks were implemented using TensorFlow packages. Our codes were also modeled on TensorFlow and GitHub tutorials [1]-[3].

3 Text representation models

3.1 Word2vec [4]

Word2vec is a natural language representation model which represents words as vectors. These vectors capture the likelihood of occurrence of any two words near each other in a sentence of that language.

In particular, in the skip-gram architecture for word2vec, the softmax function applied on the inner product of the word2vec vector of a word with that of a given center word is interpreted as the probability of the first word appearing in the neighbourhood of the center word. The loss function is then defined to be:

$$L(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{-m\leq j\leq m,\ j\neq 0} \log\left(p(w_{t+j}|w_t)\right)$$

In fact, every word in this architecture has two vector representations, one for cases in which it is the center word, and the other for situations in which it is a 'context' (non-center) word.

Minimizing this loss function over a large corpus of natural language samples yields the word2vec representation for all words in that sample. We used a ready-to-use word2vec representation that was pre-trained by Google on samples from Google News. Each word2vec representation in this set has 300 dimensions. Therefore, each review (with the length limited to 100 words as mentioned above) is represented by a 100×300 matrix in the word2vec representation.

3.2 Bag-of-words

This is a much simpler model which doesn't take word context or order into account. The number of occurrences of each word in all the reviews was compiled and the 500 most commonly occurring words were chosen as the features for each review in our experiments. These vectors were then used for prediction using the Naïve Bayes and VNN models described in Section 4.

4 Classification Models

4.1 Naïve Bayes

A Gaussian Naïve Bayes (GNB) classifier utilizing the bag-of-words features of the reviews was one of our baseline models. A NB classifier is a generative model which assumes no correlations between different features and decision rule is obtained by maximum a posteriori (MAP) estimation.

$$\hat{y} = \underset{k\in\{1,..,K\}}{\arg\max}\ p(C_k)\prod_{i=1}^{n} p(x_i|C_k)$$

4.2 Vanilla Neural Network

The other baseline model that was implemented was a vanilla neural network with an input layer, a ReLU-activated dense hidden layer, and an output layer.

4.3 Convolutional Neural Network

Convolutional Neural Networks (CNNs) were originally designed to be used for computer vision applications. These networks contain hidden convolutional layers, in addition to the dense layers used in the VNN above. CNNs are useful and effective for images because they use filters and extract features that can be used by the downstream densely connected layers for classification. The filters that best do this job are chosen by the algorithm itself via optimization of the loss function. Simple examples of filters include low-pass filters that smoothen images, and high-pass filters that extract edges.

[5] applied the idea of CNNs to sentences represented in the form of a matrix with each row containing the word2vec embedding of a word in the sentence. Convolutional filters with full width but only a certain window along the height of this matrix (we are effectively sliding a window along the words of the sentence) are then used to extract features for the following densely connected layers. We implemented this idea in our work. The following is representative of the CNNs used for text analysis [6].

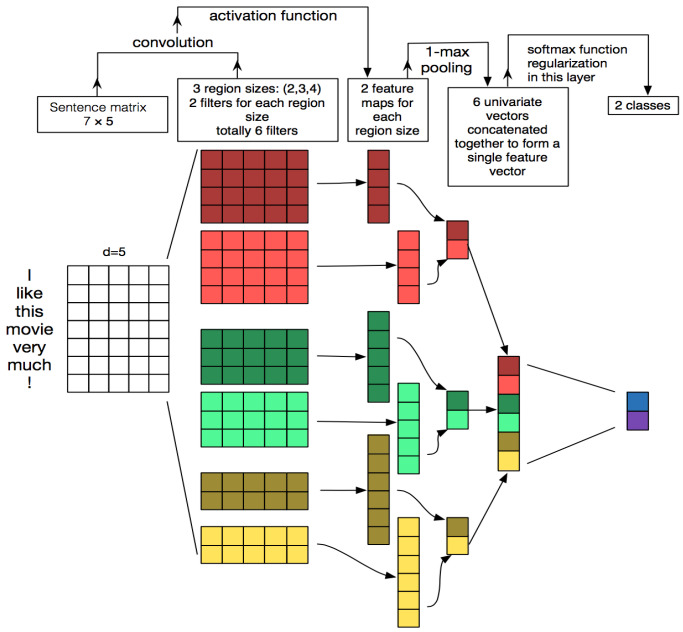


Figure 1: Convolutional Neural Network for Text Analysis

Our CNN had an input layer, one convolutional layer followed by max-pooling, one densely connected hidden layer that takes the max-pooled values as inputs, and the final output layer. The densely connected hidden layer that we had in our model is missing in the generic illustration above.

4.4 Recurrent Neural Network with LSTM

Recurrent neural networks (RNNs) is a class of neural networks that is very popular and effective in modeling sequences. Unlike traditional feed-forward neural networks, RNNs contain directed loops which represent the propagation of activations to future inputs in a sequence. This property allows RNNs to capture the order of each input and thus makes them a natural way to model sequences. An unrolled version of an RNN is shown in Figure 2 [7].

Typical problems with RNNs are the "exploding" and "vanishing gradient" problems. In RNNs, the back-propagated error tends to blow up or vanish, making it impossible to train the network. The exploding gradient problem can be avoid by clipping the gradient at a certain threshold, and several special units were designed to overcome the vanishing gradient problem. Among them, the most popular units are the Gated Recurrent Unit (GRU) and the Long Short-term Memory (LSTM) unit. A typical structure for an LSTM unit in shown in Figure 3 [7].

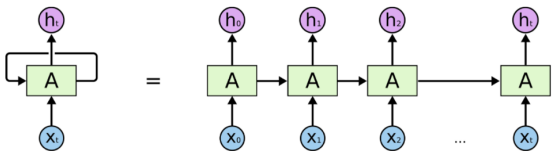


Figure 2: An Unrolled Recurrent Neural Network

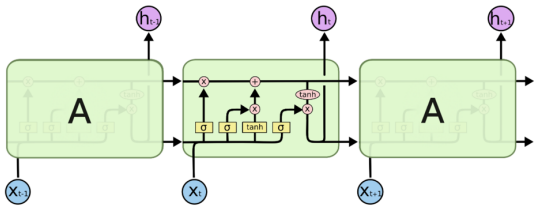


Figure 3: Long Short-Term Memory Unit

We chose to employ LSTM units in our work. A fully connected softmax layer was attached to the outputs of the final LSTM layer. A pre-trained word2vec model with 300 dimensions was used as the feature vector for each word in the input review.

4.5 Review Embeddings using Neural Networks

Just like word2vec models, a review (a collection of sentences) can also be projected into a embedding space where similar review are close to each other. However, unlike word2vec embedding where the embedding is learned with a loss function defined on word pairs, a neural network (e.g. CNN, RNN) trained for rating classification can readily be used for embedding of reviews. The last few layers of CNNs/RNNs usually contain high-level abstractions of the inputs and can thus be interpreted as an embedding of the inputs. We also explored this property on a trained RNN in this work.

5 Parameter Tuning and Results

Model	Training Accuracy	Validation Accuracy
GNB	41.4%	38.8%
VNN	58.6%	53%
CNN	72.9%	56.9%
RNN	58.8%	54.1%

Table 1: Training and Validation Accuracy of Different Models

5.1 GNB

The baseline model, GNB, achieved an accuracy of 41.4% on the training set and 38.8% on the validation set.

5.2 VNN

The final VNN achieved 58.6% training accuracy and 53% validation accuracy. It had 1500 hidden ReLU units and softmax output units. Mini-batches of size 10 were used and the optimization was run for 15 epochs. The Adadelta optimizer, with $\epsilon = 1e-4$, $\rho = 0.7$ and a learning rate of $1e-3$, was used for this purpose. Dropout did not turn out to be useful here.

5.3 CNN

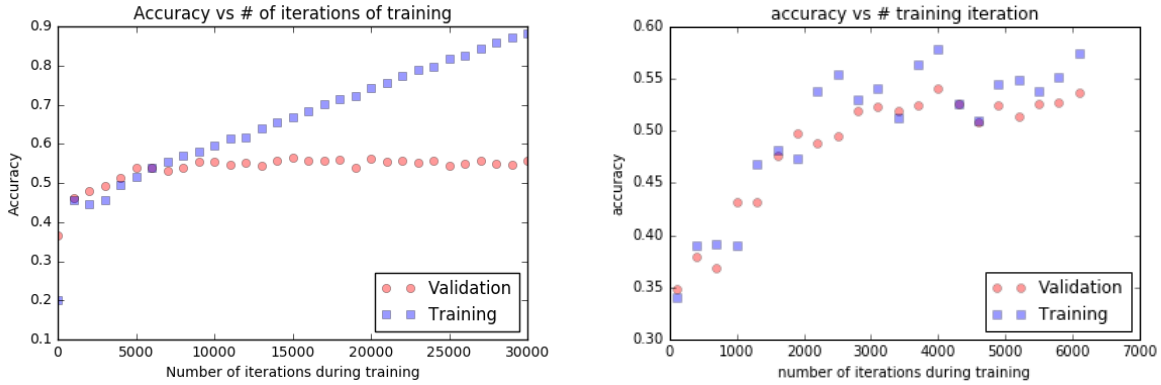
The final CNN we designed achieved a training accuracy of 72.9% and a validation accuracy of 56.9%. As described previously, it had one convolutional layer and one max-pooling layer followed by a single densely connected hidden layer that then led to the output layer. The convolutional layer had 150 filters, each with a window size of 4, while the hidden layer had 500 units. ReLU activation was used both for the convolutional layer and the densely-connected layer. The output layer was made of softmax units. Mini-batch optimization with a batch-size of 10 was run 20,000 times (20 epochs). The Adadelta optimizer, which is readily available in TensorFlow, was used with $\epsilon = 1e-4$, $\rho = 0.8$ and a learning rate of $1e-3$. Dropout was not useful here. Table 2 shows the hyperparameter tuning process. The training accuracy and validation accuracy history is displayed in Figure 4:

No. of filters	**Window size**	**No. of hidden layer units**	**Validation Accuracy**$(\%)$
250	4	500	55.9
150	4	500	56.9
50	4	500	55.2
150	3	500	56.2
150	5	500	56.3
150	4	700	55.3
150	4	300	55.5

Table 2: Variation of CNN validation accuracy (20,000 iters) with hyperparameters

5.4 RNN

The RNN achieved a training accuracy of 58.8% and a validation accuracy of 54.1%. It had 2 LSTM layers each comprising 100 LSTM units. A learning rate of $3e-3$, a batch size of 50 and a dropout rate of 0.5 were used to train the network for 20 epochs. The training accuracy and validation accuracy history is depicted in Figure 4:



(a) CNN (b) RNN

Figure 4: Training and Validation Accuracy for CNN and RNN

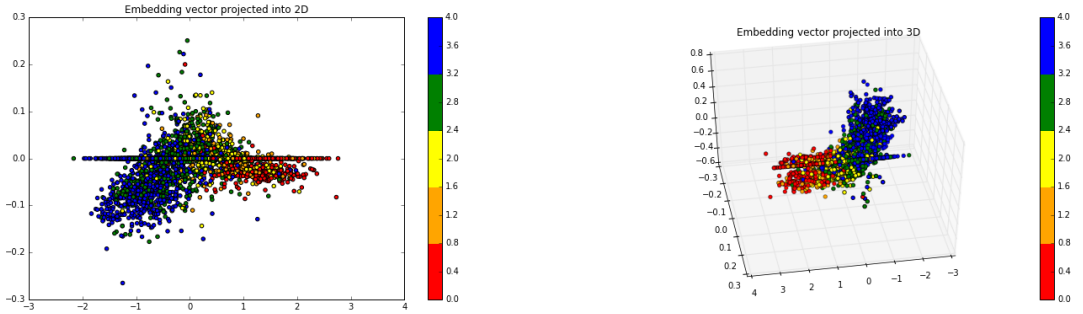
The effects of hidden layer dimensionality and dropout rate on accuracy were also examined and the results are listed in Table 3

5.4.1 Visualization of LSTM outputs using PCA

The outputs of the final LSTM layer in the trained RNN, which form a 100-dimensional vector for each review, were extracted and projected onto 2D and 3D spaces using PCA. The results are visualized in Figure 5. It can be observed that reviews with similar ratings are close to one another, and the two extreme classes (1 and 5) are at the two far ends. This can be viewed as an embedding for the reviews and similar reviews can be found by calculating their distance in the embedding/principal components space.

Dropout	No. of hidden layers	Hidden layer size	Validation Accuracy (%)
0.5	1	150	49.7
0.8	2	100	50.2
0.2	2	100	48.9
0.5	2	100	54.1
0.5	2	200	52.9
0.5	2	50	52.0

Table 3: Variation of RNN validation accuracy (10,000 iters) with hyperparameters

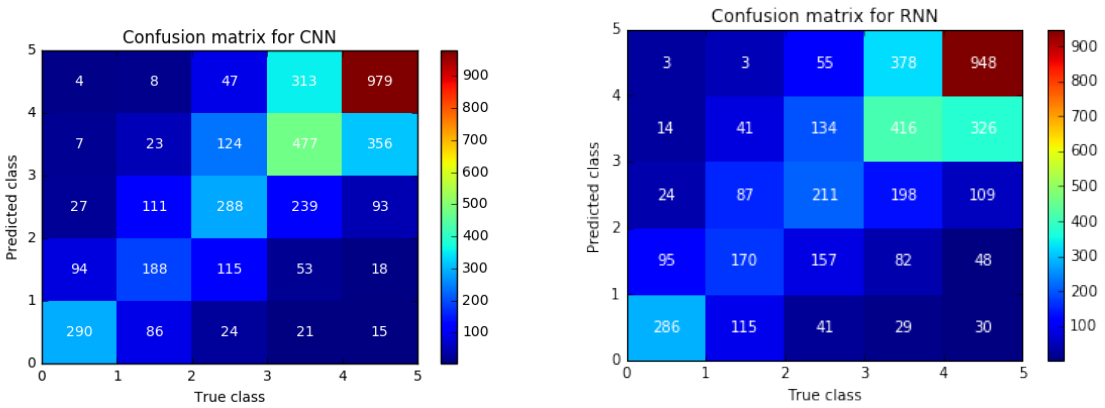


(a) Embedding into 2D (b) Embedding into 3D

Figure 5: PCA for Review Embedding

5.5 Visualization of Results

Visualizations of the confusion matrices for RNN and CNN are presented in Figure 6.



(a) Confusion Matrix for CNN (b) Confusion Matrix for RNN

Figure 6: Confusion Matrices for CNN and RNN

It can be observed that most of the wrongly classified reviews are closely aligned with the main diagonal. Though we had assumed no ordering relation between different classes while training the model, the ordering relation was actually captured by the model (i.e. a 5 star rating is closer to a 4 star rating than a 1 star rating). This also partially explains the phenomenon that the highest accuracy for this task is relatively low (below 60). Different people may give different ratings (i.e. a 4 star and 5 star) even if they submit similar reviews, so the intrinsic variance of the dataset is relatively high. We speculate that is this is one of the main reasons for the seemingly modest performance of our networks.

We can show that there is a large boost in the accuracy if this problem is recast as a 3-way or binary classification problem. For example, the 2-way (1,2,3 in one class and 4,5 in the other) classification accuracy of our CNN, which can be deduced from the confusion matrix above, is 83.7%. The 3-way classification accuracy (1,2 in one class, 4,5 in another and 3 on its own) is 76.8%.

5.6 Sample misclassified reviews

We actually took a look at the reviews on which our predictions were very off and discovered that some of them were really misleading – even humans would have failed in predicting their sentiment label.

Actual rating: 4/5
Predicted rating: 1/5

```
Sadly...there is new ownership at Don and Charlie's and no more chopped
chicken liver unless you pop 5.00 for it. Love the restaurant but not happy
with thus change.
```

Actual rating: 4/5
Predicted rating: 1/5

```
If you want to have true traditional Japanese sushi/sashimi, this place is
not a right place for you. I felt food was overpriced and food was average.
If your boyfriend take you there and enjoy upscale/romantic atmosphere, it's
a right place. Service was good and their restroom is very unique. When I
walked into the restroom, I thought I was drunk after a glass of sake. But
the washing hands sinks are located between men/women's restroom. A very kind
gentleman indicated to me the women's restroom. Thanks to him.
```

6 Conclusions

We conclude by saying that we were able to successfully implement CNNs and RNNs (using the TensorFlow libraries) for the task of predicting ratings given text reviews from the Yelp dataset. The parameters were carefully tuned and intuitive visualizations were presented for clear understanding of the results. The results were also compared with those of the baseline Naïve Bayes and VNN models and shown to be superior.

The reasons for the slightly underwhelming performance of the networks were discussed and their strength in binary/3-way classification was demonstrated. The high variance of the dataset samples was also demonstrated using two examples where our models failed.

References

[1] https://www.tensorflow.org/tutorials/layers

[2] https://www.tensorflow.org/tutorials/recurrent

[3] https://github.com/aymericdamien/TensorFlow-Examples

[4] Stanford NLP class slides available at http://web.stanford.edu/class/cs224n/syllabus

[5] Kim, Yoon. "Convolutional neural networks for sentence classification." arXiv preprint arXiv:1408.5882 (2014).

[6] http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/

[7] http://colah.github.io/posts/2015-08-Understanding-LSTMs/

[8] Hochreiter, Sepp, and Jrgen Schmidhuber. "Long short-term memory." Neural computation 9.8 (1997): 1735-1780.

[9] Palangi, Hamid, et al. "Deep sentence embedding using long short-term memory networks: Analysis and application to information retrieval." IEEE/ACM Transactions on Audio, Speech and Language Processing (TASLP) 24.4 (2016): 694-707.

[10] http://karpathy.github.io/2015/05/21/rnn-effectiveness/