CS-419(M)
“I hated it. 8/10”

Group 7

Hrithik Mhatre | Vineet Jaiswal | Vaishnav Vernekar

May 7, 2024

Project Introduction : ‘<u>I hated it. 8/10</u>’

- Generating Customer Ratings from Text Reviews for a Sample Airline Company, to reconstruct the customer ratings(1-5) only from their text reviews
- Leveraging natural language processing (NLP) and machine learning techniques to infer the missing ratings from the available text data
- This project involves preprocessing and analyzing the text reviews, using various machine learning and deep learning models, and evaluating their performance in predicting customer ratings accurately
- This would not only give airlines a solution to mitigates the impact of oversight but also helps to understand and improve to cater to customer needs efficiently

Additional Aim

- Compare different word embeddings methods and find out which one gives the best accuracy
- The word embedding methods used were
    - One Hot Encodings + Random Forest
    - Bag Of Words + Random Forest
    - C-Bow + XG boost
    - Using Pre Trained word embeddings (Word2Vec) + XG Boost / SVM / NN

Data Set

We used ‘**Singapore Airlines Reviews’** dataset which has around 10,000 anonymized customer reviews

The Singapore Airlines review dataset from Kaggle contains customer reviews and ratings for Singapore Airlines. It includes text reviews along with corresponding ratings (5 classes) provided by customers

It provides valuable insights into customer perceptions and satisfaction levels regarding Singapore Airlines' services in the past years

Link for the dataset- Singapore Airlines Reviews (kaggle.com)

MACHINE LEARNING Techniques In NLP

Natural Language Processing contains algorithms and models to deal with textual data, to understand it and to understand the structure, meaning, and context of human language, including grammar, semantics, and syntax.

NLP techniques include tokenization, parsing, stemming, lemmatization, part-of-speech tagging, named entity recognition, and semantic analysis.

- Tokenization: Tokenization is the process of breaking down a text into smaller units called tokens.
- Parsing: Parsing involves analyzing the grammatical structure of a sentence to understand its components and their relationships.
- Stemming: Stemming is the process of reducing words to their root or base form.
- Lemmatization: Lemmatization is similar to stemming but more precise.
- Part-of-Speech Tagging: Part-of-speech tagging assigns grammatical categories (such as noun, verb, adjective, etc.) to words in a sentence.

Extracting relevant features from the text data to represent it in a format suitable for machine learning algorithms.

Data Cleaning and Preprocessing

- Data Cleaning: Data cleaning involves removing noise and irrelevant information from the text data. This may include **removing special characters, punctuation, numbers, and other non-alphabetic characters** that do not contribute to the analysis
- Text Normalization: Text normalization involves standardizing text data to a common format. This includes converting text to **lowercase, Lemmatization**
- Stopword Removal: Stopwords are common words (e.g., "the," "is," "and") that occur frequently in a language but do not carry significant meaning. Removing stop words helps reduce noise and focus on the important words in the text
- Tokenization : **Tokenization** was performed to break text into individual words or smaller linguistic units, such as tokens or sentences, to facilitate analysis and processing.

Training and Validation Datasets

- The available dataset is divided into 2 datasets : Training and Validation Datasets
- Use train_test_split() function with parameter test_size=0.2 to split the dataset
- The dataset is divided into 80% training dataset and 20% validation data

One Hot Encodings

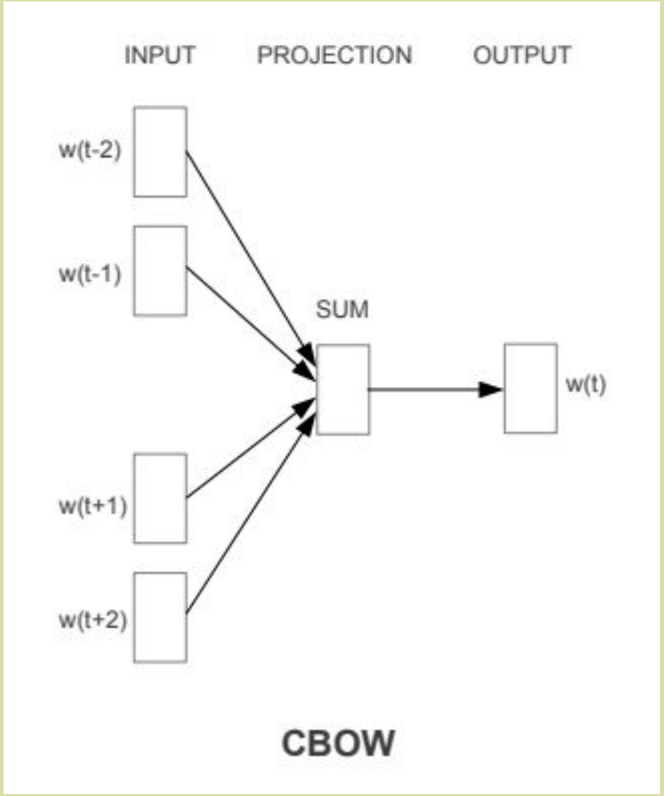
- One Hot encoding in NLP represents words or tokens in a vocabulary as binary vectors with a single "1" indicating the presence of a word and "0" elsewhere
- Advantage
  - Simple Representation
- Disadvantage
  - High Dimensionality
  - Sparse Representation
  - Lack of contextual Information
- Were not able to train the model due to very large size of vocabulary

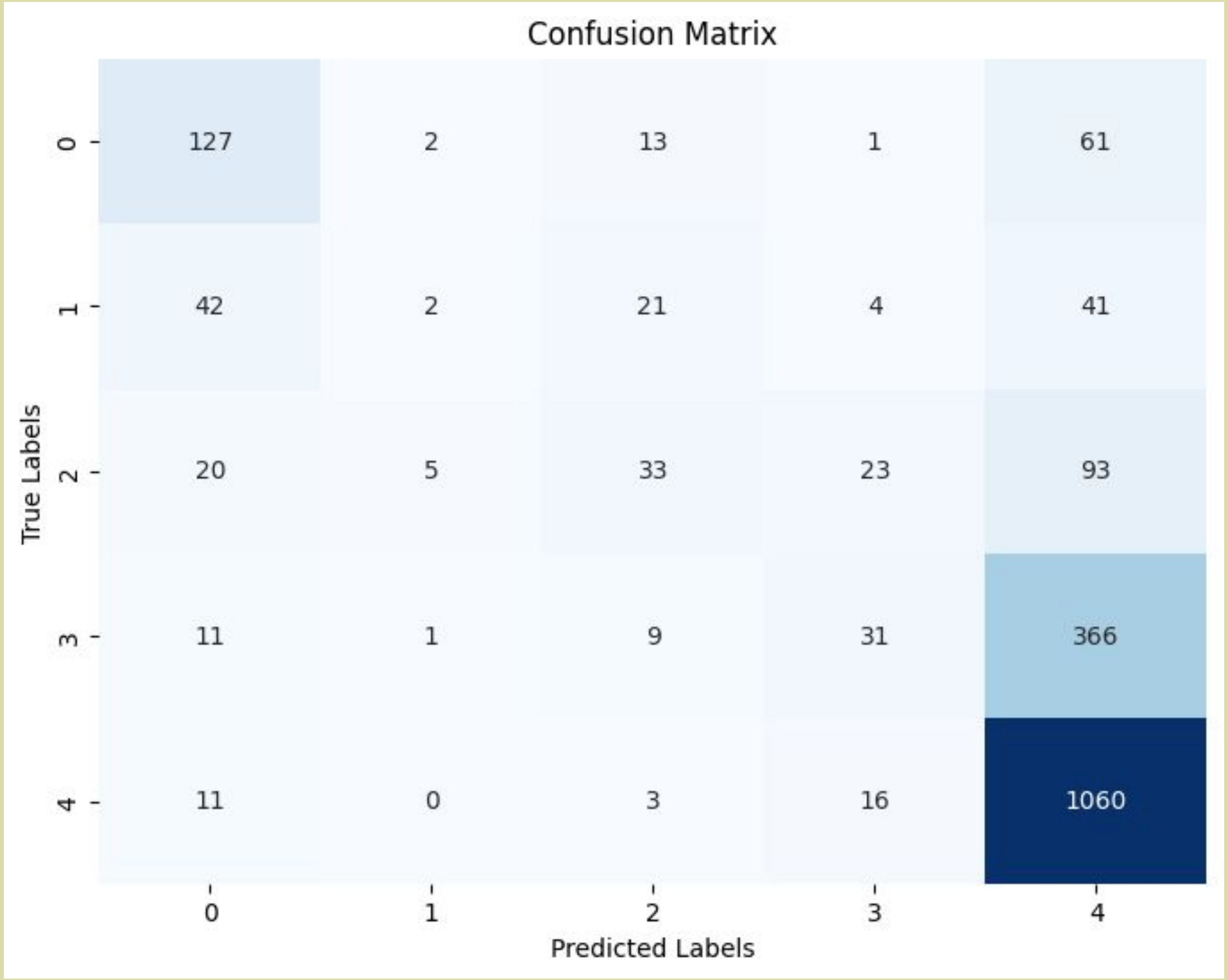
Bag OF Words

- In NLP, the Bag of Words model represents text by counting the frequency of words in a document, disregarding grammar and word order
- Max Features : 500
- Random Forest : n_estimators = 100, criterion = ‘entropy’
- Advantage over OHE : Frequency of words is taken into account
- Same disadvantages of OHE are observed
- Accuracy = 62.77 %

CBOW - Training our own word embeddings

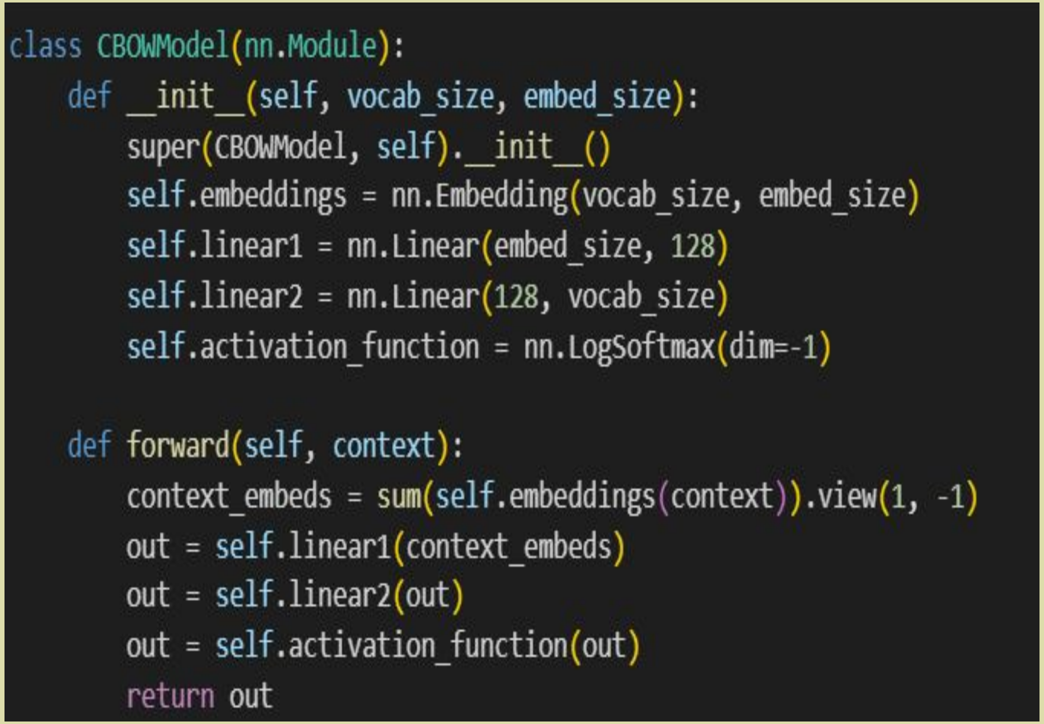
- CBOW (Continuous Bag of Words) is a word embedding method in NLP where the target word is predicted based on its context words in a given window size
- Context size = 2
- Each word was assigned a token which was used while training





True \ Predicted	0	1	2	3	4
0	127	2	13	1	61
1	42	2	21	4	41
2	20	5	33	23	93
3	11	1	9	31	366
4	11	0	3	16	1060

Model Architecture



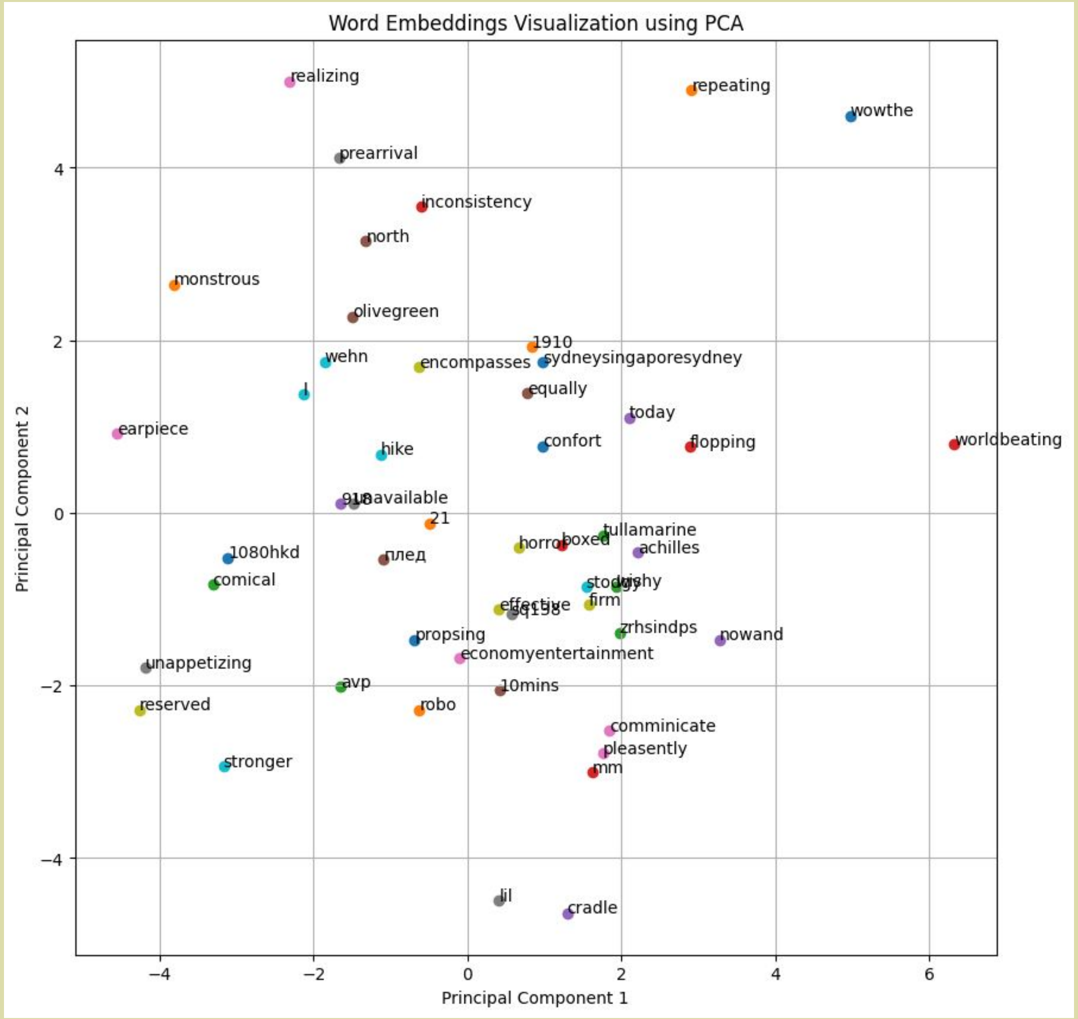
```
class CBOWModel(nn.Module):
    def __init__(self, vocab_size, embed_size):
        super(CBOWModel, self).__init__()
        self.embeddings = nn.Embedding(vocab_size, embed_size)
        self.linear1 = nn.Linear(embed_size, 128)
        self.linear2 = nn.Linear(128, vocab_size)
        self.activation_function = nn.LogSoftmax(dim=-1)

    def forward(self, context):
        context_embeds = sum(self.embeddings(context)).view(1, -1)
        out = self.linear1(context_embeds)
        out = self.linear2(out)
        out = self.activation_function(out)
        return out
```

Hyper-Parameters

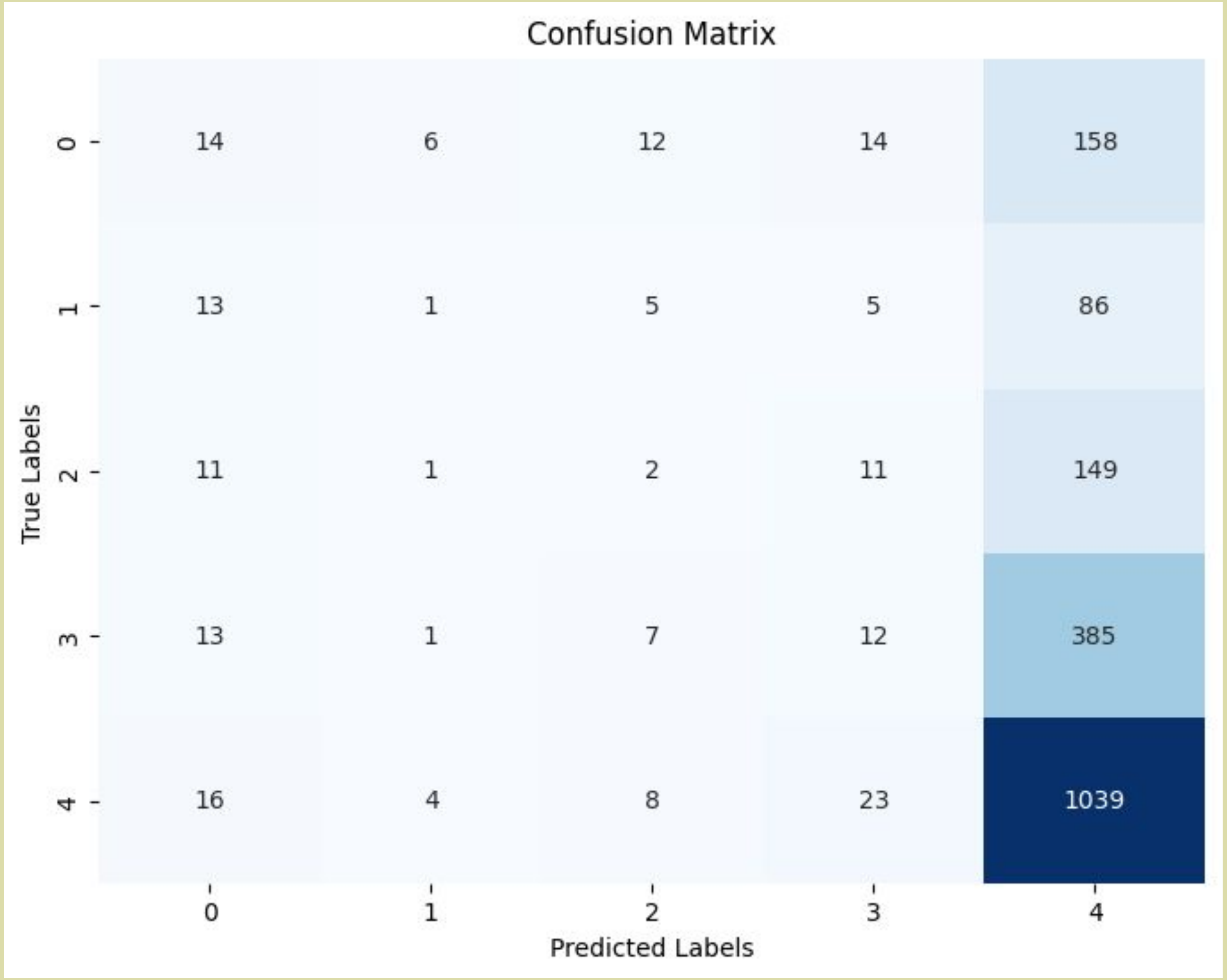
- vocab_size = len(vocabulary)
  embed_size = 100
- learning_rate = 0.001
- epochs = 25
- batch_size = 256

Loss = nn.CrossEntropy()

Optimiser = ADAM



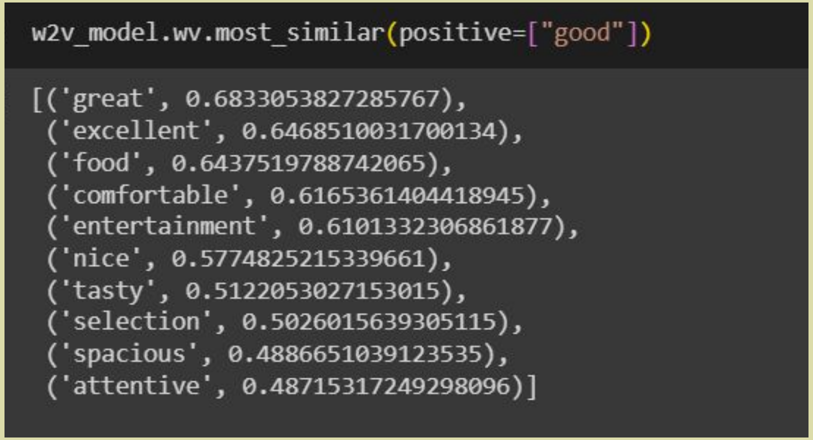
- Trained for a very less time due to computational restrictions so not able to obtain good results
- Random Forest : n_estimators = 100, criterion = ‘entropy’
- Accuracy = 53.50 %


Confusion Matrix
True Labels
Predicted Labels
0
1
2
3
4
14 6 12 14 158
13 1 5 5 86
11 1 2 11 149
13 1 7 12 385
16 4 8 23 1039

Pre Trained Word Embeddings - Word2Vec

- Word2Vec is a popular word embedding technique that maps words to dense vectors in a continuous vector space, capturing semantic similarities. It learns by predicting the surrounding words within a context window for each target word in a large corpus
- gensim-word2vec was used
- Hyper Parameters : vector-size = 300, window = 5

Results of word2vec

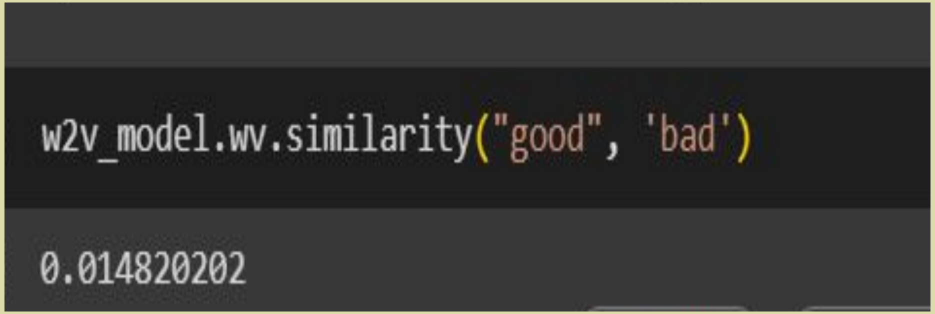


```
w2v_model.wv.most_similar(positive=["good"])
```

```
[('great', 0.6833053827285767),
 ('excellent', 0.6468510031700134),
 ('food', 0.6437519788742065),
 ('comfortable', 0.6165361404418945),
 ('entertainment', 0.6101332306861877),
 ('nice', 0.5774825215339661),
 ('tasty', 0.5122053027153015),
 ('selection', 0.5026015639305115),
 ('spacious', 0.4886651039123535),
 ('attentive', 0.48715317249298096)]
```

Most Similar Words

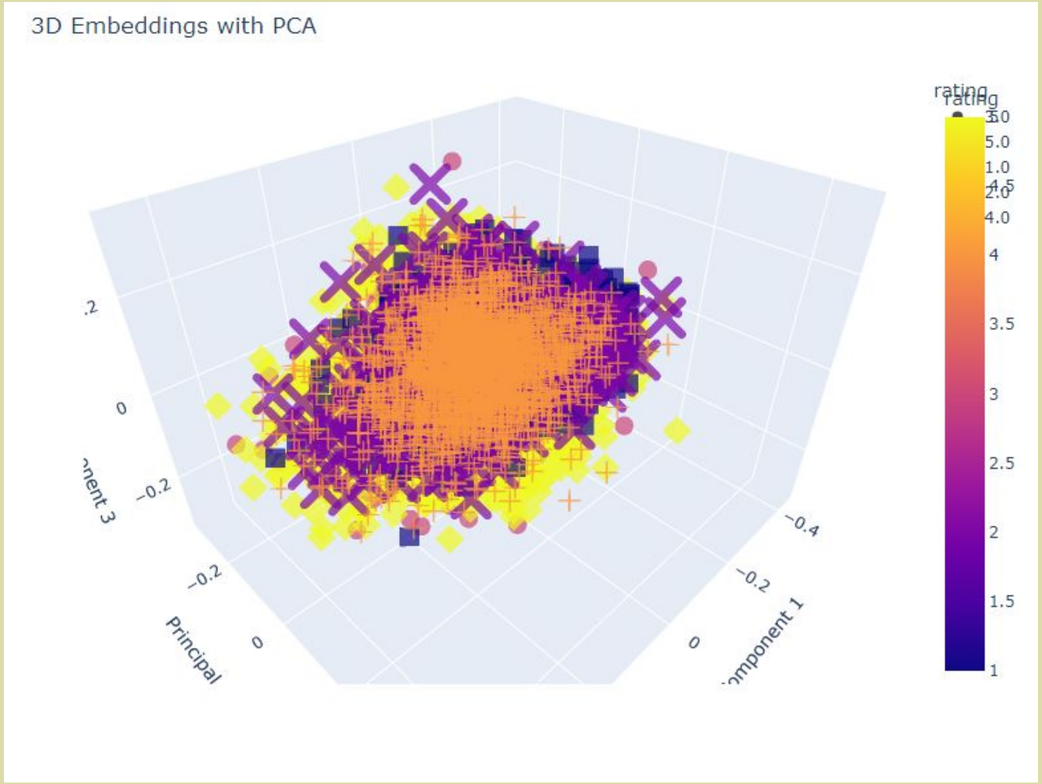
Cosine Similarity



```
w2v_model.wv.similarity("good", 'bad')
```

```
0.014820202
```

Using PCA to view all the sentences



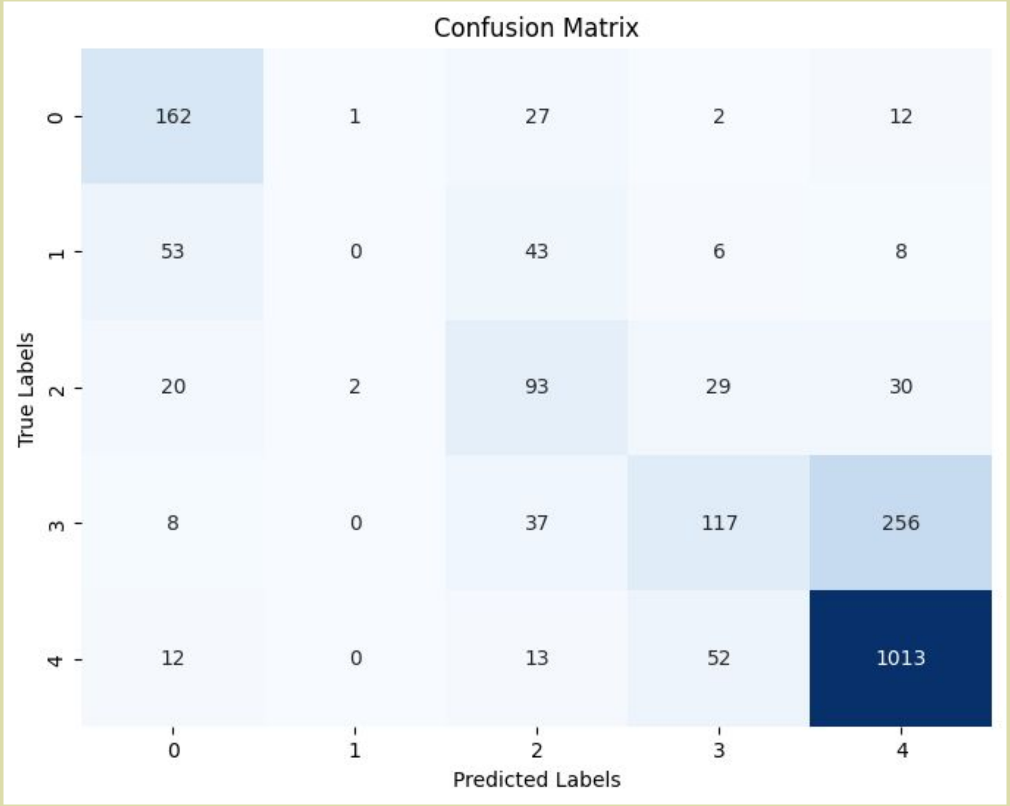
Trying to view if all the 5 classes were separated or Not

SUPPORT VECTOR MACHINE

- The code specifically utilizes the Support Vector Classifier (SVC) with a radial basis function (RBF) kernel.
- It processes embeddings extracted from the DataFrame df_w2v, representing word embeddings generated.
- The data is split into training and testing sets with a ratio of 80:20 using train_test_split from scikit-learn.
- The SVM classifier is instantiated and trained on the training data (X_train, y_train) using the fit method.
- Predictions are made on the testing data (X_test) using the trained classifier, resulting in y_pred. The accuracy of the model is calculated by comparing the predicted labels (y_pred) with the true labels (y_test) using NumPy's sum function.
- A confusion matrix is generated using scikit-learn's confusion_matrix function, providing insights into the classifier's performance across different classes.
- The confusion matrix is visualized as a heatmap plot using matplotlib and seaborn libraries, facilitating a clear understanding of classification results.

Using SVM on Word2Vec Embeddings

- Kernel = rbf
- Accuracy = 69.78 %



True \ Predicted	0	1	2	3	4
0	162	1	27	2	12
1	53	0	43	6	8
2	20	2	93	29	30
3	8	0	37	117	256
4	12	0	13	52	1013

XGBOOST

Initialized a Random Forest Classifier model ('w2v') with the following parameters:

- Number of estimators: 100 decision trees are used in the ensemble.
- Criterion for splitting nodes: Nodes are split based on the 'entropy' criterion, which measures the information gain achieved by each split.
- Random state: A random state is specified to ensure reproducibility of the results.

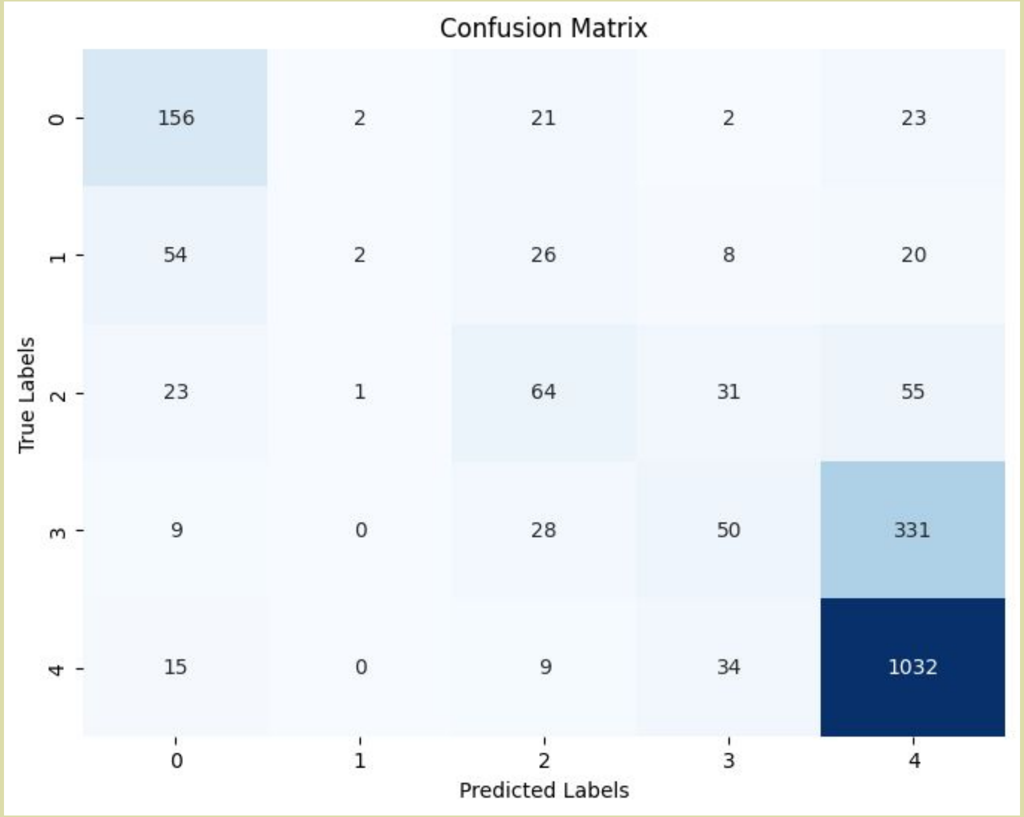
The model is trained using the training data (X_train and y_train) with the 'fit' method. Next, the model predicts the ratings for the test data using the 'predict' method, and the predicted ratings are stored in 'y_pred'.

The accuracy is calculated as the ratio of the count of correct predictions to the total number of predictions. The accuracy value is printed to the console.

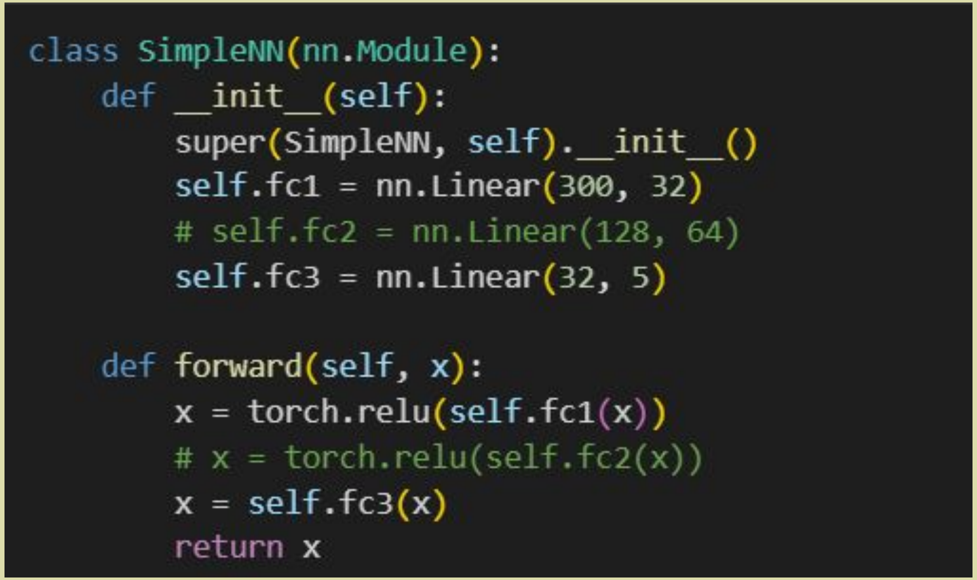
Finally, a confusion matrix and heatmap visualization of the confusion matrix is plotted using Matplotlib and Seaborn. The confusion matrix helps visualize the model's performance by showing the counts of true positive, false positive, true negative, and false negative predictions.

Using XGBoost on Word2Vec Embeddings

- Accuracy = 65.53 %



Using NN Model on Word2Vec Embeddings



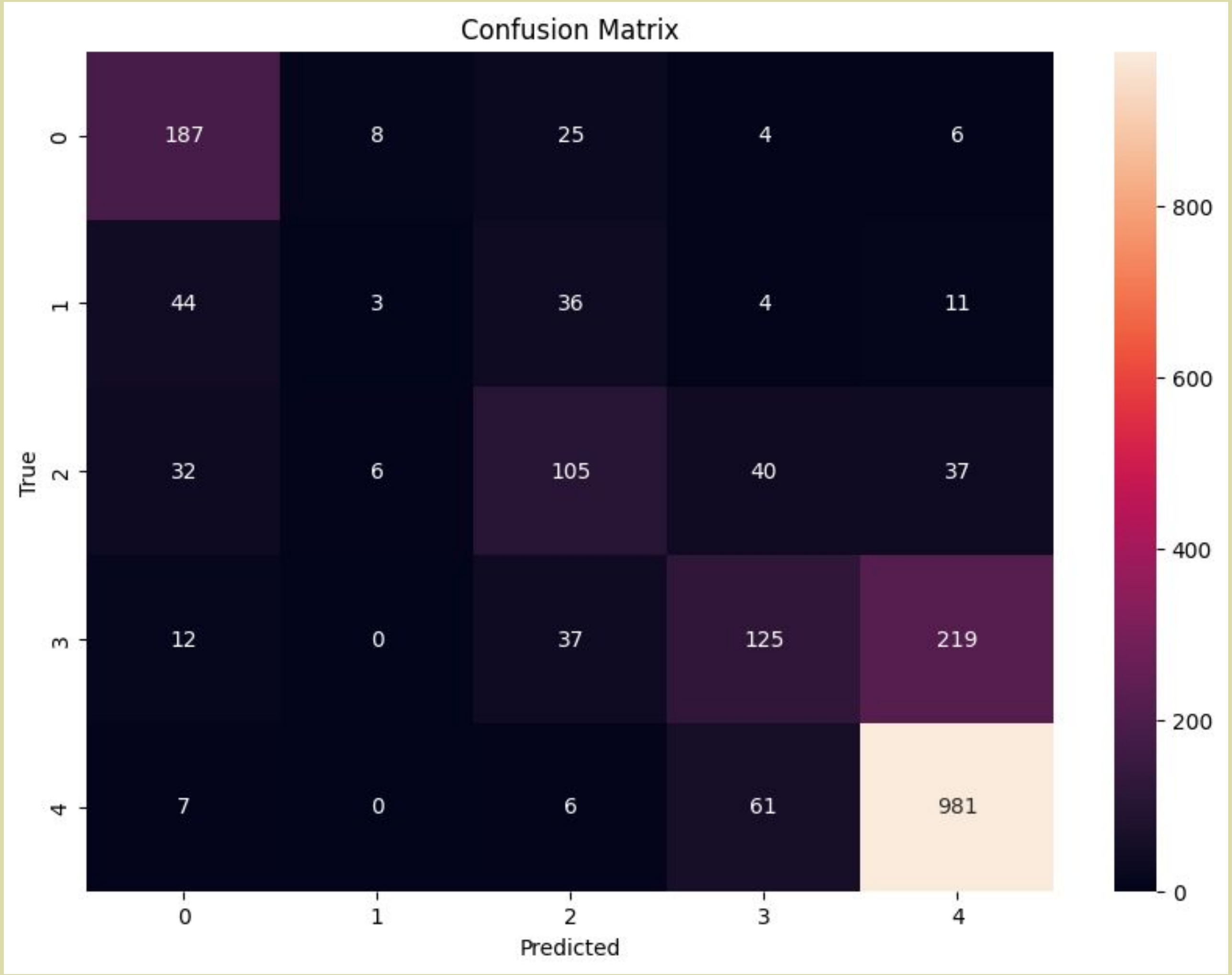
```
class SimpleNN(nn.Module):
    def __init__(self):
        super(SimpleNN, self).__init__()
        self.fc1 = nn.Linear(300, 32)
        # self.fc2 = nn.Linear(128, 64)
        self.fc3 = nn.Linear(32, 5)

    def forward(self, x):
        x = torch.relu(self.fc1(x))
        # x = torch.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

- Loss Function = nn.CrossEntropyLoss()
- Optimiser = ADAM
- Epoch = 50

Accuracy = 70.19 %

Major Problem :

Model is getting confused between texts of ratings **1 and 2** and **4 and 5**



	0	1	2	3	4
0	187	8	25	4	6
1	44	3	36	4	11
2	32	6	105	40	37
3	12	0	37	125	219
4	7	0	6	61	981