Sophia Nicolas

Prof. Vroman

CS362N.1

04/12/25

Homework 4 Report

Neural Network for Sentiment Analysis of Animal Crossing Reviews

Data Preprocessing

The dataset analyzed by the neural network was one consisting of player reviews for the video game Animal Crossing by Nintendo. The model's task was to classify a review as positive or negative, learning from positive reviews as labelled 7 or higher and negative reviews as lower than 7.

The dataset contained two columns: user reviews and grade. Reviews were tokenized by word with `nltk.word_tokenizer()` and had their stopwords removed. Preprocessed reviews and their corresponding grades were then gathered into tuples consisting of (review, grade). Reviews were converted back from token lists into strings, and grades were reduced to a binary format: 0 for negative reviews graded lower than 7, and 1 for positive reviews graded higher than 7.

The text reviews were vectorized using TFIDF. A matrix representing all the vectorized reviews where each row is a review, and each column is the frequency of a word in the vocabulary spanning across all documents. Tuples of (vectorized review, binary grade) were stored in a separate list as well. These tuples are then divided into a 70%-30% split for training and testing data.

```
# Keep each vectorized review paired with its corresponding label as a tuple:
tuple_vector_reviews = []

    # Loop through the matrix indices and access reviews and labels:
for i in range(len(matrix_all)):

        # Each row vector is one review:
    row_vector = matrix_all[i]

        # A review's corresponding 0 or 1 label will be at the same index of all_labels:
    label = all_labels[i]

        # Store a (vectorized review, binary label) tuple:
    tuple_vector_reviews.append((row_vector, label))

    # Display the first ten vectorized reviews and their binary grades:
for v in tuple_vector_reviews[:10]:
    print(v)
```

✓ 0.0s

```
(array([0., 0., 0., ..., 0., 0., 0.]), 0)
(array([0., 0., 0., ..., 0., 0., 0.]), 0)
(array([0., 0., 0., ..., 0., 0., 0.]), 0)
(array([0., 0., 0., ..., 0., 0., 0.]), 0)
(array([0., 0., 0., ..., 0., 0., 0.]), 0)
(array([0., 0., 0., ..., 0., 0., 0.]), 0)
(array([0., 0., 0., ..., 0., 0., 0.]), 0)
(array([0., 0., 0., ..., 0., 0., 0.]), 0)
(array([0., 0., 0., ..., 0., 0., 0.]), 0)
(array([0., 0., 0., ..., 0., 0., 0.]), 0)
```

Network Architecture

The neural network model for sentiment classification uses the PyTorch sequential class and functions from `torch.nn`. The architecture I chose was a simple back-and-forth pattern of hidden layers and activation layers. The network layers used `nn.Linear()` function and the activation layers used `nn.ReLU()` as the activation function.

The first layer takes in the initial input, which is a vector representing 1 review. The review vector is integer-divided into half by the first layer, activated by RELU, and passed onto the second layer. This second hidden layer integer-divides the review further by 4. The final output layer will be a tensor with 2 values representing the 2 classes, negative and positive. The final output is then activated by RELU again.

I built the model this way with the intuition that incrementally reducing the dimensionality of the review vector by half and then by fourths would help gain a "zoomed-in", clearer image of patterns that exist in the data. I was hoping that this would be help guide the model layers in clearly defining features as belonging to one class or another as they get from a vector of length 11294 to a final output of length 2.

```
model = nn.Sequential(OrderedDict([
        # Input is 1 vectorized review. All reviews are of the length as defined by the TFIDF transformation.
        # For each layer:
            # nn.Linear(size of layer input, size of output feeding into the next layer)
                                        # The second parameter is chopped in half to reduce dimensionality in hopes of getting a clearer "zoomed-in" view of feature patterns in the each review.
    ("input", nn.Linear(len(train_vectors[0][0]), len(train_vectors[0][0]) // 2)),

    ("activation", nn.ReLU()), # Use RELU for activating the hidden layers.
                                        # Reducing the second parameter again:
    ("hidden_1", nn.Linear(len(train_vectors[0][0]) // 2, len(train_vectors[0][0]) // 4)),

    ("activation", nn.ReLU()), # Use RELU for activating the hidden layers.

                    # nn.Linear(size of layer input, size of 2 to represent the 2 classes being predicted: positive and negative sentiment.)
    ("output", nn.Linear(len(train_vectors[0][0]) // 4, 2)),

    ("activation", nn.ReLU()) # Use RELU for activating the hidden layers.

    # Softmax is automatically applied for the output when the nn.CrossEntropyLoss() function is run later in the code.
])
)
```

Model Performance

The model was trained using cross entropy loss and the Adam optimizer. Although the model was trained on only one epoch, it performed well overall during training and testing on both the train and test datasets. One epoch was chosen due to it taking close to an hour (50m 32.1s) to train.

During training, the model seems to predict well, averaging in the mid-80% level of accuracy.

- First run: Output rounded to 92.18%
- Second run: 85.18342067651263 %
- Third run: 84.80228680323964 %
- Fourth run: 84.32586946164841 %

During testing on the train data, the model predicts with the highest accuracy at 96.18866126727012 %.

During testing on the test data, the model seems to generalize well on unseen data, achieving and accuracy of 88.66666666666667 %.

The precisions and recall scores seem somewhat inverse for each class:

- Precision is lower and recall is higher for class 0.
- Precision is higher while recall is lower for class 1 (positive reviews).
- The higher scores are about the same in the 90s, while the lower scores are similar in the 80s.

The accuracy and average are about the same for both classes.

	precision	recall	f1-score	support
0	0.87	0.94	0.90	497
1	0.92	0.82	0.87	403
accuracy			0.89	900
macro avg	0.89	0.88	0.88	900
weighted avg	0.89	0.89	0.89	900

Misclassification Analysis

Most reviews were in English, but there were a few reviews in other languages that were labelled false negatives or positives due to imbalances in data.

Some reviews that were gibberish or held irrelevant content were labelled falsely as well.

Terminal output:

```
Misclassified review is at index 1976 in the user reviews dataset.
User review:
1 egg , 2 eggs , 3 eggs , 4 eggs , 5 eggs , 6 eggs , 7 eggs , 8 eggs , 9 eggs ... egg

True label: 1
```

Some positive reviews with constructive criticism were labelled as negative. Reviewers detailed how they enjoyed the game while also describing what could be better, using words like "disappointed" or "disappointing", which appear frequently in negative reviews.

On the other hand, negative reviews that described ideals that Animal Crossing did not meet for the reviewer were labelled as positive. These reviews mentioned words like "enjoyment", saying that the game lacked it while positive reviews may say they did find "enjoyment" in the game.

Many positive and negative reviews talked about similar features that both types of reviewers deemed as problems with the game, such as only having one free island to play with per Nintendo switch console.

Terminal output:

```
Misclassified review is at index 689 in the user reviews dataset.
User review:
i 'm already addicted to the game.pros : the graphics are gorgeous , the items keep me
interested in collecting , but i do n't feel the need to rush through everything in
order to do so . i love the character interactions and the animations are very
cute.cons : the rng could probably be more balanced ? some of the ui could have been
more streamlined . i see why people are upset about the lacki 'm already addicted to
the game.pros : the graphics are gorgeous , the items keep me interested in collecting
, but i do n't feel the need to rush through everything in order to do so . i love the
character interactions and the animations are very cute.cons : the rng could probably
be more balanced ? some of the ui could have been more streamlined . i see why people
are upset about the lack of content for a second islander , but to me , it feels the
same as how new leaf operated , one character making all the decisions while the
others build up their own stuff . though maybe i 'm biased since i have no-one to
share the island with except filler characters anyway . definitely a bit short-sighted
decision wise , but not as greedy as some other companies.… expand

True label: 1
Predicted label: 0
```

The model seemed to struggle also with classifying very short reviews, and reviews with lots of punctuations, interjections, or multiple letters like "aaaaa".

Terminal output:

```
Misclassified review is at index 1965 in the user reviews dataset.
User review:
one town per
switch.aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaa

True label: 0
True grade: 0
Predicted label: 1
```

Potential Model Improvements

With one epoch of training only, the accuracy of the model seems to be slightly unstable, sometimes scoring in the 90s and sometimes in the 80s. Spending a greater amount of epochs training or adjusting the model structure and parameters may help achieve a consistent, high accuracy.

The initial runtime for one epoch was 32m 13.8s. This was when the input was not a tuple of (review, label) but only a review vector. Review vectors were held in two separate matrices, a training set matrix and a testing set matrix. Their true labels were in a separate list of the same length as the matrix.

While this structure was faster to train on, it was scrapped because the model could not parse two different matrix vectors of different lengths. Because TFIDF was run on a training set of text data and a testing set of text data of different sizes, the vectors were not the same length. Thus, the train/test split is now performed after a TFIDF matrix of all the reviews is computed.

The current runtime for one epoch is 50m 32.1s. This is when the loop is having to access the reviews and labels simultaneously, as they are stored in the same data structure, which is a list of (review, label) tuples. This probably increases speed and therefore lowers efficiency. However, this method is chosen to keep vocabulary consistent for the train and test sets; both sets come from the same TFIDF matrix.