NLP Assignment 2

*Lakehead University

Pedram Khoshnevis
(Computer Science)
Student ID: 1119232
Pkhayyat@lakeheadu.ca

***Abstract*—In this assignment, I implemented a multiclass classifier using word2vec and a simple convolutional neural network. The word2vec model uses the google pre-trained word2vec negative model. The highest accuracy was achieved after 1000 epoch even though after 100 epoch there was only minimal increase in accuracy and other criteria. The highest accuracy during the training process was 82% (F1 score: 0.8217) and during testing was 64% (F1 score: 0.640)**

I. Introduction

In this assignment, I implemented a multi-class sentiment analysis method. My program is consists of word2vec method and a simple convolution neural network classifier. The purpose of this assignment is to classify movie reviews into multiple classes. The data that is used for training and testing is downloaded from the provided GitHub account [1]. This dataset (Sentiment-Analysis-on-the-Rotten-Tomatoes-movie-review-dataset) is consists of thousands of Rotten Tomato (https://www.rottentomatoes.com/) movie review corpus that was analyzed by humans and manually classified into 5 classes.

1) very negative
2) negative
3) neutral
4) positive
5) very positive

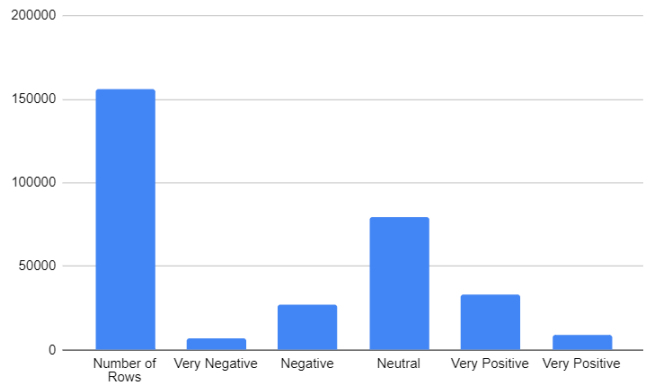

Fig. 1. Data set Class Distributions

Classes	Number of Instance
Very Negative	7072
Negative	27273
Neutral	79582
Positive	32927
Very Positive	9206

II. Methodology

Word2vec is a method that transforms a given text into a multi-dimensional vector [2]. My program takes advantage of word2vec provided by Google [3]. GoogleNewa vector negative 300, is consists of 3 million with 300 dimension English words. This model is downloaded during the runtime in my program uses this pre-trained word2vec model to transform the given text into vectors for the convolutional Neural network classifier.

A. Dataset:

I have used the Rotten Tomatoes movie review dataset that can be found on Kaggle or Github. To load the dataset, we use the panda's library. The dataset has 4 columns and 5 labels mainly, negative, somewhat negative, neutral, somewhat positive, and positive. PUT PICTURE OF THE TABLE

B. Cleaning the data:

I have used the dropna() to check if there were any NAN values in the dataset. Put picture of the table after dropping the nan values. Do df.tail()

Tokenization: It's a method to break larger sentences into smaller chunks. I used the nltk library.

C. Datapre processing:

Removing punctuations, numbers and special characters

D. Stemming:

it's a rule-based process of stripping the suffixes from the word. To do stemming in python, I used the PorterStemmer() that's part of the nltk library.

E. Lemmatization:

It converts a word into its root form. In NLTK, we use the WordNetLemmatizer to get the lemmas of words.

F. Removal of stopwords:

These are words which don't add to a sentence meaning. Therefore, they can be extracted safely without causing any alteration in the context of the sentence. There is a collection of stopwords in the nltk library and we can use these to delete stopwords from the text.

G. Features:

Since we cannot use the given text directly to the model, so we need to convert the given text into a numeric value. Bag of Words, TF-IDF and Word2vec are the most common methods to convert the text to a numeric vector. For this paper, I have used the Word2vec feature. Mention which import statements you are using.

H. Word2Vec:

Word2vec is a technique that's used in word embeddings. It uses shallow neural network. Its obtained using 2 methods: Continuous Bag of Words (CBoW) and Skip Gram.

I. CBoW:

This method takes the context of each word as the input and tries to predict the word corresponding to the context.

J. Skip Gram:

This method is used to predict the context word for a given target word. It's reverse of CBOW algorithm. Here, target word is input while context words are output.

K. Splitting the dataset:

To do the splitting, first we need to import the sklearn library and use the train_test_split method I used the train/test spilt ratio of 70:30 and a random state of 2003.

L. Calculating metrics:

Precision: it indicates how many selected items are relevant. Recall: it's the ratio of the total number of correctly classified true positive divided by true positive and false positive. High recall indicates that the class is correctly recognized. F1 score: it's the weighted average of precision and recall. An f1 score of 1 is the best value and a value of 0 is the worst value.

M. Training:

I used the model.fit() provided by keras to train the model. We need to specify the loss function, optimizer, and the metrics to train the model. This is done by the compile()

N. Testing:

I have used the model.evaluate() to test the model. I have also used the .predict() that generates output predictions for the given input samples.

O. Saving the trained model:

in keras, to save the model we used the model.save()

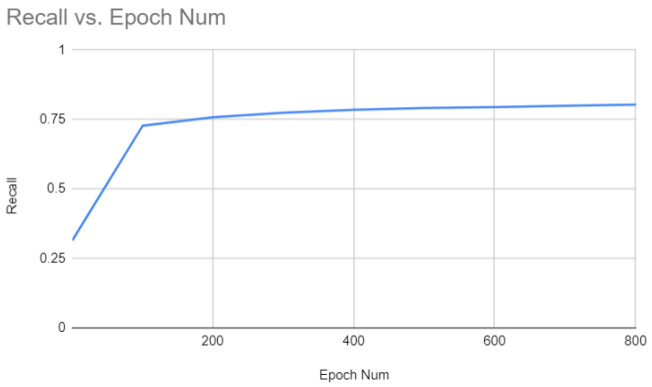

Fig. 2. Recall vs. Number of Epoch

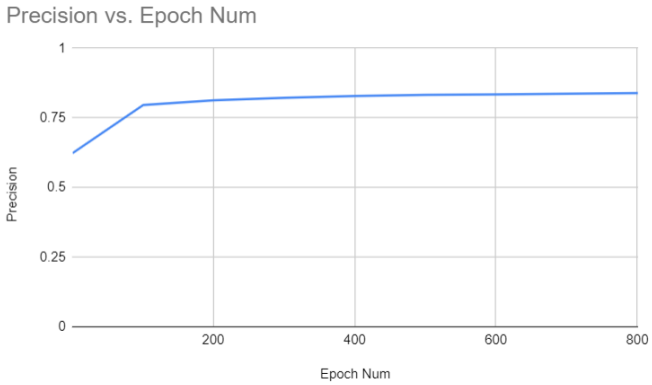

Fig. 3. Precision vs. Number of Epoch

III. Results

Based on the results showed in the charts and graphs below we can observe that the evaluation metrics grow rapidly until 100 epoch. The model was trained for 1000 epoch but the increase in accuracy was minimal. Table I shows the final evaluation score obtained from the testing dataset.

TABLE I
TESTING EVALUATION SCORE

Evaluation Type	Evaluation Score
Accuracy	0.643
F1_score	0.640
Precision	0.653
Recall	0.628

References

[1] X. Ni, "Sentiment-analysis-on-the-rotten-tomatoes-movie-review-dataset," https://github.com/cacoderquan/Sentiment-Analysis-on-the-Rotten-Tomatoes-movie-review-dataset, 2015.

[2] Y. Goldberg and O. Levy, "word2vec explained: deriving mikolov et al.'s negative-sampling word-embedding method," *arXiv preprint arXiv:1402.3722*, 2014.

[3] M. Mihaltz, "word2vec-googlenews-vectors," https://github.com/mmihaltz/word2vec-GoogleNews-vectors, 2014.

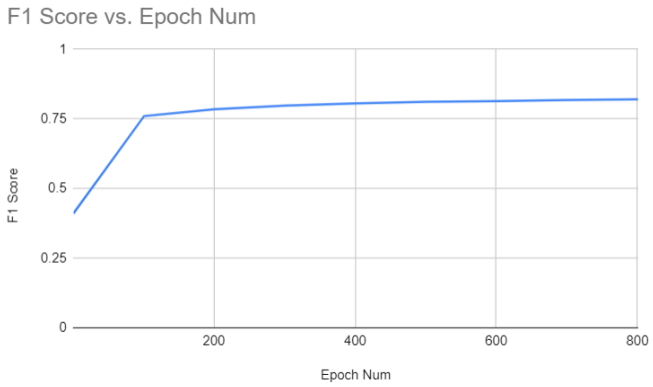

Fig. 4. F1 Score vs. Number of Epoch

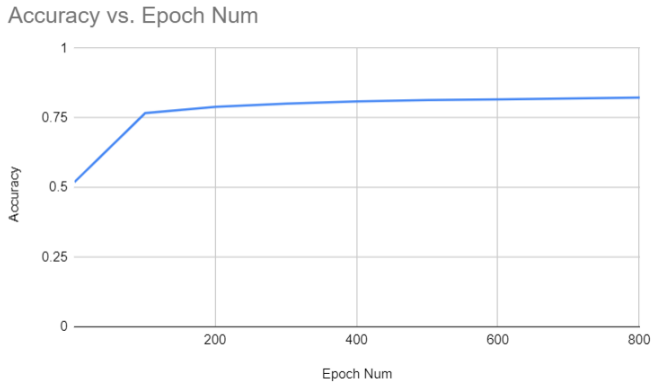

Fig. 5. Accuracy vs. Number of Epoch

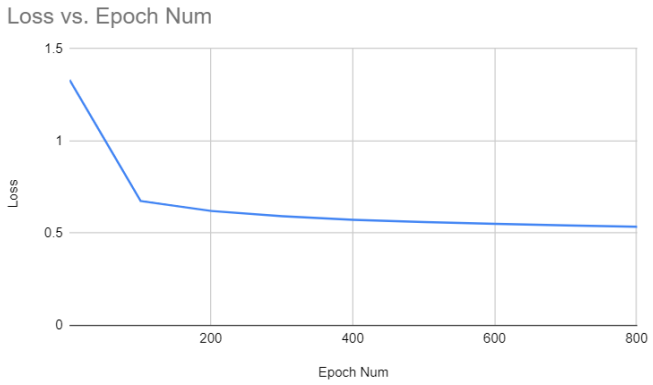

Fig. 6. Loss vs. Number of Epoch

TABLE II
TRAINING EVALUATION SCORE

Epoch Num	Loss	Accuracy	F1 Score	Precision	Recall
1	1.3306	0.5194	0.4108	0.6229	0.3151
101	0.6747	0.7669	0.7601	0.7959	0.7275
201	0.6202	0.7897	0.7844	0.8129	0.7579
301	0.5914	0.8009	0.7975	0.8220	0.7744
401	0.5729	0.8087	0.8057	0.8283	0.7844
501	05598	0.8136	0.8112	0.8321	0.7914
601	0.5509	0.8162	0.8139	0.8343	0.7946
701	0.5419	0.8195	0.8175	0.8365	0.7995
801	0.5349	0.8227	0.8205	0.8387	0.8030

IV. APPENDIX

TABLE III
DATASET HEAD

Data Set Samples		
ID	**Phrase**	**Sentiment**
0	A series of escapades demonstrating the adage ...	1
1	A series of escapades demonstrating the adage ...	2
2	A series	2
3	A	2
4	series	2
5	of escapades demonstrating the adage that what...	2
6	of	2
7	escapades demonstrating the adage that what is...	2
8	escapades	2
9	demonstrating the adage that what is good for ...	2
10	demonstrating the adage	2
11	demonstrating	2
12	the adage	2
13	the	2
14	adage	2
15	that what is good for the goose	2
16	that	2
17	what is good for the goose	2
18	what	2
19	is good for the goose	2
20	is	2
21	good for the goose	3
22	good	3
23	for the goose	2
24	for	2
25	the goose	2
26	goose	2
27	is also good for the gander , some of which oc...	2
28	is also good for the gander , some of which oc...	2
29	is also	2
30	also	2
31	good for the gander , some of which occasional...	2
32	for the gander , some of which occasionally am...	2
33	the gander , some of which occasionally amuses...	1
34	the gander ,	2
35	the gander	2
36	gander	2
37	,	2
38	some of which occasionally amuses but none of ...	2
39	some of which	2
40	some	2
41	of which	2
42	which	2
43	occasionally amuses but none of which amounts ...	2
44	occasionally	2
45	amuses but none of which amounts to much of a ...	2
46	amuses	3
47	but none of which amounts to much of a story	1
48	but	2
49	none of which amounts to much of a story	2