CMPEN 497

Exploring Text Analysis Using Self Organizing Maps and Convolutional Neural Networks

Ameya Hampihallikar

17 December 2020

Abstract

Convolutional Neural Networks (CNN)

A convolutional neural network (CNN) is a type of artificial neural network which is derived from the Multiple Layer Perceptron model (MLP). The main architecture or design of such a neural network revolves around using multiple layers so that it can learn different features of the input space using backpropagation [2]. The CNN will be built by using different convolutional layers. A convolutional layer will use an input of weights to sequentially apply the filter across the input to get a feature map [3]. This convolutional layer is often followed by a pooling layer. The main function of this pooling layer is for downsampling the feature map to retain only the important features. This is an important step as it helps the network to avoid overfitting by not being susceptible to small changes in feature maps between different inputs.

Self-Organizing Maps (SOM)

A self-organizing Map (SOM) is a type of neural network mainly used for dimensionality reduction which is useful for clustering. It involves using unsupervised learning to organize inputs into clusters, with similar input being found in the same region. The SOM algorithm is based on the winner-takes-all approach [5]. Each node's weights are randomly initialized and compared to an input vector. The node with the most similar weights to that of the input vector is the 'winner' and is 'rewarded' by updating its weigh values along with its neighbors' weight values. This process is repeated for multiple epochs until the similar nodes get grouped together and unsimilar nodes get separated.

Introduction

Text classification and clustering involves the process of tagging or labelling of text or documents into categorized groups. It is a popular technique and uses Natural language Processing concepts for text analysis. Text classification and clustering have several use cases and form a fundamental part of businesses for gaining insights from large amounts of data. For example, businesses can leverage the power of natural language processing for sentiment analysis of customer feedback to understand their customer base better to provide better services. Text classification can also be used to automate repetitive tasks, like order fulfillment, supplier lists etc. which contain data that can be tagged or labelled.

For this project, the aim will be to explore different techniques of text analysis using the concepts learned during the course. The two that I have chosen are Self Organizing Maps (SOM) and Convolutional Neural Networks (CNN); a type of MLP. The CNN will use a neural network Architecture similar to that of a MLP with

multiple layers to emulate different filtering techniques in order to classify text. On the other hand, the SOM approach will use unsupervised learning to produce a lower dimension mapping or clustering of the text/documents. This can be useful to visualize the number of "topics" in a dataset based on the clustering given unknown initial groupings [4].

This project will be worked on using Python primarily. The datasets will be chosen using open-source websites like Kaggle. Additionally, Python libraries like numpy, scikit-learn, keras, spacy etc. will be used for data preprocessing and implementation of different neural network architectures.

Methodology

After initial proposal of the project, research was done to study the different approaches for text analysis with a focus on classification and clustering. The project will focus on two techniques (SOM and CNN) to show the difference in performance between supervised and unsupervised approaches.

Dataset

Both neural network models will be trained and tested on the IMDB movie review dataset published by Stanford University [6]. This is a popular dataset which contains a large corpus of past movie reviews that are categorized into positive and negative. The initial step will be to clean the dataset and remove unwanted columns and data. Following this, the net architecture can be designed, and the dataset can be used for training and testing.

Document Preprocessing

Preprocessing of the data is an important step for text analysis. It involves breaking down the documents in the dataset and retaining the relevant and important features in each.

1. Tokenization

   The first step involves splitting each document into smaller sized chunks called tokens. For this project, each document is split into its lower-cased individual words and punctuation is removed. This is done by using the nltk library's tokenizer class.

2. Removal of Stop Words

   This process is important for feature extraction and reducing the size of the documents. Using the nltk library's English stop word corpus, each tokenized

document is stripped of stop words and single letter words. This will help in improving efficiency and effectiveness of the text analysis. A custom stop word list can also be prepared depending on the requirements.

3. Encoding Documents

   After cleaning the documents by tokenizing and removing the stop words, they need to be encoded using integer sequences. Using the above mentioned Tokenizer class, text in each document is converted to integer sequences using text_to_sequences() method. This is followed by the padding method to ensure all encoded documents have the same vector length.

Word Embeddings

Word embeddings are used to represent words using a dense vector representation. This vector is the projection of a word in a continuous vector space [6]. Here we used a pretrained word embedding set called GloVe developed by Stanford University . It is a improvement from traditional vectorizing algorithms like bag of words by replacing large vectors to represent words with dense vectors.

1. Embedding matrix

   Load the glove.6B.100d.txt file to load the 100-dimensional dense GloVe vectors. Create a matrix of embeddings for each word in the dataset. This is done by looping through the preprocessed documents and creating a mapping of words and weights for words in the dataset [1].

Convolutional Neural Network- Supervised learning approach

Please refer to the CNN_imdb_classification.py file for the implementation of the CNN approach.

Multi-Layer Model

The keras.layers API and documentation was used to create the Neural network model [6].

1. Embedding Layer

   Use the keras.Embedding layer to use the dense 100 dimensional weight vectors obtained from the word embedding matrix.

2. Dropout Layer
   Use the keras.Dropout layer to randomly removing inputs the net to reduce overfitting.

3. Conv1D Layer
   Use the keras.Conv1D layer to create a convolutional kernel that is convolved with the 1D input (text vector) to detect patterns within the text.

4. GlobalAveragePooling Layer

   Use the keras.GlobalAveragePooling to reshape the convolution to reshape the convolved vectors to equal lengths.

5. Dense Layer
   Use keras.Dense to create a fully connected with Sigmoid activation function as this a binary classification problem.

6. Optimizer
   Use keras.compile to set a optimizer and loss function for the net. Here we used the 'adam' optimizer and binary cross entropy. The adam optimizer combines gradient descent and root mean square propagation [2] to update learning rates. The binary cross entropy loss function is used as this is a binary classification task.

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 100, 100)          6126800
_________________________________________________________________
dropout (Dropout)            (None, 100, 100)          0
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 96, 64)            32064
_________________________________________________________________
global_average_pooling1d_2 ( (None, 64)                0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 65
=================================================================
Total params: 6,158,929
Trainable params: 32,129
Non-trainable params: 6,126,800
_________________________________________________________________
```

   Since we are using pretrained word embeddings the number of non-trainable parameters are high as we have fixed weight vectors of the GloVe word embeddings.

Procedure

1. Read the dataset
    a. Create pandas dataframes of positive and negative movie reviews by reading the ‘pos.txt’ and ‘neg.txt’ files from the dataset folder
    b. Create a combined movie review dataset and label positive review as 1 and negative reviews as 0

```
# Read the aclImdb Folder to extract positive and negative movie review and create 2 individual files respectively
read_folder(r'aclImdb\train\neg', r'combined_neg.txt')
read_folder(r'aclImdb\train\pos', r'combined_pos.txt')

# Create two pandas dataframes to store the postive and negative movie reviews
# Postive reviews
df_pos = pd.read_csv(r'combined_pos.txt', sep=";", names=['Text',])
df_pos['Label'] = 1

# negative movie reviews
df_neg = pd.read_csv(r'combined_neg.txt', sep=";", names=['Text',])
df_neg['Label'] = 0

# Combine the two dataframes
merge = [df_pos, df_neg]
df = pd.concat(merge).reset_index(drop=True)
```

2. Complete preprocessing of the dataset:
    a. Tokenizing of the document- tokenizing is done to separate out sentences into individual words and remove punctuations.
    b. Removal of stop words- remove words from the tokenized set that add little or no sentiment value to the overall sentence, For example ‘the’, ‘a’, ‘an’ etc.
    c. Split the dataset into training and validation data.

```
# Function to clean the text by removing punctuations and english language stopwords
from nltk.tokenize import word_tokenize
def clean_text(doc):
    stop_words = set(nltk.corpus.stopwords.words('english'))
    tokenizer = nltk.RegexpTokenizer(r"\w+")

    tokens = word_tokenize(doc)
    tokens = [word.lower() for word in tokens]
    txt = [word for word in tokens if word not in stop_words and (word.isalpha() == True) and (len(word)> 1) and (word != 'br')]
    final_txt = " ".join(map(str, txt))
    return final_txt
```

```
# Create a new dataframe for the cleaned text and call the clean_text function
df_clean = df
df_clean['tokens'] = df_clean.apply(lambda row: clean_text(row['Text']),axis =1)
```

```
#preview the cleaned dataframe
df_clean.head()
```

	Text	Label	tokens
0	Bromwell High is a cartoon comedy. It ran at t...	1	bromwell high cartoon comedy ran time programs...
1	Homelessness (or Houselessness as George Carli...	1	homelessness houselessness george carlin state...
2	Brilliant over-acting by Lesley Ann Warren. Be...	1	brilliant lesley ann warren best dramatic hobo...
3	This is easily the most underrated film inn th...	1	easily underrated film inn brooks cannon sure ...
4	This is not the typical Mel Brooks film. It wa...	1	typical mel brooks film much less slapstick mo...

3. Vectorize text-
    a. Convert tokenized sentence into a vector for representation of text in numerical form before feeding it to the neural network
    b. Use the Keras tokenizer to create integer representation of words
    c. Use padding to create equal length vectors

```
tokenizer = Tokenizer()
tokenizer.fit_on_texts(train['tokens'])
encoded_train_data = tokenizer.texts_to_sequences(train['tokens'])
encoded_train_data_padded = pad_sequences(encoded_train_data, maxlen=100, padding='post')
```

    d. Use the GloVE library as a pretrained word embeddings library. It contains 400,000 pretrained words and will provide a comprehensive dictionary of words for the CNN.
    e. Create the word embedding matrix.

```
# Use the GloVE word wmbeddings file as pretrained word embeddings
word_embeddings = {}
file = open(r'glove.6B.100d.txt', encoding='utf8')
for line in file:
    val = line.split()
    word = val[0]
    coefs = np.asarray(val[1:], dtype = 'float32')
    word_embeddings[word] = coefs
file.close()
print('Number of word embeddings:' + str(len(word_embeddings)))
```

```
Number of word embeddings:400001
```

```
# create a embedding matrix by using words only found in the dataset
words_in_data = len(tokenizer.word_index) + 1
embedding_matrix = np.zeros((words_in_data, 100))
for word, i in tokenizer.word_index.items():
    temp = word_embeddings.get(word)
    if temp is not None:
        embedding_matrix[i] = temp
```

4. Use keras library to build a new CNN.
    a. Testing of models for hidden layers. Popular models include
        i. Embedding layer

ii. Dropout layer
iii. Conv1D layer
iv. GlobalAveragePooling layer
v. Dense layer
vi. Optimizer – adam, loss function- binary cross entropy

b. Evaluate model using testing data for 50 epochs and provide a validation set to test for validation loss and accuracy.

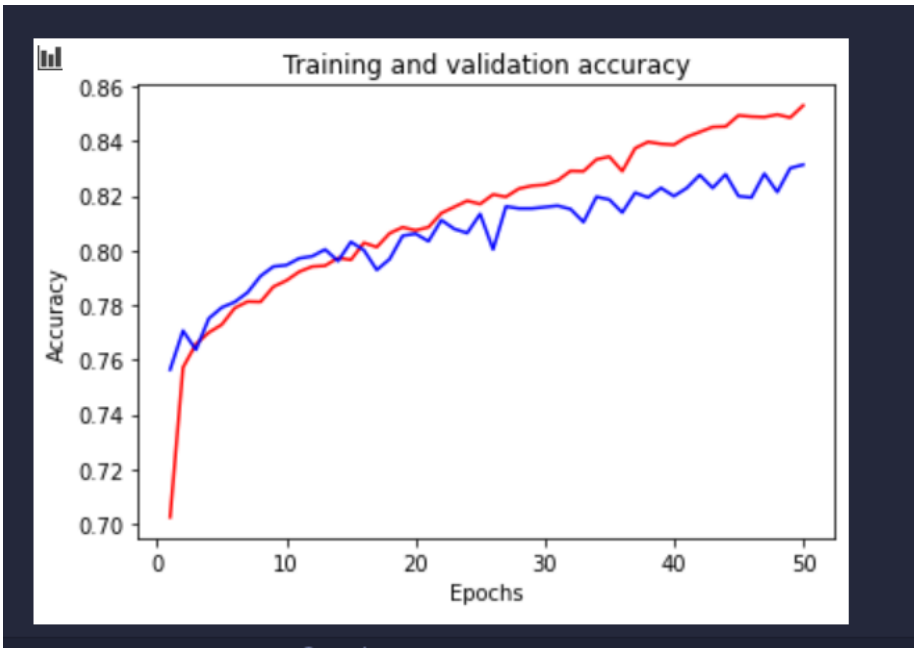
Results

Calculate the training and validation loss and accuracy using keras.evaluate function. The model was trained and tested over 50 epochs with batch size of 128 using the training dataset and a validation set.

After 50 epochs the accuracy of the neural network was 86.29%

Training accuracy – red

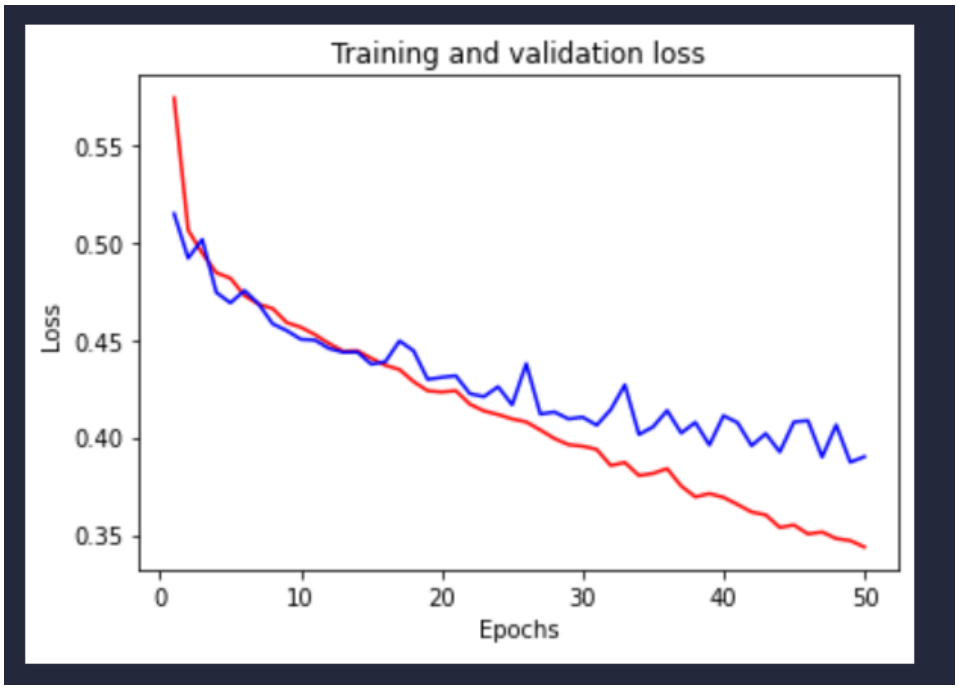
Validation accuracy – blue



The training accuracy is increasing with each epoch (favorable) while the validation accuracy seems to peak and gradually smoothen out. This might be the result of overfitting of the data.

Training loss – red

Validation loss– blue



The training loss is decreasing with each epoch (favorable) while the validation loss seems to peak and gradually smoothen out. This might be the result of overfitting of the data.

Observations

1. The accuracy of this model is around 86%.

2. From the results we can observe that in the calculation of validation loss and accuracy some overfitting was found.

Form the above observations, we can seemingly improve the net by testing different parameters in the different keras layers, increase number of epochs or keep changing validation data so the net does not overfit to particular data. These different testing methods can be used to improve the accuracy of the net.

Dimensionality Reduction

The vectorized text and word embedding matrix provides us with a high dimension representation of the documents and text. These can be viewed as large feature spaces. For the purpose of clustering the dimensionality of these feature spaces must be reduced [9].

1. Principal Component Analysis (PCA)
   Using the sklearn.PCA library, linear dimensionality reduction is performed to project the data onto a 2D coordinate plane.

Self-Organizing Map- Unsupervised learning approach

Please refer to the SOM_imdb_classification.py file for the implementation of the SOM approach.

MiniSom

MiniSom is a python library that uses the SOM winner-takes-all competitive learning algorithm to calculate the winning nodes and weight updating [7].

Use the vectorized documents and word embeddings whose dimensionality has been reduced using PCA as an input to the SOM. This will help produce a 2D representation of the respective feature spaces.

Visualize the clusters using documentation [7] provided on the website that plots the results of the clustering done by the SOM.

Procedure

1. Read the dataset
   a. Create pandas dataframes of positive and negative movie reviews by reading the 'pos.txt' and 'neg.txt' files from the dataset folder
   b. Create a combined movie review dataset and label positive review as 1 and negative reviews as 0

```
# Read the aclImdb Folder to extract positive and negative movie review and create 2 individual files respectively
read_folder(r'aclImdb\train\neg', r'combined_neg.txt')
read_folder(r'aclImdb\train\pos', r'combined_pos.txt')

# Create two pandas dataframes to store the postive and negative movie reviews
# Postive reviews
df_pos = pd.read_csv(r'combined_pos.txt', sep=";", names=['Text',])
df_pos['Label'] = 1

# negative movie reviews
df_neg = pd.read_csv(r'combined_neg.txt', sep=";", names=['Text',])
df_neg['Label'] = 0

# Combine the two dataframes
merge = [df_pos, df_neg]
df = pd.concat(merge).reset_index(drop=True)
```

2. Complete preprocessing of the dataset-
    a. tokenizing of the document- tokenizing is done to separate out sentences into individual words and remove punctuations.
    b. Removal of stop words- remove words from the tokenized set that add little or no sentiment value to the overall sentence, For example ‘the’, ‘a’, ‘an’ etc.

```
# Function to clean the text by removing punctuations and english language stopwords
from nltk.tokenize import word_tokenize
def clean_text(doc):
    stop_words = set(nltk.corpus.stopwords.words('english'))
    tokenizer = nltk.RegexpTokenizer(r"\w+")

    tokens = word_tokenize(doc)
    tokens = [word.lower() for word in tokens]
    txt = [word for word in tokens if word not in stop_words and (word.isalpha() == True) and (len(word)> 1) and (word != 'br')]
    final_txt = " ".join(map(str, txt))
    return final_txt
```

```
# Create a new dataframe for the cleaned text and call the clean_text function
df_clean = df
df_clean['tokens'] = df_clean.apply(lambda row: clean_text(row['Text']),axis =1)
```

```
#preview the cleaned dataframe
df_clean.head()
```

	Text	Label	tokens
0	Bromwell High is a cartoon comedy. It ran at t...	1	bromwell high cartoon comedy ran time programs...
1	Homelessness (or Houselessness as George Carli...	1	homelessness houselessness george carlin state...
2	Brilliant over-acting by Lesley Ann Warren. Be...	1	brilliant lesley ann warren best dramatic hobo...
3	This is easily the most underrated film inn th...	1	easily underrated film inn brooks cannon sure ...
4	This is not the typical Mel Brooks film. It wa...	1	typical mel brooks film much less slapstick mo...

3. Vectorize text-
    a. Convert tokenized sentence into a vector for representation of text in numerical form before feeding it to the neural network
    b. Use the Keras tokenizer to create integer representation of words
    c. Use padding to create equal length vectors

```
tokenizer = Tokenizer()
tokenizer.fit_on_texts(train['tokens'])
encoded_train_data = tokenizer.texts_to_sequences(train['tokens'])
encoded_train_data_padded = pad_sequences(encoded_train_data, maxlen=100, padding='post')
```

    d. Use the GloVE library as a pretrained word embeddings library. It contains 400,000 pretrained words and will provide a comprehensive dictionary of words for the CNN.
    e. Create the word embedding matrix.

```
# Use the GloVE word wmbeddings file as pretrained word embeddings
word_embeddings = {}
file = open(r'glove.6B.100d.txt', encoding='utf8')
for line in file:
    val = line.split()
    word = val[0]
    coefs = np.asarray(val[1:], dtype = 'float32')
    word_embeddings[word] = coefs
file.close()
print('Number of word embeddings:' + str(len(word_embeddings)))
```

```
Number of word embeddings:400001
```

```
# create a embedding matrix by using words only found in the dataset
words_in_data = len(tokenizer.word_index) + 1
embedding_matrix = np.zeros((words_in_data, 100))
for word, i in tokenizer.word_index.items():
    temp = word_embeddings.get(word)
    if temp is not None:
        embedding_matrix[i] = temp
```

4. Perform dimensionality reduction
    a. Use sklearn.PCA() to get vectors of shape (1, 2) so that the SOM can cluster onto a 2D axis.
    b. Perform PCA on both the embedding matrix and document vectors.

```
#PCA for dimensionality reduction of embedding matrix feature space
dim_reduc = PCA(2).fit(embedding_matrix)
data_dim_reduc = dim_reduc.transform(embedding_matrix)
```

```
print(embedding_matrix.shape)
```

```
(67156, 100)
```

```
print(data_dim_reduc[0])
print(data_dim_reduc.shape)
```

```
[0.0415178  0.32232709]
(67156, 2)
```

5. Build SOM:
    a. Using MiniSom Library set neighborhood radius and learning rate parameters of the SOM with shape (1,2)
    b. Model was based on the Kohonen algorithm that uses the Winner takes All approach by using 'best matching unit'. Train the SOM to cluster the embedding matrix and document vectors with reduced dimensionality.
    c. Plot the identified clusters using the MiniSom plotting script.

```
#create a self-organizing map that will cluster the document vectors feature space into 2 clusters
som_shape_2 = (1, 2)

#use the minisom library to set SOM parameters to have neighborhood radius as 1 and learning rate as 0.5
som_2 = MiniSom(1, 2, 2, sigma=1, learning_rate=0.5,
              neighborhood_function='gaussian', random_seed=10)

#train the SOM to cluster the document vectors feature space
som.train_random(data_dim_reduc_2, 500, verbose=True)
```

```
[ 500 / 500 ] 100% - 0:00:00 left
quantization error: 6266.030831565144
```

```
# calculate the winning nodes
winner_coordinates_2 = np.array([som_2.winner(x) for x in data_dim_reduc_2]).T

#distribute coordinates of winning nodes onto 2D axis
cluster_index_2 = np.ravel_multi_index(winner_coordinates_2, som_shape_2)
```

```
print(winner_coordinates_2.shape)
```

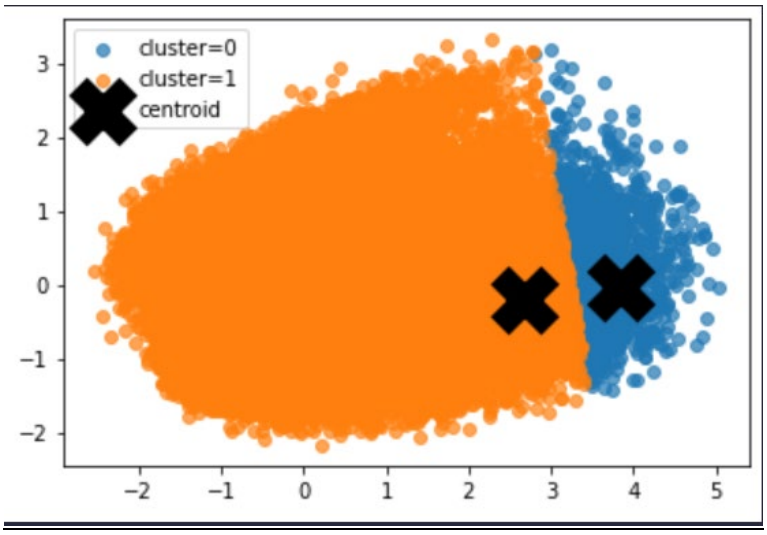
```
(2, 24987)
```

```
# use MiniSom documentation to plot the clusters
for c in np.unique(cluster_index):
    # plotting the clusters using the first 2 dimentions of the data
    plt.scatter(data_dim_reduc_2[cluster_index_2 == c, 0],
                data_dim_reduc_2[cluster_index_2 == c, 1], label='cluster='+str(c), alpha=.7)

for centroid in som_2.get_weights():
    plt.scatter(centroid[:, 0], centroid[:, 1], marker='x',
                s=80, linewidths=35, color='k', label='centroid')
plt.legend();
```

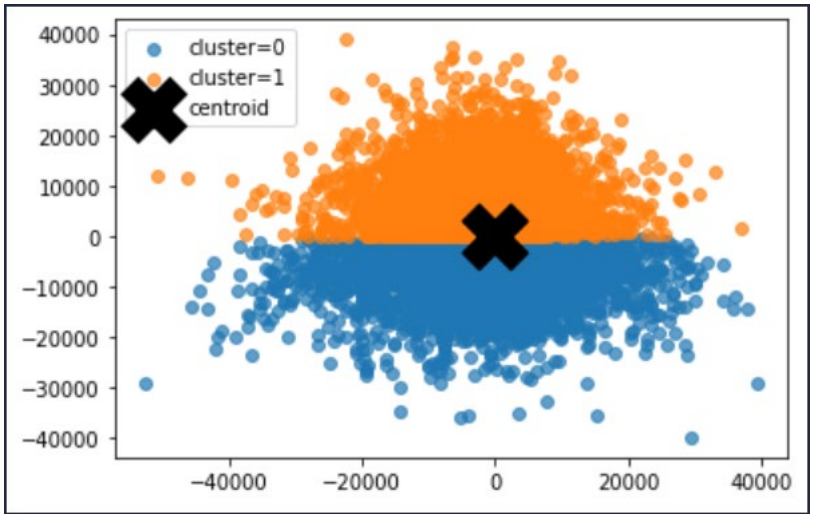
Results

Clustering of word embedding matrix



The SOM created 2 clusters from the word embedding matrix whose feature space uses the dense word embedded vectors. It can be observed that there are two centroids which represent the mean of the values of the points of data in the cluster. It can be interpreted that similar words have been clustered together for the word embeddings. There seem to be more of one group of words than the other.

Clustering of document vectors



The SOM created 2 clusters from the document vectors whose feature space uses the encoded and padded versions of the text documents. It can be observed that the clusters have almost been evenly split. This can be considered a favorable

result as the original dataset contains an almost equal amount of both positive and negative movie reviews.

Observations

1. The SOM did a good job in forming 2 clusters out of the movie review dataset. The document vectors were almost evenly split between the clusters, which is line with the original structure of the dataset.

Form the above observations, we can seemingly improve the net by testing different parameters in the SOM initialization. These include the neighborhood radius for weight updating and the learning rate. Finally, with different combinations of the above-mentioned parameters and changing the number of epochs we can observe different results.

Conclusion

Text analysis is a complex problem which aims to convert unstructured data into a structured format that can be used to extract meaningful information from it. Under the branch of Artificial Intelligence, Natural Language Processing has become an emerging field to tackle the problem of text analysis.

As part of this project two approaches for text classification were looked at, supervised learning using CNNs and unsupervised learning using SOM. Both these methods proved to have their own advantages. The CNN model performed well at feature extraction from the labelled dataset and was able to classify the testing data with up to 86% accuracy. On the other hand, the main advantage of the SOM was its ability to self-learn, without labelled data. However due to this, it becomes hard to provide strong initial weights. There will be a trial-and-error phase to come up with appropriate initial weights as this will greatly affect the clustering of the data. The data also needs to be organized in such a manner so that nearby vectors are similar to each other. To overcome this, pretrained word embeddings were used in both cases to increase efficiency.

Bibliography

1. Brownlee, Jason. “How to Use Word Embedding Layers for Deep Learning with Keras.” *Machine Learning Mastery*, 3 Oct. 2017, https://machinelearningmastery.com/use-word-embedding-layers-deep-learning-keras/.

2. Jacovi, Alon, et al. “Understanding Convolutional Neural Networks for Text Classification.” *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, Association for Computational Linguistics, 2018, pp. 56–65. *DOI.org (Crossref)*, doi:10.18653/v1/W18-5408.

3. Kalchbrenner, Nal, et al. “A Convolutional Neural Network for Modelling Sentences.” *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Association for Computational Linguistics, 2014, pp. 655–65. *DOI.org (Crossref)*, doi:10.3115/v1/P14-1062.

4. Kohonen, T., et al. “Self Organization of a Massive Document Collection.” *IEEE Transactions on Neural Networks*, vol. 11, no. 3, May 2000, pp. 574–85. *DOI.org (Crossref)*, doi:10.1109/72.846729.

5. Liu, Yuan-Chao, et al. “Application of Self-Organizing Maps in Text Clustering: A Review.” *Applications of Self-Organizing Maps*, Nov. 2012. *www.intechopen.com*, doi:10.5772/50618.

6. Maas, Andrew L., et al. “Learning Word Vectors for Sentiment Analysis.” *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, Association for Computational Linguistics, 2011, pp. 142–150, http://www.aclweb.org/anthology/P11-1015.

7. Team, Keras. *Keras Documentation: Keras Layers API*. https://keras.io/api/layers/. Accessed 17 Dec. 2020.

8. *Keras Documentation: Using Pre-Trained Word Embeddings*. https://keras.io/examples/nlp/pretrained_word_embeddings/. Accessed 13 Dec. 2020.

9. “What Is Principal Component Analysis (PCA) and How It Is Used?” *Sartorius,* https://www.sartorius.com/en/knowledge/science-snippets/what-is-principal-component-analysis-pca-and-how-it-is-used-507186. Accessed 18 Dec. 2020.