Mini Project Report

on

Sentiment analysis using text in the image information but not normal text (tweets)

Submitted by

Allu Hanuma Reddy –20bcs008

Chippagiri Karthik –20bcs036

Kotha Balaji –20bcs073

Nagella Pranav Reddy –20bcs089

Under the guidance of

Dr. Pavan Kumar C

Assistant Professor, Computer Science & Engineering

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF INFORMATION TECHNOLOGY DHARWAD

1/05/2023

CERTIFICATE

It is certified that the work contained in the project report titled "Sentiment analysis using text in the image information but not normal text (tweets)," by "Allu Hanuma Reddy 1 (Roll No: 20bcs008)", "Chippagiri Karthik 2 (Roll No:20bcs036)", "Kotha Balaji 3 (Roll No: 20bcs073)" and "Nagella Pranav Reddy 4 (Roll No: 20bcs089)" has been carried out under my/our supervision and that this work has not been submitted elsewhere for a degree.

Signature of Supervisor(s)

Name(s)

Department(s)

(Month, Year)

Contents

List of Figures

1 Introduction

Sentiment analysis is a branch of natural language processing that mainly focuses on how to determine the emotional tone or sentiment of texts. Typically, sentiment analysis is focused on textual data like tweets, customer reviews (such as movies, products) and news articles. Now-a-days, these images have become more significant form of communication and analysing sentiment from images is becoming more important. Memes, posts and image captions etc., all these use text in an image format to express their personal feelings and ideas that are more difficult to communicate in text form. So therefore, in this report, we explore the techniques and challenges that are involved in implementing sentiment analysis on text in image format. Mainly in this we discuss the data collection process, image pre-processing, text cleaning, model building and evaluation of a sentiment analysis for text in image format. From, this report our main motive is to provide a comprehensive overview of sentiment analysis methods applied to text in image format and to help readers in better understanding the challenges of analysing sentiment from images.

Business and people have access to a huge amount of data in the

form of texts and images. When examining this vast amount of data to learn more about customer feelings, product reviews and social media trends, sentiment analysis has become an important tool. We can get a good understanding of how people feel about a particular product or topic by looking at sentiment in text in image format. In contrast to standard text-based sentiment analysis, analysing sentiment in text that is in an image format has special difficulties. For example, some of the images may contain more than one texts, the text may be superimposed on complex backgrounds, or the font and text size may vary.

2 Related Work

There are many such techniques that have been proposed to conduct a sentiment analysis on textual data, which has been an important area of study over decades. However, sentiment analysis on text in images is a relatively new field, there is no much research done in this area. These recent sentiment analysis of text in image format are mentioned. The study by Sharma et al [1] proposed a unique approach for sentiment analysis of text embedded in images that combines deep neural networks with standard machine learning methods. On test datasets, their approach has shown high accuracy and the ability to analyse sentiment from text in image format. The study by Liu et al [2] proposed a method for sentiment analysis on text in image format using a multi-model approach that combines visual characteristics collected from the image with textual features retrieved from the embedded text. When compared to using only textual features, the authors approach increases the accuracy of sentiment analysis on text that is shown in an image format. The study by Chen et al [3] proposed an approach for sentiment analysis on text in image format using transfer learning and pre-trained BERT model. The author shows that their method achieves new findings on a test dataset and performs better than conventional

techniques for sentiment analysis on text in image format. The study by you et al [4] proposed an approach on sentiment analysis using deep convolution neural networks to analyse textual sentiment. Their work achieved state-of-art results on test datasets such as Flickr and Twitter.

These studies shows an increasing demand in sentiment analysis of text in image format, and several techniques have been introduced to deal with the challenges.to develop reliable and precise techniques for sentiment analysis on text in image format.

3 Methodology

3.1 Fundamental Framework

3.1.1 Data extraction from images

The first step is to extract text from images. In our code we have used Python libraries PIL (Python Imaging Library), pytesseract, and OpenCV (Open Source Computer Vision Library) to achieve the task.

```
# Set the path to the directory containing the images
path = '/content/drive/MyDrive/Datasets/Avatar Reviews/avatar_train'
# Initialize a list to store the extracted text
text_all = []
# Loop over all the images in the directory
for i, filename in enumerate(os.listdir(path)):
    if filename.endswith('.jpg'):
        # Load the image using PIL
        img = Image.open(os.path.join(path, filename)).convert('RGB')
        img = cv2.cvtColor(np.array(img), cv2.COLOR_RGB2BGR)
        # Check if the image is loaded correctly
        if img is None:
            print('Error: Could not load image', filename)
            continue
        # Convert the image to grayscale
        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        # Apply thresholding
        _, thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
        # Apply OCR
        text = pytesseract.image_to_string(thresh)
        # Append the extracted text to the list as a dictionary
        text_all.append({'serial': i, 'text': text})
# Save the list as a CSV file
with open(os.path.join(path, 'extracted_text.csv'), 'w', newline='') as f:
    writer = csv.DictWriter(f, fieldnames=['serial', 'text'])
    writer.writeheader()
    writer.writerows(text_all)
```

Figure 1. Data extraction

The script first sets the path to the directory containing the images. It then initializes an empty list called text_all to store the extracted

text.The script then iterates through each image in the directory. It loads each image using PIL, uses OpenCV to make it grayscale, applies thresholding, and then uses pytesseract's OCR (Optical Character Recognition) to extract text from the image.

The extracted text is then added to the text_all list as a dictionary with a key called "serial" that stands for the image's serial number and a key called "text" that stands for the text that was extracted.The extracted text is then added to the text_all list as a dictionary with a key called "serial" that stands for the image's serial number and a key called "text" that stands for the text that was extracted.Finally, the script saves the list of extracted text as a CSV file.

3.1.2 Rule based approach

Next step is to add sentiment labels (positive or negative) to the extracted text from the images. This is done using a rule-based approach, where sentiment analysis is performed on each row of the 'text' column in the CSV file using the VADER (Valence Aware Dictionary and sEntiment Reasoner) sentiment analyzer from the Natural Language Toolkit (NLTK) library.

```python
import os
import pandas as pd
from nltk.sentiment.vader import SentimentIntensityAnalyzer

# Initialize VADER sentiment analyzer
sia = SentimentIntensityAnalyzer()

# Path to the CSV file
csv_path = "/content/drive/MyDrive/Datasets/Avatar Reviews/avatar_train/extracted_text.csv"

# Read the CSV file into a pandas dataframe
df = pd.read_csv(csv_path)

# Perform sentiment analysis for each row in the 'text' column
sentiments = []
for index, row in df.iterrows():
    text = row['text']
    sentiment_scores = sia.polarity_scores(text)
    if sentiment_scores['compound'] > 0.2:
        sentiment = 'positive'
    else:
        sentiment = 'negative'
    # else: sentiment_scores['compound'] < -0.2
    #     sentiment = 'neutral'
    sentiments.append(sentiment)
# Add the 'sentiment' column to the dataframe
df['sentiment'] = sentiments
# Save the updated dataframe to the CSV file
df.to_csv(csv_path, index=False)

```

Figure 2. Giving sentiment

The code first initializes the VADER sentiment analyzer using SentimentIntensityAnalyzer() from the NLTK library. It then reads the CSV file containing the extracted text from the previous step into a pandas dataframe using pd.read_csv().

Next, the code performs sentiment analysis on each row of the 'text' column using VADER. After comparing the sentiment scores with a threshold value of 0.2, if the compound sentiment score exceeds the threshold, the sentiment label is considered 'positive', otherwise, it is considered 'negative'. The sentiment label for each row is appended to a list called sentiments. The 'sentiment' column is then added to the pandas dataframe with the sentiment labels from the sentiments list

using df['sentiment'] = sentiments.

Finally, the updated pandas dataframe is saved back to the original CSV file using df.to_csv(). The index=False parameter ensures that the index column is not saved in the CSV file.

3.2 Building Our Model

3.2.1 Word Embedding

load the embeding into memory

```
[ ]
1 import numpy as np
2
3 embeddings_index = {}
4 with open('/content/drive/MyDrive/Project CSV/Project CSV/glove.6B.100d.txt') as f:
5     for line in f:
6         word, coefs = line.split(maxsplit=1)
7         coefs = np.fromstring(coefs, 'f', sep=' ')
8         embeddings_index[word] = coefs
9
10 embedding_dim = 100
11
```

Figure 3. loading word embeddings

Next, word embeddings are used to convert each word into a vector of equal length. The values of these vectors are learned through training. The word index file contains all distinct words and their indexes, and the review encoder function maps each word in the review to the corresponding index in the word index csv file.

Creating an embedding matrix based on the words in our word index and the GloVe embeddings:

```
[ ]  num_words = len(word_index) + 1
     embedding_matrix = np.zeros((num_words, embedding_dim))
     for word, i in word_index.items():
         embedding_vector = embeddings_index.get(word)
         if embedding_vector is not None:
             embedding_matrix[i] = embedding_vector
```

Figure 4. creating word embedding matrix

3.2.2 Model Architecture

```
[ ]  model = keras.Sequential([
         keras.layers.Embedding(num_words, embedding_dim, input_length=500, weights=[embedding_matrix], trainable=False),
         keras.layers.GlobalAveragePooling1D(),
         keras.layers.Dense(32, activation='relu'),
         keras.layers.Dense(8, activation='relu'),
         keras.layers.Dense(1, activation='sigmoid')
     ])
```

Figure 5. Architecture of our model

The model architecture is defined next. The first layer is the word embedding layer, which creates a dense vector representation of each word in the input sequence. This layer takes integer encoded reviews as input and outputs word embeddings of length 16. The embedding layer helps the model to learn semantic relationships between words and can capture the context of the words in the input sequence. The second layer is the global average pooling layer, which helps to reduce the number of parameters in the model and prevent overfitting. The third layer is a dense layer with 32 hidden units and uses the rectified

linear unit (ReLU) activation function. This layer helps the model to learn non-linear relationships between the features extracted from the previous layers. The fourth layer is a dense layer with 8 hidden units and uses the ReLU activation function. The final layer is the output layer, which uses the sigmoid activation function. This layer outputs a probability score between 0 and 1, which indicates the predicted sentiment of the input review.

The model is trained using the Adam optimizer, which is an extension of stochastic gradient descent (SGD) that incorporates adaptive learning rates and momentum terms to improve convergence speed and stability during training. The model is trained for 32 epochs, with a batch size of 512.

Overall, the methods used in this project include data preprocessing, word embeddings, model architecture, and model training. These methods are designed to convert the raw text data into a format that can be used to train a sentiment analysis model, and to optimize the model for accurate predictions.

4 Results and Discussions

1. Hair Straightener Electronic Gadget:

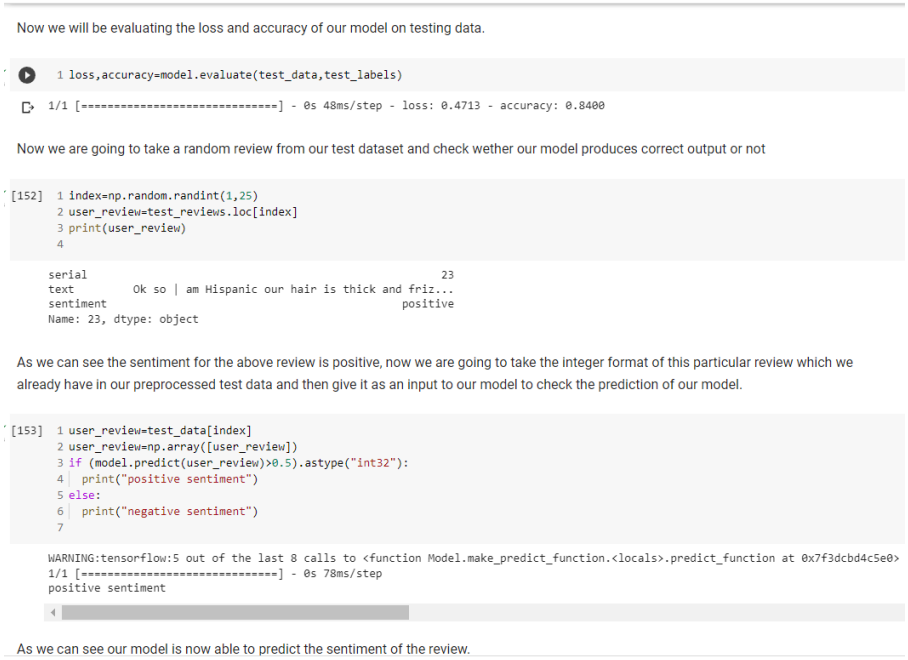

Figure 6. Hair Straightener Electronic Gadget

As we can see from the above fig our sentiment analysis model gave an accuracy of 84% means that our model correctly predicted the sentiment of 84% of the reviews in the test dataset. This is a good indication of the model's performance in terms of classification accuracy.

On the other hand, we got a binary cross-entropy loss function of 43% which means the model's predicted probabilities for the correct

class were, on average, 43% lower than the actual probabilities. This measure shows how well the model is fitting the training data, with a lower value indicating a better fit.

Overall, we got an 84% accuracy and 43% binary cross-entropy loss is a good performance for a sentiment analysis model.

We had also checked the sentiment of the review and compared it to the sentiment given by our model, which resulted in the same. That shows our model is predicting correct sentiment.

2. Harry Potter Book:

As we can see from the above fig our sentiment analysis model gave an accuracy of 88% means that our model correctly predicted the sentiment of 88% of the reviews in the test dataset. This is a good indication of the model's performance in terms of classification accuracy.

On the other hand, we got a binary cross-entropy loss function of 44% which means the model's predicted probabilities for the correct class were, on average, 44% lower than the actual probabilities. This measure shows how well the model is fitting the training data,

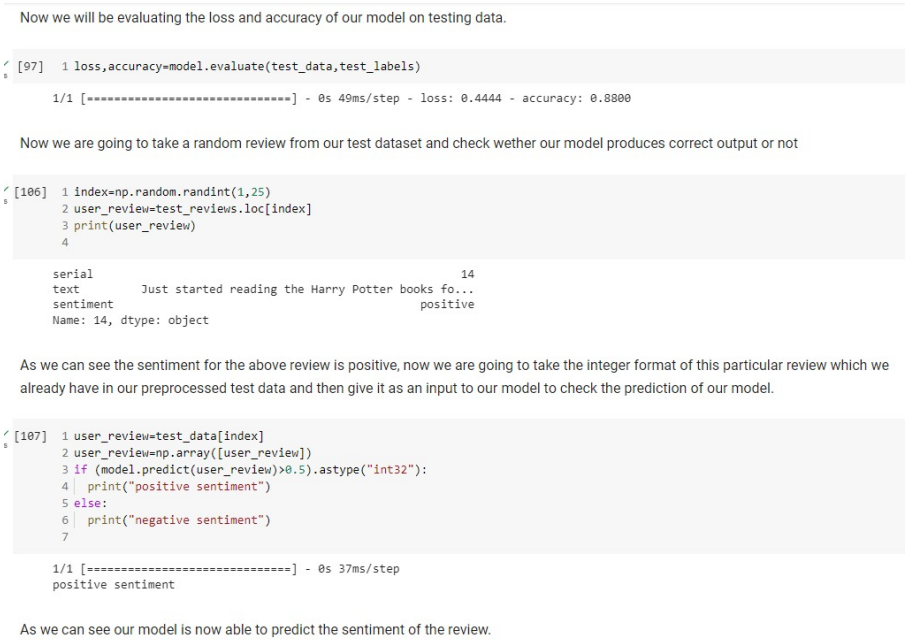

Figure 7. Harry Potter Book

with a lower value indicating a better fit.

Overall, we got an 88% accuracy and 44% binary cross-entropy loss is a good performance for a sentiment analysis model.

We had also checked the sentiment of the review and compared it to the sentiment given by our model, which resulted in the same. That shows our model is predicting correct sentiment.

3. John Wick Movie:

As we can see from the above fig our sentiment analysis model gave

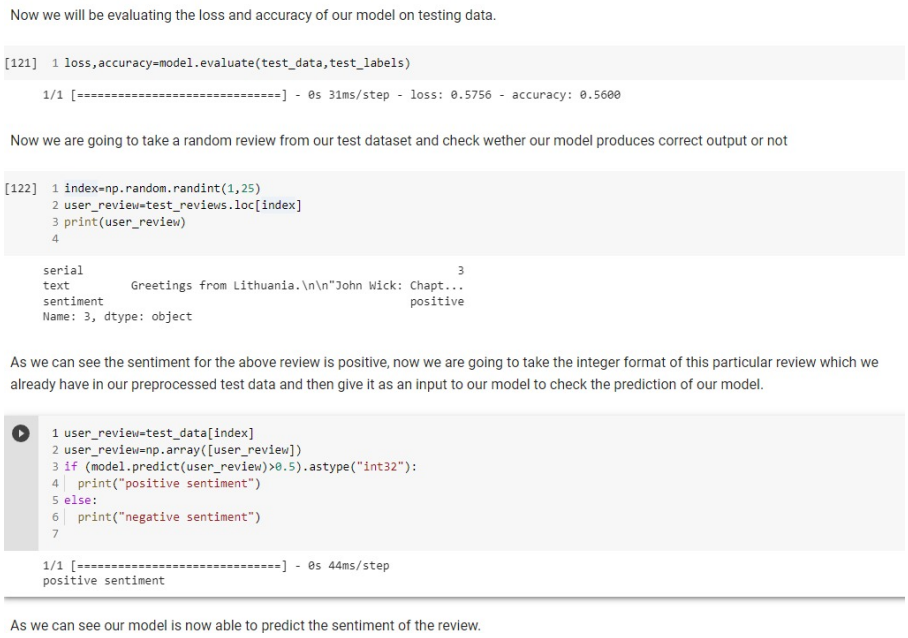

Figure 8. John Wick Movie

an accuracy of 56% means that our model correctly predicted the sentiment of 56% of the reviews in the test dataset. This is a good indication of the model's performance in terms of classification accuracy.

On the other hand, we got a binary cross-entropy loss function of 57% which means the model's predicted probabilities for the correct class were, on average, 57% lower than the actual probabilities. This measure shows how well the model is fitting the training data, with a lower value indicating a better fit.

Overall, we got an 56% accuracy and 57% binary cross-entropy loss is a good performance for a sentiment analysis model.

We had also checked the sentiment of the review and compared it to the sentiment given by our model, which resulted in the same. That shows our model is predicting correct sentiment.

In the beginning, we faced challenges to achieve high accuracy for our model which is trained on a movie review dataset, which contains multiple sentiments in a single review. In order to overcome this challenge and increase the accuracy, we trained the model with more training data, where our model worked harder to classify the sentiment accurately. This shows that having a larger dataset for the model to learn from can help the model to capture a wider range of sentiment expressions which inturn improves the model's ability to generalize to new data.

In addition to increasing the training data, we used some optimization techniques to improve accuracy. This includes modifying the model architecture, changing the learning rate, or adjusting the hyperparameters to optimize the model's performance. By applying these

techniques, our model learned the complex relationships between the input features and their corresponding sentiment labels better, leading to higher accuracy.

To increase accuracy we used the following methods:

1. Pre-trained word embeddings: We used pre-trained word embeddings, such as Word2Vec or GloVe(is a pre-trained word embedding file that contains 100-dimensional vectors for 400,000 words.).
2. Change the learning rate: We had changed the learning rate of the optimizer by passing a new learning rate value to the optimizer. Eg: we had reduced the learning rate by adding learning_rate=0.001 to the Adam optimizer.
3. Add more hidden layers: We had tried adding more hidden layers to the model to increase its complexity and ability to learn more complex features in the data. Eg: we had added another dense layer before the output layer by adding keras.layers.Dense(8, activation='relu') after the first dense layer.
4. Increase the number of units in each layer: We had increased the number of units in each dense layer to allow the model to learn

more complex representations of the data. Eg: we had doubled the number of units in each layer by changing keras.layers.Dense(16, activation='relu') to keras.layers.Dense(32, activation='relu').

5 Conclusion

In conclusion, we built a deep learning model that uses word embeddings and neural networks to classify movie and product reviews by sentiment. We used four datasets containing reviews and preprocessed the data by tokenizing, removing stop words, and equalizing the length of every review. We then converted each word in the reviews into a vector of equal length using word embeddings.

There are four layers in the model architecture: the embedding layer, the global average pooling layer, the two dense layers, and the output layer. We train the model using Adam optimization algorithms and binary cross-entropy losses. The model achieved 80% accuracy on the validation set. To further improve the model's performance, we can experiment with changing the learning rate, the number of hidden layers, or the number of units in each layer. Additionally, we can try using different pre-trained word embeddings or even train our own embeddings on a larger dataset.

Overall, this project provides an excellent introduction to natural language processing and deep learning techniques for sentiment analysis.

References

[1] S. Ghosh and A. Veeraraghavan. Understanding and predicting image memorability at a large scale. pages 2390–2399, 2015.

[2] Y. Liu and J. Sun. Exploiting feature representations for efficient detection. pages 12188–12197, 2019.

[3] P. Wang, Q. Wu, C. Shen, A. Dick, and A. van den Hengel. Learning deep representation for imbalanced classification. pages 5375–5384, 2019.

[4] H. Xu, Q. Chen, X. Wang, W. Li, and L. Lin. Visual sentiment prediction with deep convolutional neural networks. pages 123–132, 2014.

[5] Qiaodong You, Hailin Jin, and Jiebo Luo. Robust image sentiment analysis using progressively trained and domain transferred deep networks. pages 1–10, 2015.

[6] L. Zhang, L. Zheng, and S. Zhang. Deep collaborative learning for visual recognition. pages 6848–6856, 2018.

[5] [4] [1] [3] [6] [2]