Data Analytics Project 1

Abuaf Pelo Ian & D'Onofrio Jury Andrea

April 2023

1 Introduction

The datasets provided come from the paper Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank written by Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher Manning, Andrew Ng and Christopher Potts.

The original folder includes:

1. `original_rt_snippets.txt` contains 10,605 processed snippets from the original pool of Rotten Tomatoes HTML files. Some of the snippets contain two or more sentences. The original paper was concerned with finding a sentiment value for each sentence in the dataset, rather than with the entire snippet.
2. `datasetSentences.txt` contains the sentence index, followed by the sentence string separated by a tab. These are the sentences of the train/dev/test sets for the original paper.
3. `dictionary.txt` contains all phrases and their IDs, separated by a vertical line |
4. `sentiment_labels.txt` contains all phrase ids and the corresponding sentiment labels, separated by a vertical line.

   We can recover the 5 classes by mapping the sentiment value using the following cut-offs:

   - $[0, 0.2] \rightarrow$ very negative
   - $(0.2, 0.4] \rightarrow$ negative
   - $(0.4, 0.6] \rightarrow$ neutral
   - $(0.6, 0.8] \rightarrow$ positive
   - $(0.8, 1.0] \rightarrow$ very positive

   Please note that phrase ids and sentence ids are not the same.
5. `SOStr.txt` and `STree.txt` encode the structure of the parse trees.

   STree encodes the trees in a parent pointer format. Each line corresponds to each sentence in the `datasetSentences.txt` file.

   The Matlab code of this paper is not provided.
6. `datasetSplit.txt` contains the sentence index (corresponding to the index in datasetSentences.txt file) followed by the set label separated by a comma:
   1 = train
   2 = test
   3 = dev

The `datasetSentences.txt` file has more sentences/lines than the `original_rt_snippet.txt`.
Each row in the latter represents a snippet as shown on RT, whereas the former is each sub sentence as determined by the Stanford parser (which we did not have access to).

2 Mapping

Since the Matlab code was not provided, we have tried to recover the orignial mapping from `original_rt_snippets.txt` to `datasetSentences.txt` and from `datasetSentences.txt` to `dictionary.txt`.
Mapping the dictionary of phrases to the corresponding sentiment value provided in the `sentiment_labels.txt` file was easy enough, a single join operation on the two datasets was sufficient.
However, we spent a long time trying to recreate the mapping from the original snippets to the sentences and then to the phrases. We applied different methods to standardize the data, such as the word tokenizer provided in the `nltk` library, a regex tokenizer that removes all punctuation and a mix of the two.
Certain substrings had to be replaced separately from the tokenizer, like how the open parenthesis character "(" in the snippets was replaced with "`-LRB-`" in the sentences, and then the phrases included the same snippet twice with both the substring "`-LRB-`" and the character "(".
For example, the original snippets contain the following review:

```
1 Its mysteries are transparently obvious, and it's too slowly paced to be a
    thriller. (But it's) worth recommending because of two marvelous
    performances by Michael Caine and Brendan Fraser.
```

This review is then split into two different sentences in the `datasetSentences.txt` file:

```
1 Its mysteries are transparently obvious , and it 's too slowly paced to be a
    thriller .
2 -LRB- But it 's -RRB- worth recommending because of two marvelous performances
    by Michael Caine and Brendan Fraser .
```

Notice how the parenthesis are replaced with `-LRB-` and `-RRB-`. The review is then further split in phrases found in the dictionary file.
Some of the phrases start at the beginning of the review, and they are the following:

```
1 Its mysteries
2 Its mysteries are transparently obvious
3 Its mysteries are transparently obvious ,
4 Its mysteries are transparently obvious , and
5 Its mysteries are transparently obvious , and it 's too slowly paced to be a
    thriller
6 Its mysteries are transparently obvious , and it 's too slowly paced to be a
    thriller .
7 Its mysteries are transparently obvious , and it 's too slowly paced to be a
    thriller . ( But it 's ) worth recommending because of two marvelous
    performances by Michael Caine and Brendan Fraser
8 Its mysteries are transparently obvious , and it 's too slowly paced to be a
    thriller . ( But it 's ) worth recommending because of two marvelous
    performances by Michael Caine and Brendan Fraser .
9 Its mysteries are transparently obvious , and it 's too slowly paced to be a
    thriller . -LRB- But it 's -RRB- worth recommending because of two marvelous
     performances by Michael Caine and Brendan Fraser .
```

Notice now how some of them include the parenthesis characters while in others it's replaced with the `-LRB-` and `-RRB-` substrings.
In other cases, the string "empowerment" in the snippets was parsed as "em powerment" in the sentences, and "don't" was parsed as "do n't". As we didn't have access to the parser, these cases are handled by us with individual `replace` calls.
These inconsistencies made it really hard for us to find a mapping from the snippets to the phrases. While we were eventually able to map each original snippet to one or more sentences, we were unable to do the same for the phrases of the `dictionary.txt` file.
As such, our mapping got mostly unused in the end, as the model was only trained on the phrases of `dictionary.txt` and we had no mapping from the original snippets to them.
The combined `datasetSentences.txt` and `original_rt_snippet.txt` file can be computed via the `ipynb` script that we provide and is saved in `temp/combined.csv`.

3 Visualizations

For visualizations, we chose to focus on pandas + seaborn, since our dataset requires a lot of preprocessing that either can't be done in Tableau, or is very complicated to do in Tableau.

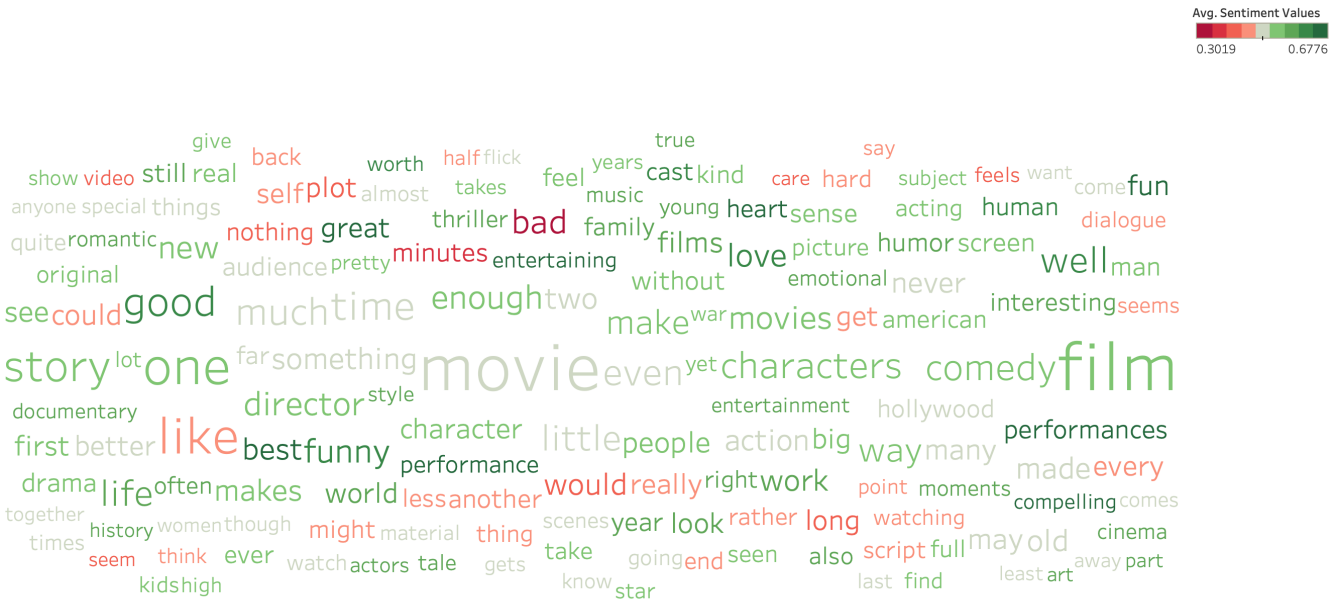


Figure 1: Word frequency of the most prevalent words in the dataset, colored my the average sentiment value.

However, we used Tableau for visualizing Figure 1.
It shows the words in the dataset, scaled by the number of occurrences, and then colored by the average sentiment value that of the phrases the words appear in, green being words that appear more often in positive phrases and red words that appear more often in negative phrases.
Only words that appear more than 700 times in total are shown.
In order to visualize the data in Tableau, we had to preprocess it a bit in Python. First, we split each row of the dataset by words, and then "exploded" the rows in order to create one for every work occurrence in the dataset.
We also used Python to filter stopwords, using the set of english stopwords from `nltk`, and also removed words that were shorter than 2 characters.
We can notice that, as expected, a lot of "positive-sounding" words are green, such as "fun", "best" and "entertaining", while a lot of "negative-sounding" words are red, such as "bad". Strangely, "like" is also mostly found in negative phrases.

3.1 Length to sentiment values

While we will see other graphs in Section 4 and onwards, we now focus on the relation between sentiment value and length of the phrase.
While there aren't many other variables we can look at for visualizations apart from word frequency (which we already saw) and sentiment values, there might be some correlation between the length of a phrase and its sentiment value, which is what we look at now.
Figures 2 and 3 show this relation and, as expected, there is not much relation between these two variables as the length increases. (Although we can see a slight upwards trend in Figure 3 at the very right, indicating that most of the longest snippets in this dataset are positive)
However notice how most of the phrases with small length tend to be scored with a neutral (0.5) sentiment value. This is indicated by the concentration of phrases in Figure 2 near the center of the x axis and at the bottom of the y axis.

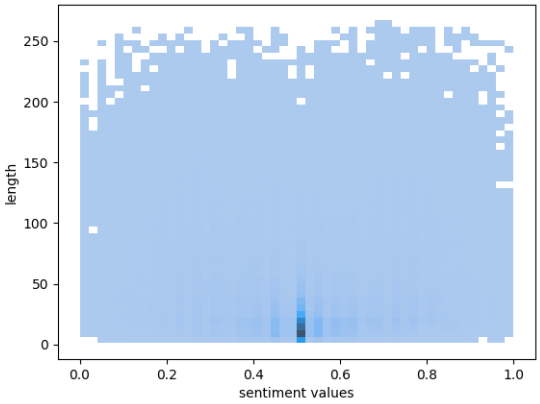


Figure 2: Histogram plot with sentiment value on the x axis and length on y axis. Notice the high concentration in the middle.

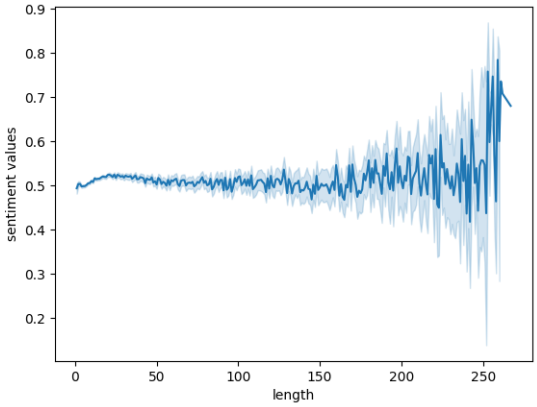


Figure 3: Relation between sentiment value and length in a lineplot

This is because, as more and more sections of the entire snippet are removed, the context of a review is lost, so there
For example, consider the earlier snippet seen in Section 2: The phrase "Its mysteries" loses a lot of the context of the complete review or even of the first sentence "Its mysteries are transparently obvious , and it 's too slowly paced to be a thriller". Thus, it is likely counted as neutral.
Because there are many phrases like these, the total number of neutral phrases is very high, as we will see in the next section.
As the length of a phrase increases, more and more context is added until the sentiment value becomes clear.

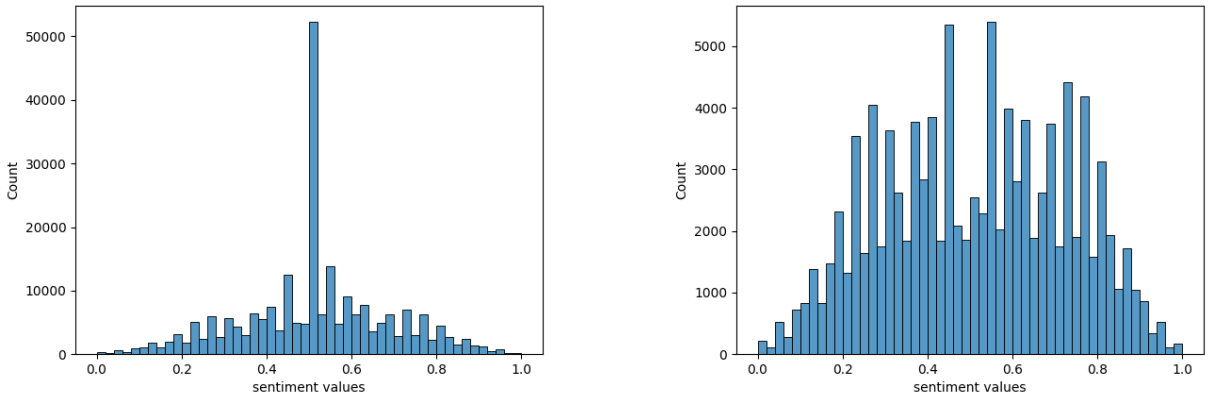
4 Prediction

For this project we implemented 3 models for prediction of the sentiment value.
As mentioned before, these predictions are based on the phrases provided in the dictionary file rather than on entire sentences.

4.0.1 Filtering Dataset

Before we trained our model, we noticed that a lot of our dataset is made up of phrases with a 0.5 sentiment value. This can be seen in Figure 4a, as the central pillar vastly overtakes the rest of the dataset.

This is due to the fact that a lot of phrases are made up by very few, if not single words. As we saw in Section 3.1, these are often scored with a 0.5 sentiment value or similar.

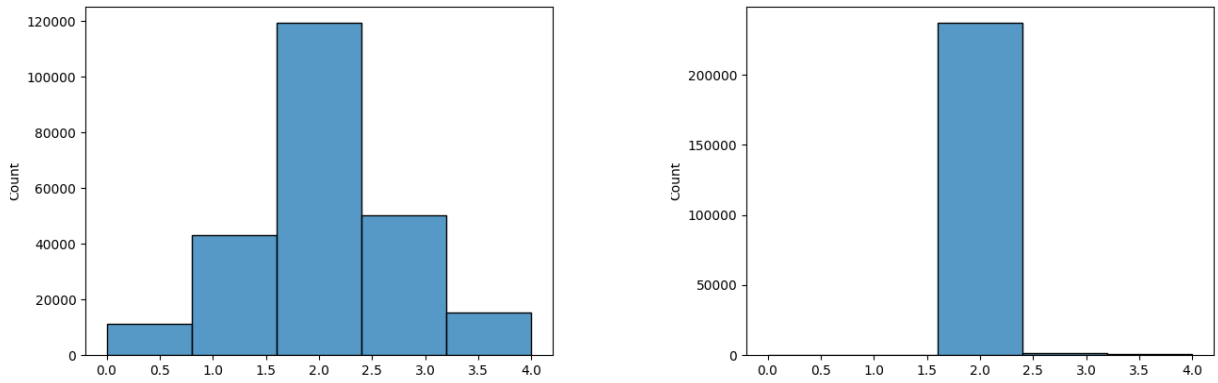


(a) Before removing data points

(b) After removing data points

Figure 4: Distribution of sentiment values before and after removing data points.

When we tried to train our models on the entire dataset, not only training very slow, but also we noticed that a lot of our models tended to award scores closer to 0.5.
This can be seen in Figure 5, which shows the comparison between the distribution of categories in the real test set of the database and the prediction given by a classification model trained without accounting for the majority of sentiment values being 0.5.



(a) Distribution of categories in dataset

(b) Prediction of categories by linear classification model

Figure 5: Distribution of categories in the dataset compared to a linear classifier's prediction.

Our solution to this problem was to pre-process the data and take out many of these 0.5 data points. We tried many iterations and settled on the following formula for filtering our dataset:

```
filter =
(Length >= 30) &
((isSentence == True) | (sentimentValue != 0.5))
```

where:

- `Length` is the number of characters in each phrase;
- `isSentence` is whether each phrase starts with an uppercase letter and terminates with a full stop character ".". We used this approximation to find full sentences in our dictionary of phrases.

- `sentimentValue` is the sentiment value of each phrase, a number from 0 to 1.

This is meant to reduce the number of completely neutral (0.5) phrases without removing those that are full sentences.
The result of filtering the dataset this way can be seen in Figure 4b. As we can see, the distribution is a lot flatter once this is in effect. For the rest of our predictions we will use this reduced dataset.
For splitting the dataset into a train and a test set we used the built-in sklearn function `train_test_split`. we have a split of 40% of data into the test set and 60% into the train set.

4.0.2 Feature Extraction

It should be mentioned that all of our models preprocess the input with a simple `CountVectorizer()`, which turns the strings into their bag-of-words representation.
We also tried to add a further tf-idf transformer after the count vectorizer, but we only saw worse performance with it. We think this is because the data points have too few words for the term frequency to become a relevant factor. Therefore, all models will use a simple bag-of-words feature extraction.

4.1 Linear Classification

The first model we trained is a classification model which predicts for a given review in which of the 5 categories it falls in (very negative, negative, neutral, positive or very positive).
It is a simple classifier using Stochastic Gradient Descent, using the class `SGDClassifier()` from sklearn.
With this model we got an accuracy score of 0.513 and a mean squared error of 0.770. It should be noted that the categories are enumerated from 0 to 4.

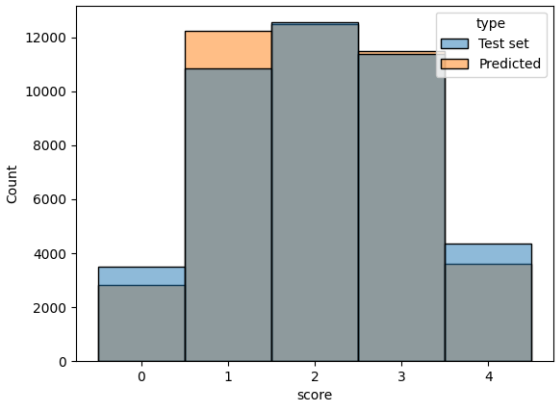


Figure 6: Distribution of the classifier prediction compared to test set

Figure 6 shows the distribution of the test set as predicted by the classifier (orange) and as it is in reality (blue). Compared to the other models that we will see, the distribution is farely similar, however it performs the worse out of the models.

4.2 Linear Regression

Our first regression attempt was with a linear regressor from the `LinearRegression()` class.
This regressor performs similarly to the linear classifier, with a score of 0.565 and a mean squared error of 0.020.
Figure 7 shows the distribution of the test set as predicted by the regressor (orange) and as it is in reality (blue). Notice that the distribution of the predicted set is similar to a gaussian distribution, and that certain data points lie outside the expected bound of sentiment values, that is between 0 and 1.

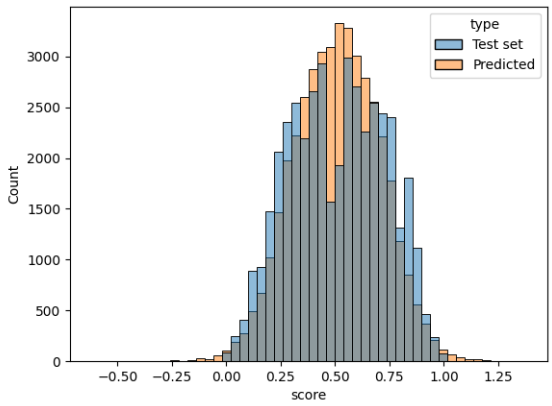


Figure 7: Distribution of the linear regressor prediction compared to test set

4.3 Neural Network Regression

The final and most complicated regressor we trained is a neural-network based regressor created with the `MLPRegressor()` class.
For this neural network we added a few hidden layers, in particular 4 layers each with 100 nodes, in order to improve predictions. The activation function for the hidden layers is left as the default, which is relu.
The solver is also left as the default for sklearn, which is adam.
We also limited the number of iterations to 100, although it manages to converge faster due to its tolerance of 0.0001, that is it stops if after 10 iterations the loss function doesn't improve by 0.0001.
This model is the best performing model out of the ones that we trained, with a MSE of 0.014 and a score of 0.688.

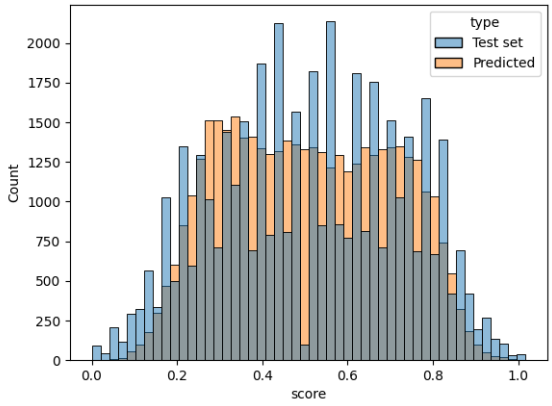


Figure 8: Distribution of the neural network regressor prediction compared to test set

Figure 8 shows the distribution of the test set as predicted by the regressor (orange) and as it is in reality (blue).