Multi-class Sentiment Analysis using Deep Learning

Instructor: Dr. T.Akilan
Professor at Lakehead University

Dhairya Shaileshbhai Patel
Student Id: 1110149
Lakehead University

***Abstract*—Sentiment analysis is a popular subject of study in social networks today.Nevertheless, most state-of - the-art work and study into the automated analysis of sentiment of texts from social networks and micro-blogging platforms is focused on either the binary classification or the ternary classification. Here I have used The movie review repository of Rotten Tomatoes from GitHub. This CNN model is build up by 2 layers and ReLU as an activation function. I have tried different values of pre-processing steps along with multiple values of hyper-parameters to get the best accuracy and figure-of-merit score. I have also tried multiple activation functions like ReLU, sigmoid, softmax etc. Thus, best model is build up.**

***Index Terms*—Multi-class Sentiment analysis, Convolution Neural Network(CNN), Keras**

I. Introduction

To start with, what is Sentiment Analysis? Models of Sentiment analysis identify polarity inside a text-whether it's a whole paper, an article, a phrase, or a clause-e.g. a positive or a negative review.Sentiment analyses utilizes numerous techniques and algorithms such as rules-based programs for the processing of natural languages (NLP), which conduct sentiment analysis based on a set of manual laws. Automatic systems that rely on machine learning techniques. A hybrid structures which uses both rule-based and automatic system.As I used Deep Learning as including CNN in this model, let's take a quick glance at CNN. CNN is a multi-layer neural network which can use different activation functions as per the requirement. Here, Rotten Tomatoes movie review dataset is used for sentiment analysis. Dataset contains 0 to 4 type of sentiment. I have used this dataset and build my model using Term Frequency-Inverse Dense Frequency(TF-IDF).

II. Literature Review

A. *Working of sentiment analysis:*

This model learns in the training phase (A), based on examples used for the testing, to integrate a specific input (i.e. a text) with the corresponding output (tag). The extractor method converts the text data to a feature vector. The machine learning algorithms are fed pairs of feature vectors and tags (e.g. positive, negative or neutral) to construct a prototype.
The feature extractor is used in prediction (B) to convert unknown text inputs into feature vectors. Such feature vectors are then fed to the model that creates predicted tags (positive, negative, or neutral).

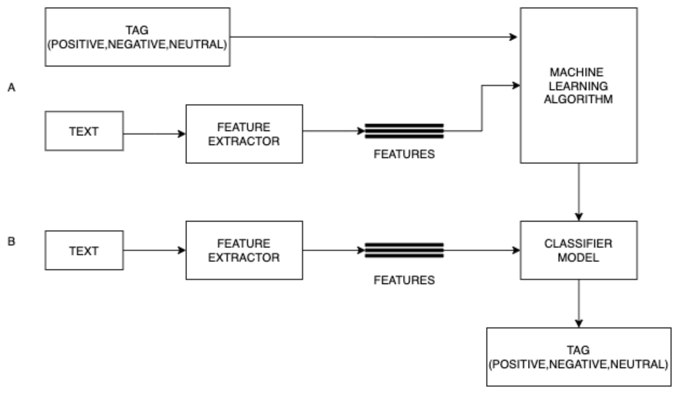


Fig. 1. sentiment analysis

Feature Extraction from Text:
The first step in a classification scheme is to transform the extracted or vectorized text of language or document, so the traditional solution is bag-of-words or bag-of-ngrams.
New extraction strategies focused on word embedding (also known as word vectors) have more recently been implemented. These representations allow related terms to be interpreted in a specific way, which may enhance the efficiency of classifiers.

B. *Vectorization methods*

1) Bag of words(BoW): BoW converts text into the matrix of occurrence of words within a document. The more frequent a word, the more important it might be. It is way to determine the significant words in a text. A binary form of BoW model concerns about whether given words occurred or not in the document.
Limitations of BoW:
1.Vocabulary: It requires careful design, most specifically to manage the size which impacts the sparsity of the document representations.
2.Sparsity: Sparse representations are harder to model both for computational and for information reasons.
3.Meaning: Discarding word order ignores the context, and in turn meaning of words in the document (semantics).

2) TF-IDF: Allows you to determine the most important words in each document. Each corpus may have shared words beyond just stop words. These words should be down weighted in importance. Ensures most common words don't show up as key words. Keeps document specific frequent words weighted high.

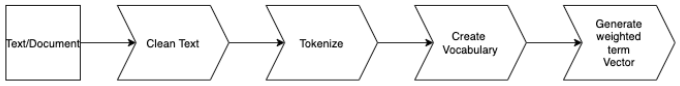


Fig. 2. TF-IDF

TF-IDF focuses on finding the 'valuable' classification key words for each document. TF of a word in a document increases its value, and DF decreases the it.The vector of these key words is then fed to deep learning models as features in order to determine the topic of the documents that contains the key word. Equation of TF-IDF is in the Appendix V.1.

3) Word2Vec: It generates vector for each word occurs in text.Word2vec is great for digging into documents and identifying content and subsets of content.Word2vec represents each word's context, the n-grams of which it is a part

III. Proposed model

A. Dataset

The film review repository of Rotten Tomatoes is a series of film reviews from Pang and Lee. This is a corpus examined where the tree structure parses each paragraph, with a fine-grained feeling mark of 0 to 4 allocated to the node where the numbers are respectively worst, bad, moderate, good and the best.

TABLE I
Detail about Dataset

Attribute Name	Data type	Description
PhraseID	Float64	numbering of the each sentence in incremental order.
SentenceID	Int64	ID of the unique sentence.
Phrase	Object	reviews of the dataset.
Sentiment	Int64	classification of sentiment in 0 to 4.

B. Libraries

Keras: It is used as an Open-source neural netowrk, which is used in applications such as computer vision and natural language processing. Designed to enable fast experimentation with deep neural networks. I have used these libraries along with Keras:
• Pandas
• Sklearn
• Numpy
• Nltk

C. Data Pre-processing

I have used Rotten tomatoes dataset as comma separated values file from GitHub URL. The model I build up takes the array as an input. So to convert the dataset into numpy array, sklearn library is used. As the dataset contains so many unnecessary data, cleaning must be done before execution. To refine the dataset, I have taken only complete sentences from the dataset. This step led me to the small clear dataset of around 8K entries only. After reduction in the database, these pre-processing steps occurs:
1. Tokenization: It is basically deriving the each word (tokens) from the sentence.
2. Removal of stop-words: removing the useless words from the sentence which doesn't make any sense.
3. Removal of Punctuation: removing unnecessary symbols like ?, _, !, : etc.
4. Normalization: using stemming and lemmatization, words can be converted in its original form.
5. Vectorization: the data has to be converted into vector format. TF-IDF is used here for vectorization.

D. pooling layers

The output of ReLU activation function becomes the input for the pooling layer. There are three types of pooling:

- Sum Pooling
- Avg Pooling
- Max Pooling

In most of the cases, majorly max pooling is used. After that we flatten our matrix vectors feed into the fully connected neural network. Finally, we have an activation function such as softmax to classify the outputs.

E. Coding

Firstly, I have load the dataset through a URL link and then taken only full sentences among the whole dataset. The reason behind it to improve accurate result.
secondly, data pre-processing steps done as I discussed earlier. Finally, I chose ReLU and softmax activation function. After that I added the CNN layers(dense,flatten,max-pooling layers) for processing. multiple epochs have trained. Thus by creating metrics, I evaluate the final results through testing.

F. over fitting

The most critical problem is overfitting in the area of machine learning. over - Fitting is done when the model is fully configured in the testing dataset, but not adequately checked. To trying to solve the issue of unnecessary padding, I slowly raised the amount of layers until it does't show error in any circumstances.

G. Optimizers

When the drop-out layer is used to tackle problems of over-fitting, I find that the accuracy of the test dataset is that. In the following table we can see that the ADADELTA, ADAM and ADAGRAD optimizers outperform ADADELTA in all situations for varying learning levels. ADADELTAs and ADAGRAD optimizers have been used to calculate the optimal performance.

H. Experimental Results

Below Table 2, 3 and Table 4 are showing experimental results.

TABLE II
COMPARISON OF DIFFERENT MODELS

Pre-processing steps	remove-stopwords	use-stemming	use-lemma	remove-puncs	Accuracy
model_1	true	true	true	true	58.7
model_2	true	false	false	true	60.5
model_3	true	false	true	true	61.9

TABLE III
COMPARISON OF ACTIVATION FUNCTIONS

Pre-processing steps	Activation function	Accuracy
model_1	ReLU	61.9
model_2	sigmoid	60.5
model_3	softmax	61.9

TABLE IV
COMPARISON BETWEEN OPTIMIZERS

model$_no$	Optimizer	Accuracy
model_1	ADAGRAD	60.5
model_1	ADADELTA	61.1
model_1	ADAM	61.5
model_2	ADAGRAD	61.8
model_2	ADADELTA	61.3
model_2	ADAM	61.9
model_3	ADAGRAD	61.5
model_3	ADADELTA	61.9
model_3	ADAM	61.9

I. *Performance Evaluation*

Below Table 6 showing final output of the model. This outputs might be changed if parameters are changed.
Below parameters that can affect the model(Table 5).

TABLE V
PARAMETERS OF THE MODEL

measurement parameters	Result
Batch-size	256
epochs	50
Kernel-size	8
CNN layers	2
Learning rate	0.01
drop-out rate	0.5

TABLE VI
RESULTS

measurement parameters	Result
Accuracy	0.619
Recall	0.551
Precision	0.650
F-1 score	0.593
Test-Loss	0.974

J. *accuracy vs epoch*

As per the plot of graph, it can be elicited that training accuracy increases as epoch level increases. But after one

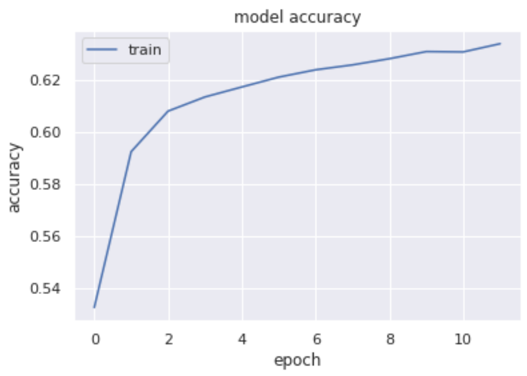
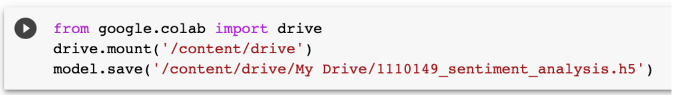


Fig. 3. Training Accuracy with respect to epochs

time, slow growth of accuracy is noticed.

K. *store the model*

I stored the model into the drive after developing the CNN model. I first attach my drive to the colab using the code seen in the figure to save the file. The .h5 extension pattern has been saved.



```
from google.colab import drive
drive.mount('/content/drive')
model.save('/content/drive/My Drive/1110149_sentiment_analysis.h5')
```

Fig. 4. Snippet for storing the model

IV. CONCLUSION

By using CNN for multi-class sentiment analysis, with 2 CNN layers, epoch set to 50 and batch size of 256 is set. Kernel size of 5 with ADADELTA optimizer gives the best performance. This proposed model led to the 61.9% accuracy along with 0.59 F-1 measure score. Test loss is 0.97 with 0.65 precision and 0.55 recall.
As per the experimental results, Implementation considers Modular coding,Over fitting issue,Number of trainable parameters,Train/Test split ratio of 70:30 with random state 2003.
Thus, Aim is achieved in all the manner and that's how CNN for multi-class sentiment analysis works with Keras.

REFERENCES

[1] https://raw.githubusercontent.com/cacoderquan/Sentiment-Analysis-on-the-Rotten-Tomatoes-movie-review-dataset/master/train.tsv

[2] https://adventuresinmachinelearning.com/convolutional-neural-networks-tutorial-in-pytorch/

[3] https://towardsdatascience.com/choosing-the-right-encoding-method-label-vs-onehot-encoder-a4434493149b

[4] https://en.wikipedia.org/wiki/Tf

V. Appendix

A. TF-IDF

Equation to find TF-IDF[4]:

$$W_{i,j} = tf_{i,j} \times log(\frac{N}{df_i}) \tag{1}$$

Where,

$W_{i,j}$	= TF-IDF weight for token i in document j
$tf_{i,j}$	= Number of occurrences of token i in document j
df_i	= Number of documents that has token i
N	= Total number of documents in the training corpus

GITHUB link: https://github.com/dhairyapatel96/sentiment_analysis.git