Sentiment Analysis on Yelp data with Neural Networks

CSD316: Introduction to Machine Learning
under
Professor Harish Chandra Karnick

By Amir Suhail

Motive

- The yelp dataset contains over 6 million text reviews from users on businesses, as well as their rating.

- This dataset is interesting because it is large enough to train even advanced machine learning models like LSTMs (Long Short-Term Memories).

- We ultimately want to train our machine to predict whether the review is positive or negative given only the text section.

Index

1. Gathering System Requirements

Before doing any large scale data analysis, we need to know how much resources are available on our system. The main resources we care about are:

- RAM
- Disk Space
- No of cores on our CPU

The laptop I’m working on has 4 cores and 8GB of RAM available.

To get efficient results in data preprocessing parts, we will divide the work on all our four cores at any given time to facilitate multiprocessing during runtime.

2. Text Preprocessing

- In this section, we will start by converting the text data of Yelp, to a numpy array, which can fit in memory and is suitable to the ML models we want to run.
- We have a dataframe with two columns: ‘stars’ and ‘text’.

We have two issues to figure out here:

1. To feed the review text to a neural network, for example, we need to encode it first(into an array of numbers).
2. Loading the full dataset in memory is barely doable on a laptop/computer with 16 GB of RAM.

2. Text Preprocessing

The important things we are focusing on are:

- organizing a python package in a modular way.
- Considerations in algorithmic complexity.
- Efficient memory management.
- Multiprocessing.

2. Text Preprocessing

We use a tiered processing approach to transform this data into a structure that would entirely fit in memory.

For each review, we will take the following steps:

- extract the words from the review text
- encode words into integers
- convert the lists of integers to a numpy array

The first two steps are CPU intensive, and the last one needs a lot of RAM.

2.1 Divide and Conquer

Tasks than require a lot of RAM may not be parallelized on a single machine, since all cores share the same RAM. For these reasons, I decided to isolate CPU- and RAM-intensive tasks in different processes.

1. Now, to be able to perform CPU-intensive tasks in parallel on different cores, we need to split our sample into chunks, so that each core can take care of a particular chunk (we have four cores, so we will make 4 chunks of 16671475 lines each).

2.2 **Text Parsing using Natural Language Toolkit**

There are two tasks that are going to be done in several scripts: parallelization to several cores, and the definition of some of the command line arguments. Indeed, these scripts have the following in common:

- they read several input files from a directory
- they perform some task on each file, possibly returning tokenized results.
- All reviews have been split into words, and now we need to find out how to convert these words into a suitable format.

2.3 **Building the Vocabulary and Encoding**

In this section, we will do the encoding .To do that, we will build a vocabulary, which is an ordered list of all possible words in all reviews. Then, a word can be encoded as a number giving its position in the vocabulary.

- We use an Index dictionary, which makes the correspondence between a word as key and the index as value.
- When a new key / value pair is added to the dictionary, the key is hashed (converted to an integer).
- index.pck contains the vocabulary we will need further.

After the vocabulary is built for the whole yelp dataset, we will encode all reviews. For that, we will look up the integer corresponding to each word in the index. That’s a massive number of searches, so it must be fast.

2.4 Converting the encoded dataset to a numpy array

- Numpy arrays are used as input for machine learning because they have excellent performance in numerical analysis, and is represented in memory by a contiguous section.
- In our case, we need to have an array with two axes, like an excel table. The first axis (rows) will index the examples (the reviews) and the second axis (columns) will contain the rating (stars) followed by all encoded words in the review text.
- We will impose a maximum review size, to deal with variations in review size.
- So we want a numpy array containing the whole yelp review data, that fits entirely in memory.
- We will limit the review size to 250 words only to fit it into our memory and run the script to convert the dataset to a numpy array.

3. Exploring the data

On the raw dataset (review.json) we have almost 6.7 million lines.

- The “stars” field contains the notation, and the “text” field contains the review.
- Our goal was to train our machine to predict whether a review is positive or negative given the review text.

Next we try to explore our processed data file (data.h5)

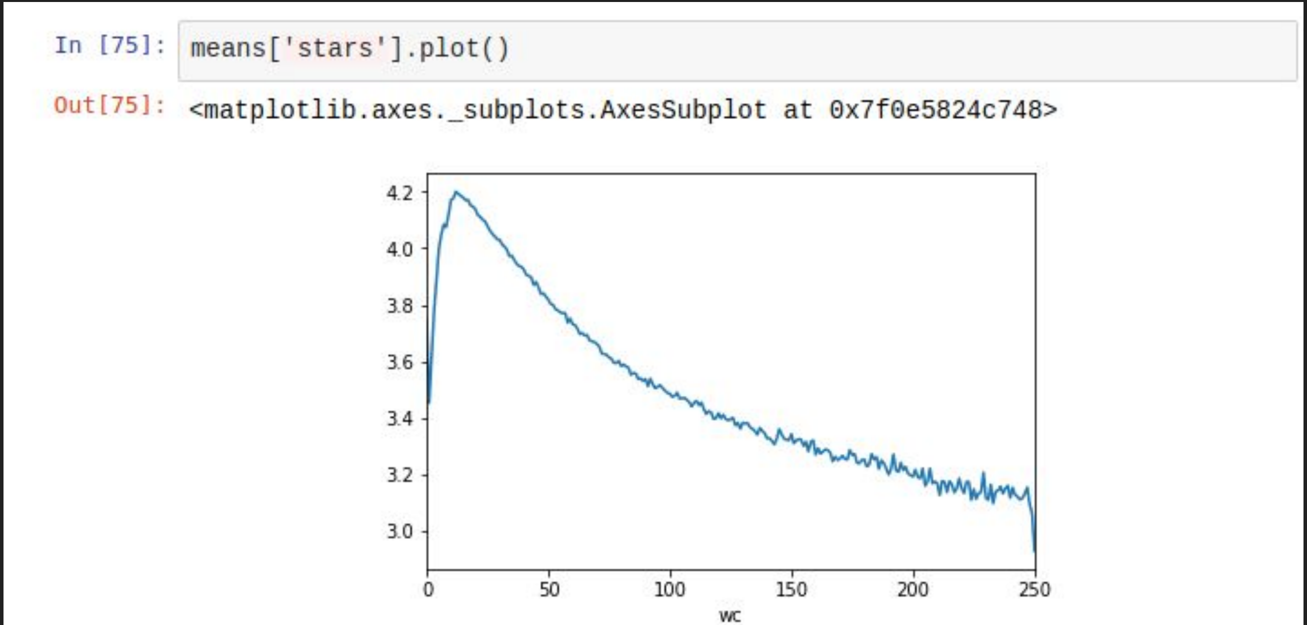
The data is as expected: 6685900 reviews, and for each of them, we have 254 values in the same order:

1. Rating
2. Useful
3. funny
4. Cool
5. 250 encoded words.

3. Exploring the data

Initially, we look at how the average rating evolves with the number of words in our python notebook.

- After a peak around 15 words, the average rating decreases steadily with the number of words. It looks like the more angry people are, the more they write.



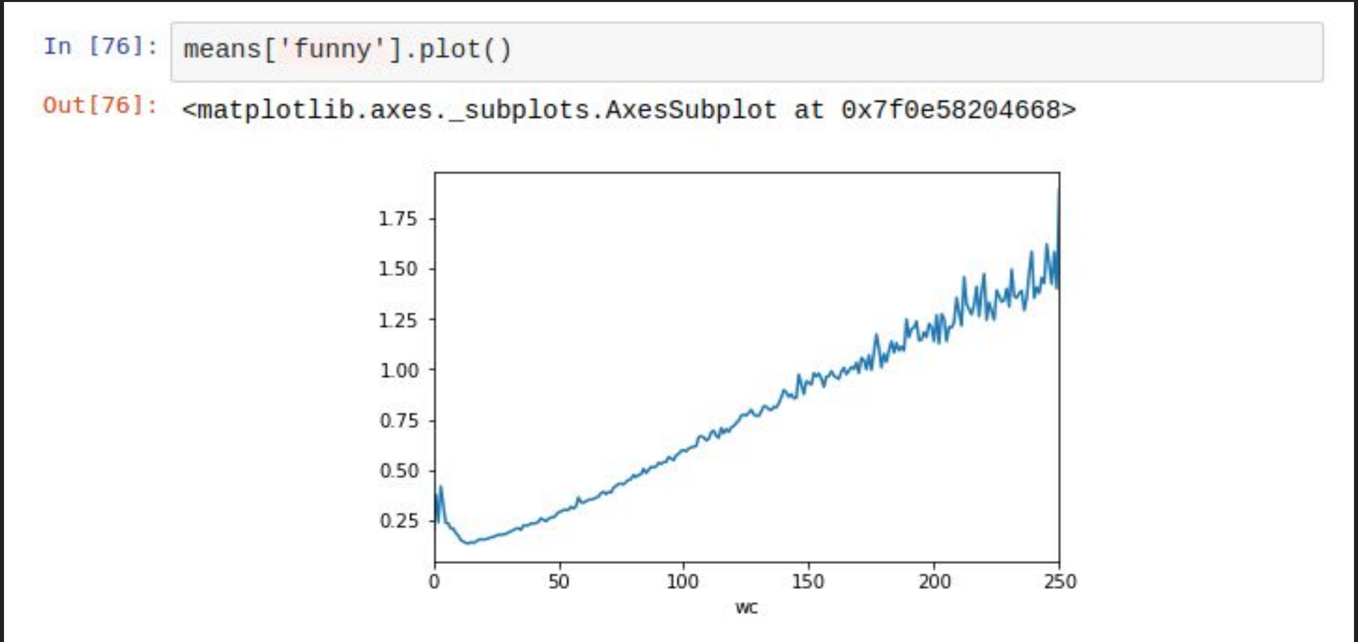
```
In [75]: means['stars'].plot()

Out[75]: <matplotlib.axes._subplots.AxesSubplot at 0x7f0e5824c748>
```

3. Exploring the data

Taking a look at the average of ‘funny’ scores:

- Long reviews are more funny than the short ones. But actually, some short reviews are funny too, but I'm not too sure why.



```
In [76]: means['funny'].plot()

Out[76]: <matplotlib.axes._subplots.AxesSubplot at 0x7f0e58204668>
```

4. Sentiment analysis using Neural Networks

- Sentiment analysis is an important application of natural language processing, as it makes it possible to predict what a person thinks given the text she has written.
- We will try to classify the reviews of the yelp dataset as positive or negative with two different neural networks (DNN and CNN).
- First, we will try a simple network consisting of an embedding layer, a dense layer, and a final sigmoid neuron.
- Then, we will see how convolutional layers can help us improve performance.

4. Sentiment analysis using Neural Networks

- Our goal is to predict whether the review text is positive or negative. Therefore, we need to label our examples in two categories: 0 (negative) and 1 (positive).

- We see that the number of stars ranges from 1 to 5, so it's not possible for a reviewer to give no star.

- Then, we want to split the dataset in two categories that have roughly the same number of examples.

- I prefer to define as positive all reviews with 4 stars or more.

4.1 A simple dense neural network

Our first deep neural network will contain:

- An embedding layer,
- A dense layer, responsible for interpreting the results of the embedding,
- A final sigmoid neuron that will output the probability for the review to be positive.

At first, we will use 20000 examples for the test sample, and "only" 100,000 examples for the training sample.

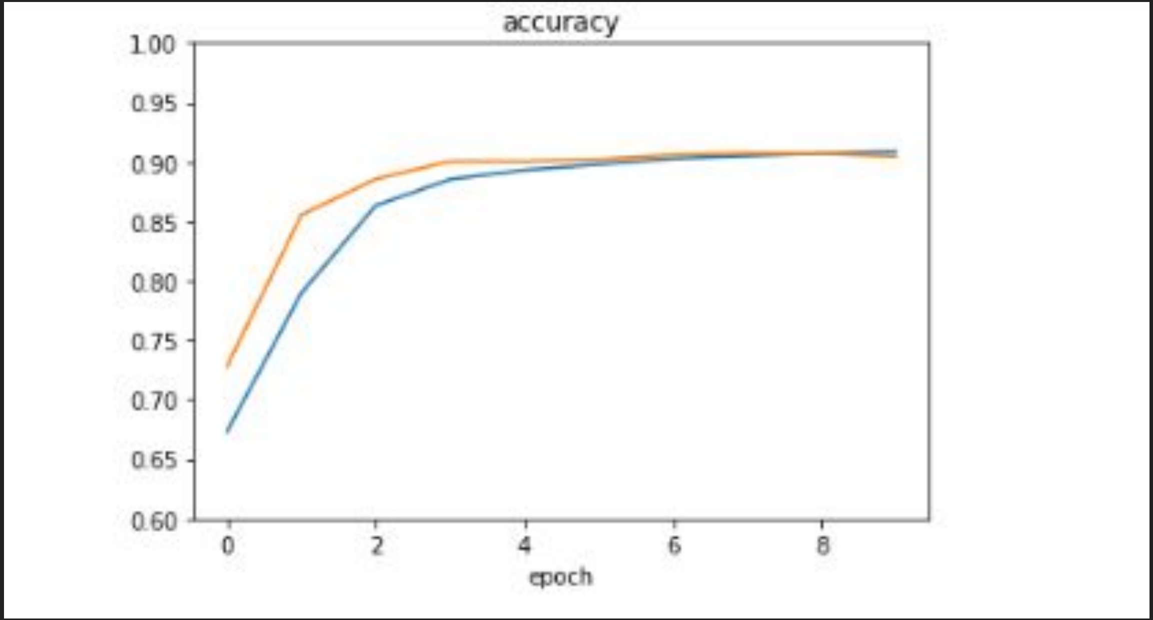
- The first layer will be the embedding layer. Its role is to convert each integer representing a word into a vector in N-dimensional space. In this space, words with similar meaning will be grouped together. We start with a 2-dimensional embedding space.

4.1 A simple dense neural network

- Since the input to the layer has a rank greater than 2, then it is flattened before moving forward. This 2D array cannot be used directly as input to a dense layer, so we need to flatten it into a 1D array with 500 slots
- Then, we add dropout regularization. This forces the network to learn different paths to solve the problem, and helps reduce overfitting .
- We end with a dense layer consisting of a single neuron with a sigmoid activation function.
- this neuron will produce a value between 0 and 1, which is the estimated probability for the example review to be positive.

We see that we end up with an accuracy of about 90%.

4.1 A simple dense neural network



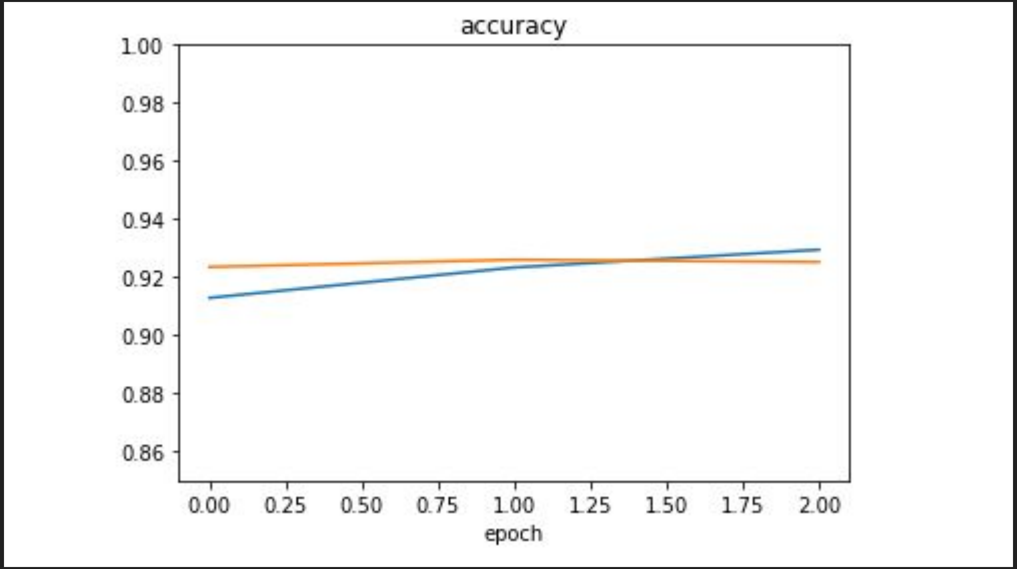
- The training accuracy plateaus at 90%, so training further will not help much.

- What we see here is that this network underfits the data, meaning that architecture is not complex enough to fit the data. By making it more complex, the training and testing accuracies can certainly be improved.

4.2 Dense network: increasing complexity

- After some tuning, I converged to the following architecture with one embedding layer, 3 dense layers and then the final sigmoid activation function. The structure of the network is complex, so I use the full dataset to avoid overfitting.

- That's an improvement over the previous attempt, but the training is quite long, and it seems we will not be able to reach 93% classification accuracy on the test sample with this technique. In the next section we try a different strategy.



4.3 Using a simple convolution network

- In this section, we will introduce a 1D convolutional layer in our network, with a kernel size of 3 and 64 filters. This means that 64 features (values) will be extracted from each position of the kernel.
- At each step, the kernel moves by one word in the review text.

- The convolutional layer will find it whatever its position in the sentence. Also, it will be easy for the network to understand the meaning of words which form meaning together. On the contrary, in our previous attempt for example, not and good are not directly considered together.
- This means that 64 features (values) will be extracted from each position of the kernel

4.3 Using a simple convolution network

With the convolutional layer, we get almost the same performance (92.55 accuracy) as with our best try with a simple dense network. However, please note that:

- there are only 2 million parameters in the network, instead of 10 million
- the convolutional network is less subject to overfitting, and we could restrict the number of training examples to 1 million instead of 6.7 millions, and the training was much faster
- there is room for optimization as the model is really simple for now.

4.4 Stacked convolutional layers

- In this section, we will optimize our convolutional network further by stacking convolutional layers.
- We perform max pooling between each convolutional layer, and the layers extract more and more features as we progress in the network.
- To avoid overfitting, we use the whole dataset for training except for 20000 events that are kept for testing.
- Almost 93.7 percent accuracy is achieved, with only a little bit of overfitting.