IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 7.39 ⁜ Vol. 11, Issue 4, April 2022

DOI: 10.17148/IJARCCE.2022.11419

Sentiment Analysis of Movie Reviews Using Machine Learning Techniques

Nikita Verma[1], Ritik Kaushik[2], Sumit Tyagi[3], Raman Dahiya[4]

Indraprastha Engineering College, Department of Computer Science and Engineering

AKTU university, India, Uttar Pradesh

Abstract: Sentimental analysis is the analysis of opinions and emotions from any type of message or text. Sentimental analysis of the data is very helpful to convey the emotion of the majority or particular individual. It is also called opinion mining. This technique is performed to find the sentiment of the individual for a given source of content. Online platforms like social media contain a high volume of data in the form of reviews, tweets, comments, blogs, etc. This paper analyzes the movie reviews using various methods such as KNN, Naive Bayes, LSTM, and Random Forest Classifier.

Keywords: Dataset, Sentiment Analysis, Reviews, K-Nearest Neighbor, Naive Bayes.

I. INTRODUCTION

The Internet has evolved on such a large scale that everything can be controlled through the web because people want to share every part of their life on social media, which also includes their views on every single thing happening in the real world. Nowadays, the success or failure of a product is fully dependent on the reviews and satisfaction of consumers whether it be a new or already existing product. These opinions people share on social networking sites are generally known as Short Texts because of their length [1]. These short texts have taken over the traditional approaches because of the amount of the public getting influenced by them. This is why it is important to classify them based on how they're used (slang, sarcasm, sentiment, etc.).

The movie review is a popular way of doing an analysis of a movie and collectively determining whether the movie is worth recommending. The idiosyncrasy of a movie review is that it does not simply evaluate the movie but also gives the viewer the required aspect, which is the key foundation of the review. Reviews are posted by evaluating factors including plot, acting, dialogues, cinematography, and music. These reviews are shared and stored on various platforms and one of them is IMDB, which stores the data of every movie with summary, rating, and collective reviews from viewers. The task of Sentiment analysis mainly includes Preprocessing [2], Classification [3], and at last analysis of the results.

Sentiment analysis is the automated process of analyzing sentences to determine the sentiment formulated which can be either positive, negative, or neutral. These sentiments are evaluated based on emotions like happiness, sadness, agony, hated resentment, and affection.

Sentiment analysis is so efficient, and because of its accuracy it has numerous popular applications like:

- **Social media monitoring.** Social media posts often demonstrate some of the most honest views about products. Multiple tweets are posted every second and processing all of them manually will be troublesome. But, with the help of ML software, we can filter that data in a few minutes.
- **Customer feedback.** Sentiment analysis can also be used to gain intuitions from the collection of feedback available online.

Sentiment analysis will calculate whether the comment should be considered positive or negative.

- **Market and competitor research.** Sentiment analysis can also be used to find out who's trending among your competitors and how it can affect the market. It will analyze the competitor's data to find out what could work out for the public by understanding the strengths and weaknesses of both sides.

II. MACHINE LEARNING METHODS

A. DECISION TREE

It is the type of supervised machine learning algorithm and it is used for both classification and regression problems. It is a graph that uses tree-based methods to illustrate a tree's every possible outcome. It is a decision support tool and it uses a tree-like structure and its possible consequences.

It starts with a single node and branches off into other possibilities.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 7.39 ⁜ Vol. 11, Issue 4, April 2022

DOI: 10.17148/IJARCCE.2022.11419

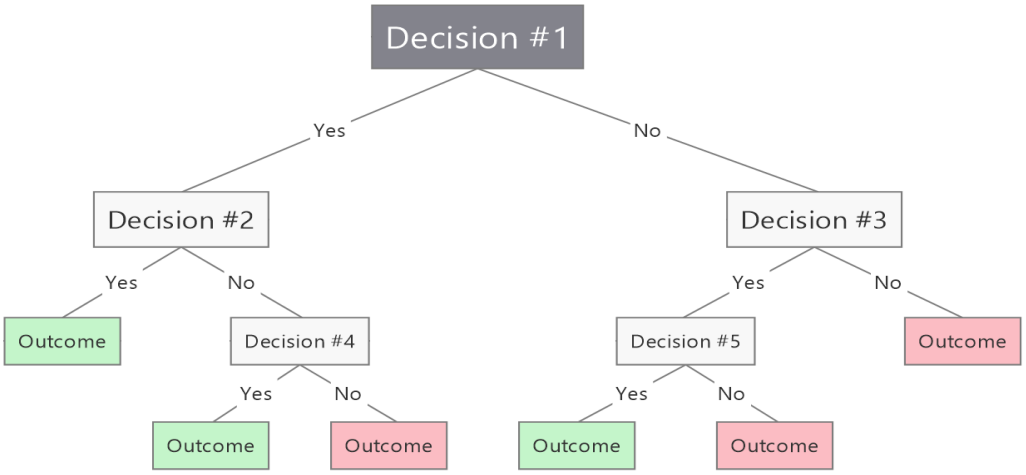
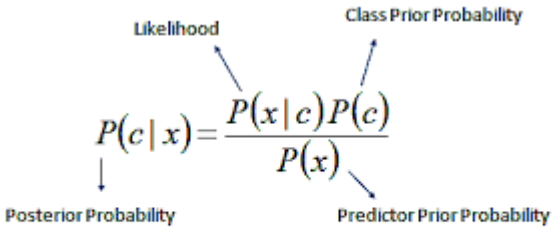


Fig 3.1.1 A figure that represents different states of decision and what outcome should be generated as a result using conditions (yes/no).

B. NAIVE BAYES

It is a supervised machine learning algorithm, and it works based on Bayes' theorem. Naive Bayes uses conditional probability theorems to classify the data. Bayes classifiers will assume strong or naïve independence between attributes of data points. These methods are broadly used in text categorization-based problems as it is easy to implement. it is also known as independence Bayes.



$$P(c \mid X) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$

Fig. 3.1.2 A figure that represents how to calculate posterior probability using different terms p(x|c) that means likelihood, p(c) means class prior probability and p(x) means predictor prior probability.

C. K-NEAREST NEIGHBOR

It is the simplest classification algorithm that attempts to determine how near the group of data points are.it used both classification and regression base problems.

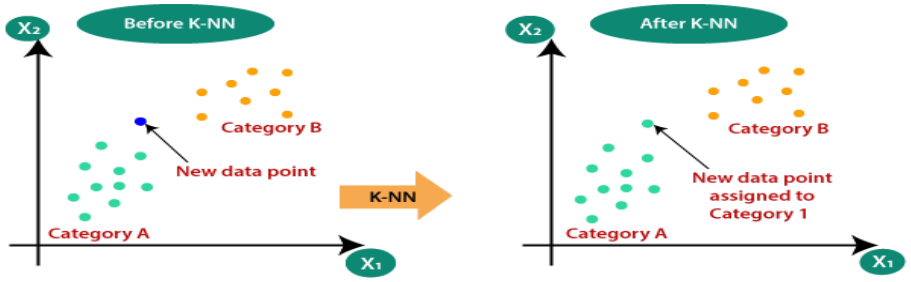


Fig. 3.1.3 In the above representation there are 2 categories (category A and category B), where problem is that we have a new data point x1 that makes confusion that in which category a new data point lies? To solve this problem, we required a K-NN algorithm because K-NN easily identifies the category of a particular dataset.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 7.39 ⁜ Vol. 11, Issue 4, April 2022

DOI: 10.17148/IJARCCE.2022.11419

D. LSTM (LONG SHORT TERM MEMORY)

It is the revised version of a recurrent neural network. It is used for classification and attempts to predict the sentiment of the movie review.

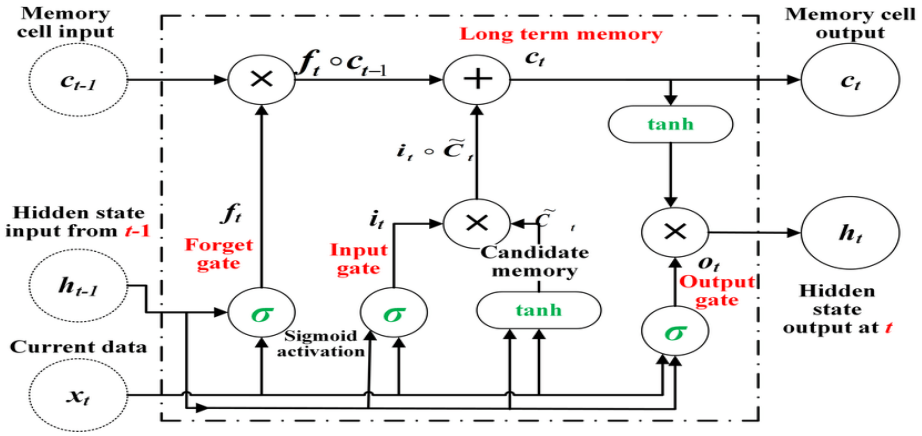


Fig. 3.1.4 In the above figure LSTM contains feedback connections which make them different to more traditional feedforward neural networks that enable LSTMs to process entire sequences of data are being biased on each point in the list independently and LSTMs specifically focus on processing continuous data such as text, speech, and general time-series.

III. LITERATURE SURVEY

In paper [4] Joscha et. al, performed a comparison between various machine learning methods like a bag of words, n-grams for using semantic information to enhance the performance of sentiment analysis. In earlier approaches, there was no consideration of the semantic associations between sentences or documents. [5] neither paper 2, compared the different methodological variants nor provided the most favorable method to merge the disclosure units. They aimed to improve the sentiment analysis by using Rhetoric Structure Theory as it gives hierarchical representations at the document level. An integration grid search and weighting were proposed by them to find out the average scores of sentiments from the RST tree. To reduce the complexity of the RST tree, binary data was encoded into the random forest using feature engineering. Also, they gave the conclusion that ml raised the balanced accuracy and gave a high F1 score of 71.90%. In paper [6] Dhiraj Murthy provided what role is played by tweets in the political elections. Analysis of the sentences was done. Sentences that contain quantitative terms to classify. They were classified as opinionated or non-opinionated and identifications of the polarity were expressed using fuzzy set theory. The fuzzy base knowledge base was developed by interviewing various health center doctors. There was various research done in the field but they haven't considered the quantitative data contained in the reviews while analyzing the sentiment polarity of the review [6]. [7] In sentiment analysis, the challenging part is the opinion of words in a sentence there are situations when a word(sentence) is considered as positive and sometimes they are considered as negative this creates confusion whereas the traditional text processing considers that if there is a 2-bit change in the word then no change is considered. In paper [8] Ahmad Kamal, an opinion mining framework was designed that facilitates objectivity or subjectivity analysis, features extraction and reviews summarization, etc.

In paper [9], Researchers examined the prevalence and geographic variations for opinion polarities about e-cigarettes on Twitter. They collected data from Twitter by pre-defined seven keywords. They classified the tweets into four categories: Irrelevant to e-cigarettes, Commercial tweets, organic tweets with attitudes (supporting or against or neutral) the use of e-cigarettes, and the geographic locations information city and state. In paper [10], Akshay Amolik et. al. created the dataset using Twitter posts of movie reviews and related tweets about those movies. On tweets, Sentence level sentiment analysis was performed. In paper [11] Humera Shaniya et al. classified movie reviews for sentiment analysis using WEKA Tool. They enhanced the earlier work done in sentiment categorization analysis and categorized them as positive and negative.

PROPOSED METHOD

Sentiments are the opinion or view that is held and expressed. Sentiments (emotions) are of different types like positive, negative. There are many reasons why these sentiments can be positive or negative based on the attitude(likes or dislikes) of the person giving the movie review. We have employed the LSTM algorithm in the suggested system to identify the sentiment of the movie review.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

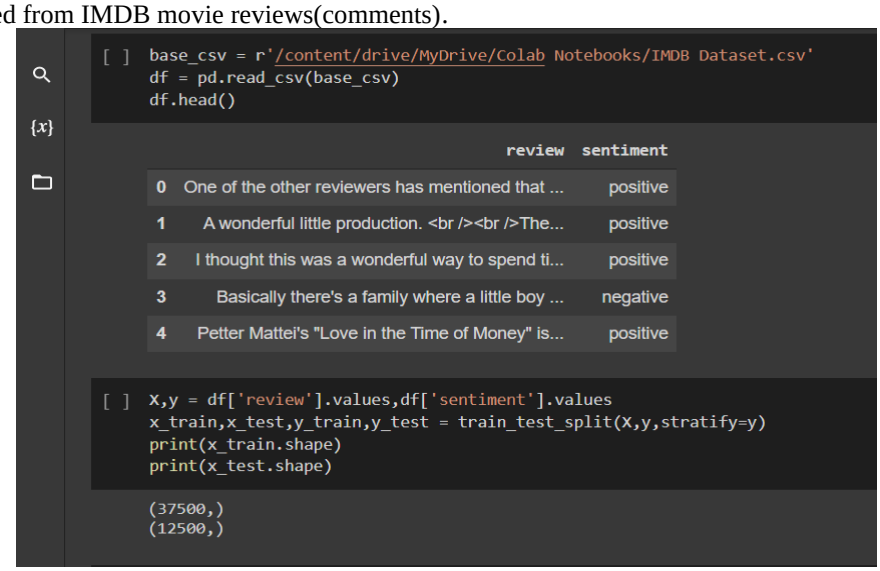
International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 7.39 ⁜ Vol. 11, Issue 4, April 2022

DOI: 10.17148/IJARCCE.2022.11419

A. Dataset

This collection contains 50k movie reviews and the associated sentiment which is either positive or negative. The dataset have been collected from IMDB movie reviews(comments).



```
[ ] base_csv = r'/content/drive/MyDrive/Colab Notebooks/IMDB Dataset.csv'
    df = pd.read_csv(base_csv)
    df.head()
```

	review	sentiment
0	One of the other reviewers has mentioned that ...	positive
1	A wonderful little production. The...	positive
2	I thought this was a wonderful way to spend ti...	positive
3	Basically there's a family where a little boy ...	negative
4	Petter Mattei's "Love in the Time of Money" is...	positive

```
[ ] X,y = df['review'].values,df['sentiment'].values
    x_train,x_test,y_train,y_test = train_test_split(X,y,stratify=y)
    print(x_train.shape)
    print(x_test.shape)

    (37500,)
    (12500,)
```

Fig:2 DATASET BREAKUP (TABLE I) DATASET IMAGES

B. Preprocessing

Data Pre-Processing plays a very important role in machine learning. It transforms raw data into a useful data format. Commonly it is the primary step to clean the data. Data Pre-Processing transforms the data into a format which is more easy and error free processing for the classification. The dataset is first processed to remove the null attributes and the records that contain the NA attributes.

After that we will process the sentences. we will be performing tokenization i.e sentences will be broken into smaller parts called as tokens and we will be removing the stopwords like is, are, the etc.

Next step will be performing padding operations. It will help to make the length of the sentences equal.

All the experiments are run over a set of train and test data.

```
def preprocess_string(s):
    s = re.sub(r"[^\w\s]", '', s)
    s = re.sub(r"\s+", '', s)
    s = re.sub(r"\d", '', s)

    return s
def tockenize(x_train,y_train,x_val,y_val):
    word_list = []

    stop_words = set(stopwords.words('english'))
    for sent in x_train:
        for word in sent.lower().split():
            word = preprocess_string(word)
            if word not in stop_words and word != '':
                word_list.append(word)

    corpus = Counter(word_list)

    corpus_ = sorted(corpus,key=corpus.get,reverse=True)[:1000]

    onehot_dict = {w:i+1 for i,w in enumerate(corpus_)}


    final_list_train,final_list_test = [],[]
    for sent in x_train:
            final_list_train.append([onehot_dict[preprocess_string(word)] for word in sent.lower().split()
                                     if preprocess_string(word) in onehot_dict.keys()])
    for sent in x_val:
            final_list_test.append([onehot_dict[preprocess_string(word)] for word in sent.lower().split()
                                    if preprocess_string(word) in onehot_dict.keys()])

    encoded_train = [1 if label =='positive' else 0 for label in y_train]
    encoded_test = [1 if label =='positive' else 0 for label in y_val]
    return np.array(final_list_train), np.array(encoded_train),np.array(final_list_test), np.array(encoded_test),onehot_dict

[ ] x_train,y_train,x_test,y_test,vocab = tockenize(x_train,y_train,x_test,y_test)
```

we split the dataset into train and test to get the information how our model performs on unseen data. The ratio of split is 80-20. 80% of the total is train and 20% contains testdata.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 7.39 ⁜ Vol. 11, Issue 4, April 2022

DOI: 10.17148/IJARCCE.2022.11419

C. Feature Selection

It is one the most important steps, here we will select the features that will increase the efficiency of the model.

D. Model Selection And Training

We have employed LSTM as the machine learning algorithm that will be used and we will perform training of the model on the dataset. LSTM basically refers to long short term memory. It helps to store the important words in its memory that will help to predict the sentiment.

```python
class SentimentRNN(nn.Module):
    def __init__(self,no_layers,vocab_size,hidden_dim,embedding_dim,drop_prob=0.5):
        super(SentimentRNN,self).__init__()

        self.output_dim = output_dim
        self.hidden_dim = hidden_dim

        self.no_layers = no_layers
        self.vocab_size = vocab_size

        # embedding and LSTM layers
        self.embedding = nn.Embedding(vocab_size, embedding_dim)

        #lstm
        self.lstm = nn.LSTM(input_size=embedding_dim,hidden_size=self.hidden_dim,
                            num_layers=no_layers, batch_first=True)


        # dropout layer
        self.dropout = nn.Dropout(0.3)

        # linear and sigmoid layer
        self.fc = nn.Linear(self.hidden_dim, output_dim)
        self.sig = nn.Sigmoid()

    def forward(self,x,hidden):
        batch_size = x.size(0)
        # embeddings and lstm_out
        embeds = self.embedding(x)  # shape: B x S x Feature   since batch = True
        #print(embeds.shape)  #[50, 500, 1000]
        lstm_out, hidden = self.lstm(embeds, hidden)

        lstm_out = lstm_out.contiguous().view(-1, self.hidden_dim)

        # dropout and fully connected layer
        out = self.dropout(lstm_out)
        out = self.fc(out)

        # sigmoid function
        sig_out = self.sig(out)

        # reshape to be batch_size first
        sig_out = sig_out.view(batch_size, -1)

        sig_out = sig_out[:, -1] # get last batch of labels

        # return last sigmoid output and hidden state
        return sig_out, hidden
```

Shape:

- Input: $(N, C_{in}, H_{in}, W_{in})$
- Output: $(N, C_{out}, H_{out}, W_{out})$ where

$$H_{out} = \left\lfloor \frac{H_{in} + 2 \times \text{padding}[0] - \text{dilation}[0] \times (\text{kernel\_size}[0] - 1) - 1}{\text{stride}[0]} + 1 \right\rfloor$$

$$W_{out} = \left\lfloor \frac{W_{in} + 2 \times \text{padding}[1] - \text{dilation}[1] \times (\text{kernel\_size}[1] - 1) - 1}{\text{stride}[1]} + 1 \right\rfloor$$

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 7.39 ⁜ Vol. 11, Issue 4, April 2022

DOI: 10.17148/IJARCCE.2022.11419

```
[ ]  clip = 5
     epochs = 7
     valid_loss_min = np.Inf

     epoch_tr_loss,epoch_vl_loss = [],[]
     epoch_tr_acc,epoch_vl_acc = [],[]

     for epoch in range(epochs):
         train_losses = []
         train_acc = 0.0
         model.train()
         # initialize hidden state
         h = model.init_hidden(batch_size)
         for inputs, labels in train_loader:

             inputs, labels = inputs.to(device), labels.to(device)
             # Creating new variables for the hidden state, otherwise
             # we'd backprop through the entire training history
             h = tuple([each.data for each in h])

             model.zero_grad()
             output,h = model(inputs,h)

             # calculate the loss and perform backprop
             loss = criterion(output.squeeze(), labels.float())
             loss.backward()
             train_losses.append(loss.item())
             # calculating accuracy
             accuracy = acc(output,labels)
             train_acc += accuracy
             #`clip_grad_norm` helps prevent the exploding gradient problem in RNNs / LSTMs.
             nn.utils.clip_grad_norm_(model.parameters(), clip)
             optimizer.step()



         val_h = model.init_hidden(batch_size)
         val_losses = []
         val_acc = 0.0
         model.eval()
         for inputs, labels in valid_loader:
                 val_h = tuple([each.data for each in val_h])

                 inputs, labels = inputs.to(device), labels.to(device)

                 output, val_h = model(inputs, val_h)
                 val_loss = criterion(output.squeeze(), labels.float())

                 val_losses.append(val_loss.item())
```

E. Prediction

Once the model is trained we will be performing the prediction on the unseen data and checking whether the predicted sentiment is correct or not.

```
[ ] def predict_text(text):
        word_seq = np.array([vocab[preprocess_string(word)] for word in text.split()
                     if preprocess_string(word) in vocab.keys()])
        word_seq = np.expand_dims(word_seq,axis=0)
        pad =  torch.from_numpy(padding_(word_seq,500))
        inputs = pad.to(device)
        batch_size = 1
        h = model_pred.init_hidden(batch_size)
        h = tuple([each.data for each in h])
        output, h = model_pred(inputs, h)
        return(output.item())
```

```
index = 30
print(df['review'][index])
print('='*70)
print(f'Actual sentiment is  : {df["sentiment"][index]}')
print('='*70)
pro = predict_text(df['review'][index])
status = "positive" if pro > 0.5 else "negative"
pro = (1 - pro) if status == "negative" else pro
print(f'Predicted sentiment is {status} with a probability of {pro}')
```

```
Taut and organically gripping, Edward Dmytryk's Crossfire is a distinctive suspense thriller, an unlikely "message" movie using the look and devices of the noir cycle.<br /><br />Bivouacked in Washington, DC, a company of soldiers cope with their restlessness by hanging out in bars.
======================================================================
Actual sentiment is  : positive
======================================================================
Predicted sentiment is positive with a probability of 0.9499144554138184
/usr/local/lib/python3.7/dist-packages/torch/nn/modules/rnn.py:692: UserWarning: RNN module weights are not part of single contiguous chunk of memory. This means they need to be compacted at every call, possibly greatly increasing memory usage. To compact weights again call flatten_
  self.dropout, self.training, self.bidirectional, self.batch_first)
```

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 7.39 ⁜ Vol. 11, Issue 4, April 2022

DOI: 10.17148/IJARCCE.2022.11419

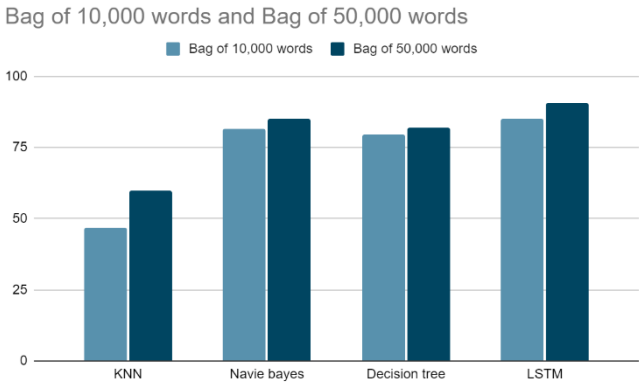
Flask Web App

After we've created this model, we'll build a flask web application and deploy it to the cloud.

IV. EXPERIMENTATION AND RESULT

As discussed above, we have tried multiple classification algorithms like Decision tree, Naive Bayes, KNN, LSTM on our movie review dataset containing movie reviews and associated sentiment. While working we can get a result that LSTM showed the best result among the tiered algorithms and KNN showed the worst result approximately 60%. The accuracy was in order LSTM> Naive Bayes > Decision Tree> KNN. When we found that the accuracy of LSTM was higher than others, we tried hyper tuning of parameters like learning rate, number of epochs. When we tried the number of epochs = 5 and learning rate as 0.1 accuracy was 86% and when hyper tuned the parameter and put epochs=10 and learning rate =0.01 accuracy came out 90.9%.

	LSTM	Naive Bayes	Decision Tree	KNN
Bag of words of 10,000	85.1	81.5	79.7	46.7
Bag of 50,000	90.9	85	82	60



In the above graph, we have made a comparison between different algorithms on different amounts of words and we concluded when we increased the number of words LSTM performed well. While training our model on LSTM we also performed some hyper tuning of parameters like learning rate and several epochs etc.

Experiment

```
def preprocess_string(s):
    s = re.sub(r"[^\w\s]", '', s)
    s = re.sub(r"\s+", '', s)
    s = re.sub(r"\d", '', s)

    return s
def tockenize(x_train,y_train,x_val,y_val):
    word_list = []

    stop_words = set(stopwords.words('english'))
    for sent in x_train:
        for word in sent.lower().split():
            word = preprocess_string(word)
            if word not in stop_words and word != '':
                word_list.append(word)

    corpus = Counter(word_list)

    corpus_ = sorted(corpus,key=corpus.get,reverse=True)[:1000]

    onehot_dict = {w:i+1 for i,w in enumerate(corpus_)}


    final_list_train,final_list_test = [],[]
    for sent in x_train:
            final_list_train.append([onehot_dict[preprocess_string(word)] for word in sent.lower().split()
                                     if preprocess_string(word) in onehot_dict.keys()])
    for sent in x_val:
            final_list_test.append([onehot_dict[preprocess_string(word)] for word in sent.lower().split()
                                    if preprocess_string(word) in onehot_dict.keys()])

    encoded_train = [1 if label =='positive' else 0 for label in y_train]
    encoded_test = [1 if label =='positive' else 0 for label in y_val]
    return np.array(final_list_train), np.array(encoded_train),np.array(final_list_test), np.array(encoded_test),onehot_dict
```

```
[ ]  x_train,y_train,x_test,y_test,vocab = tockenize(x_train,y_train,x_test,y_test)
```

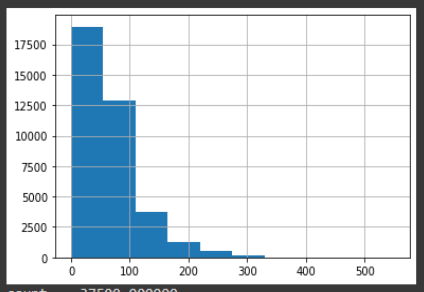
IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 7.39 ⁜ Vol. 11, Issue 4, April 2022

DOI: 10.17148/IJARCCE.2022.11419

```
[ ] x_train,y_train,x_test,y_test,vocab = tockenize(x_train,y_train,x_test,y_test)

    /usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:34: VisibleDeprecation

    print(f'Length of vocabulary is {len(vocab)}')

    Length of vocabulary is 1000

[ ] rev_len = [len(i) for i in x_train]
    pd.Series(rev_len).hist()
    plt.show()
    pd.Series(rev_len).describe()
```




```
count    37500.000000
mean        69.013973
std         47.853196
min          0.000000
25%         39.000000
50%         54.000000
75%         84.000000
max        549.000000
dtype: float64
```

```
[ ]
    def padding_(sentences, seq_len):
        features = np.zeros((len(sentences), seq_len),dtype=int)
        for ii, review in enumerate(sentences):
            if len(review) != 0:
                features[ii, -len(review):] = np.array(review)[:seq_len]
        return features

[ ] x_train_pad = padding_(x_train,500)
    x_test_pad = padding_(x_test,500)

[ ] train_data = TensorDataset(torch.from_numpy(x_train_pad), torch.from_numpy(y_train))
    valid_data = TensorDataset(torch.from_numpy(x_test_pad), torch.from_numpy(y_test))

    batch_size = 50


    train_loader = DataLoader(train_data, shuffle=True, batch_size=batch_size)
    valid_loader = DataLoader(valid_data, shuffle=True, batch_size=batch_size)

    dataiter = iter(train_loader)
    sample_x, sample_y = dataiter.next()

    print('Sample input size: ', sample_x.size()) # batch_size, seq_length
    print('Sample input: \n', sample_x)
    print('Sample input: \n', sample_y)

    Sample input size:  torch.Size([50, 500])
    Sample input:
     tensor([[  0,   0,   0,  ..., 225,  19,  91],
            [  0,   0,   0,  ...,  14, 287,  14],
            [  0,   0,   0,  ...,  45, 311, 857],
            ...,
            [  0,   0,   0,  ..., 357,  51, 276],
            [  0,   0,   0,  ...,  81, 239, 300],
            [  0,   0,   0,  ..., 258,   2, 327]])
    Sample input:
     tensor([0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1,
```

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 7.39 ⁜ Vol. 11, Issue 4, April 2022

DOI: 10.17148/IJARCCE.2022.11419

```python
class SentimentRNN(nn.Module):
    def __init__(self,no_layers,vocab_size,hidden_dim,embedding_dim,drop_prob=0.5):
        super(SentimentRNN,self).__init__()

        self.output_dim = output_dim
        self.hidden_dim = hidden_dim

        self.no_layers = no_layers
        self.vocab_size = vocab_size

        # embedding and LSTM layers
        self.embedding = nn.Embedding(vocab_size, embedding_dim)

        #lstm
        self.lstm = nn.LSTM(input_size=embedding_dim,hidden_size=self.hidden_dim,
                            num_layers=no_layers, batch_first=True)


        # dropout layer
        self.dropout = nn.Dropout(0.3)

        # linear and sigmoid layer
        self.fc = nn.Linear(self.hidden_dim, output_dim)
        self.sig = nn.Sigmoid()

    def forward(self,x,hidden):
        batch_size = x.size(0)
        # embeddings and lstm_out
        embeds = self.embedding(x)  # shape: B x S x Feature   since batch = True
        #print(embeds.shape)  #[50, 500, 1000]
        lstm_out, hidden = self.lstm(embeds, hidden)

        lstm_out = lstm_out.contiguous().view(-1, self.hidden_dim)

        # dropout and fully connected layer
        out = self.dropout(lstm_out)
        out = self.fc(out)

        # sigmoid function
        sig_out = self.sig(out)

        # reshape to be batch_size first
        sig_out = sig_out.view(batch_size, -1)

        sig_out = sig_out[:, -1] # get last batch of labels

        # return last sigmoid output and hidden state
        return sig_out, hidden
```

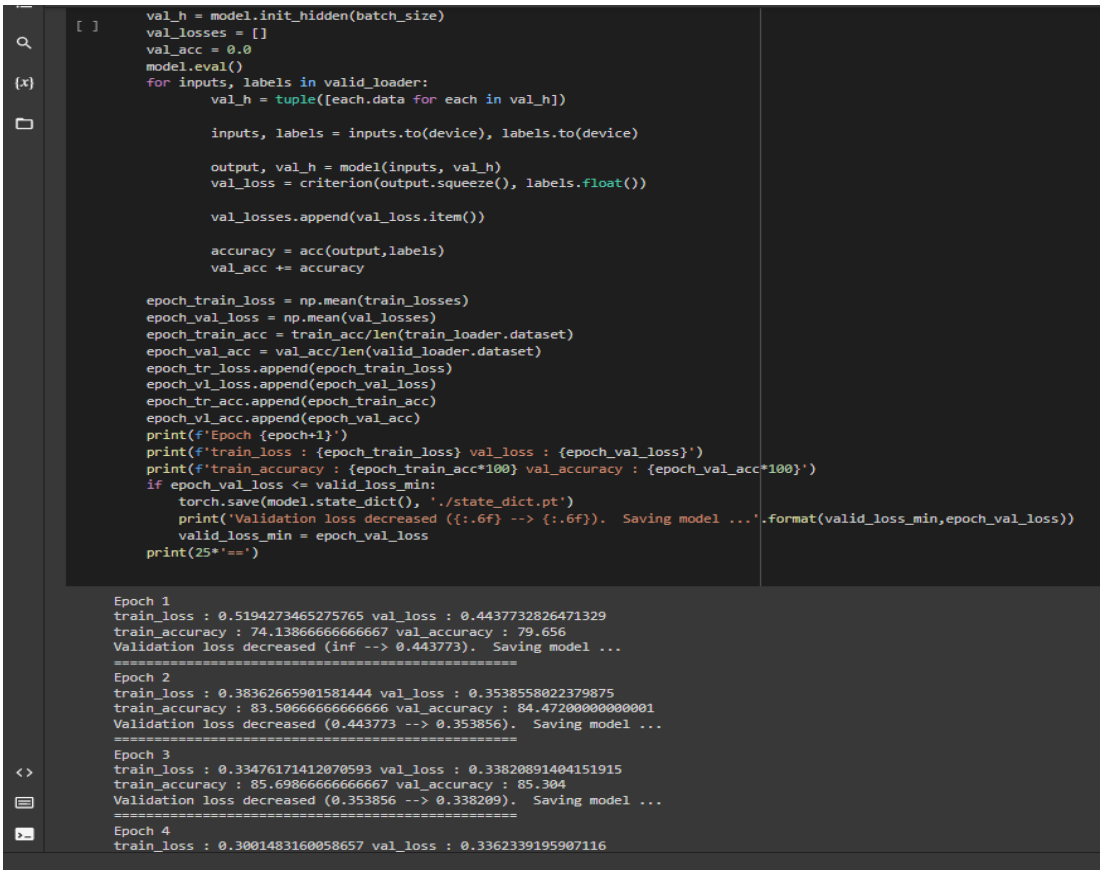
```python
[ ] clip = 5
    epochs = 7
    valid_loss_min = np.Inf

    epoch_tr_loss,epoch_vl_loss = [],[]
    epoch_tr_acc,epoch_vl_acc = [],[]

    for epoch in range(epochs):
        train_losses = []
        train_acc = 0.0
        model.train()
        # initialize hidden state
        h = model.init_hidden(batch_size)
        for inputs, labels in train_loader:

            inputs, labels = inputs.to(device), labels.to(device)
            # Creating new variables for the hidden state, otherwise
            # we'd backprop through the entire training history
            h = tuple([each.data for each in h])

            model.zero_grad()
            output,h = model(inputs,h)

            # calculate the loss and perform backprop
            loss = criterion(output.squeeze(), labels.float())
            loss.backward()
            train_losses.append(loss.item())
            # calculating accuracy
            accuracy = acc(output,labels)
            train_acc += accuracy
            #`clip_grad_norm` helps prevent the exploding gradient problem in RNNs / LSTMs.
            nn.utils.clip_grad_norm_(model.parameters(), clip)
            optimizer.step()


        val_h = model.init_hidden(batch_size)
        val_losses = []
        val_acc = 0.0
        model.eval()
        for inputs, labels in valid_loader:
                val_h = tuple([each.data for each in val_h])

                inputs, labels = inputs.to(device), labels.to(device)

                output, val_h = model(inputs, val_h)
                val_loss = criterion(output.squeeze(), labels.float())

                val_losses.append(val_loss.item())
```

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

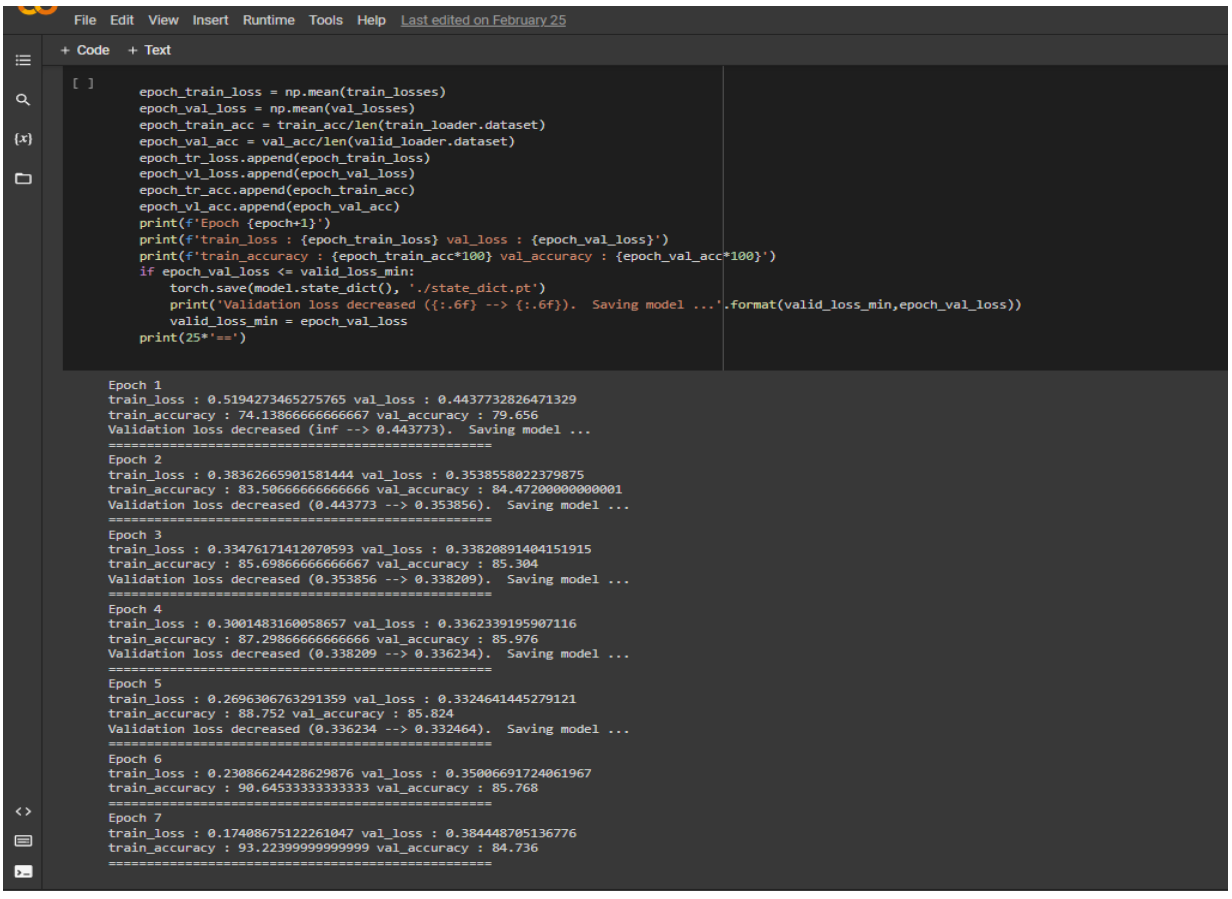
Impact Factor 7.39 ⁜ Vol. 11, Issue 4, April 2022

DOI: 10.17148/IJARCCE.2022.11419



```
[ ]
        val_h = model.init_hidden(batch_size)
        val_losses = []
        val_acc = 0.0
        model.eval()
        for inputs, labels in valid_loader:
                val_h = tuple([each.data for each in val_h])

                inputs, labels = inputs.to(device), labels.to(device)

                output, val_h = model(inputs, val_h)
                val_loss = criterion(output.squeeze(), labels.float())

                val_losses.append(val_loss.item())

                accuracy = acc(output,labels)
                val_acc += accuracy

        epoch_train_loss = np.mean(train_losses)
        epoch_val_loss = np.mean(val_losses)
        epoch_train_acc = train_acc/len(train_loader.dataset)
        epoch_val_acc = val_acc/len(valid_loader.dataset)
        epoch_tr_loss.append(epoch_train_loss)
        epoch_vl_loss.append(epoch_val_loss)
        epoch_tr_acc.append(epoch_train_acc)
        epoch_vl_acc.append(epoch_val_acc)
        print(f'Epoch {epoch+1}')
        print(f'train_loss : {epoch_train_loss} val_loss : {epoch_val_loss}')
        print(f'train_accuracy : {epoch_train_acc*100} val_accuracy : {epoch_val_acc*100}')
        if epoch_val_loss <= valid_loss_min:
            torch.save(model.state_dict(), './state_dict.pt')
            print('Validation loss decreased ({:.6f} --> {:.6f}).  Saving model ...'.format(valid_loss_min,epoch_val_loss))
            valid_loss_min = epoch_val_loss
        print(25*'==')
```

```
Epoch 1
train_loss : 0.5194273465275765 val_loss : 0.4437732826471329
train_accuracy : 74.13866666666667 val_accuracy : 79.656
Validation loss decreased (inf --> 0.443773).  Saving model ...
==================================================
Epoch 2
train_loss : 0.38362665901581444 val_loss : 0.3538558022379875
train_accuracy : 83.50666666666666 val_accuracy : 84.47200000000001
Validation loss decreased (0.443773 --> 0.353856).  Saving model ...
==================================================
Epoch 3
train_loss : 0.33476171412070593 val_loss : 0.33820891404151915
train_accuracy : 85.69866666666667 val_accuracy : 85.304
Validation loss decreased (0.353856 --> 0.338209).  Saving model ...
==================================================
Epoch 4
train_loss : 0.3001483160058657 val_loss : 0.3362339195907116
```



File Edit View Insert Runtime Tools Help Last edited on February 25

+ Code + Text

```
[ ]
        epoch_train_loss = np.mean(train_losses)
        epoch_val_loss = np.mean(val_losses)
        epoch_train_acc = train_acc/len(train_loader.dataset)
        epoch_val_acc = val_acc/len(valid_loader.dataset)
        epoch_tr_loss.append(epoch_train_loss)
        epoch_vl_loss.append(epoch_val_loss)
        epoch_tr_acc.append(epoch_train_acc)
        epoch_vl_acc.append(epoch_val_acc)
        print(f'Epoch {epoch+1}')
        print(f'train_loss : {epoch_train_loss} val_loss : {epoch_val_loss}')
        print(f'train_accuracy : {epoch_train_acc*100} val_accuracy : {epoch_val_acc*100}')
        if epoch_val_loss <= valid_loss_min:
            torch.save(model.state_dict(), './state_dict.pt')
            print('Validation loss decreased ({:.6f} --> {:.6f}).  Saving model ...'.format(valid_loss_min,epoch_val_loss))
            valid_loss_min = epoch_val_loss
        print(25*'==')
```

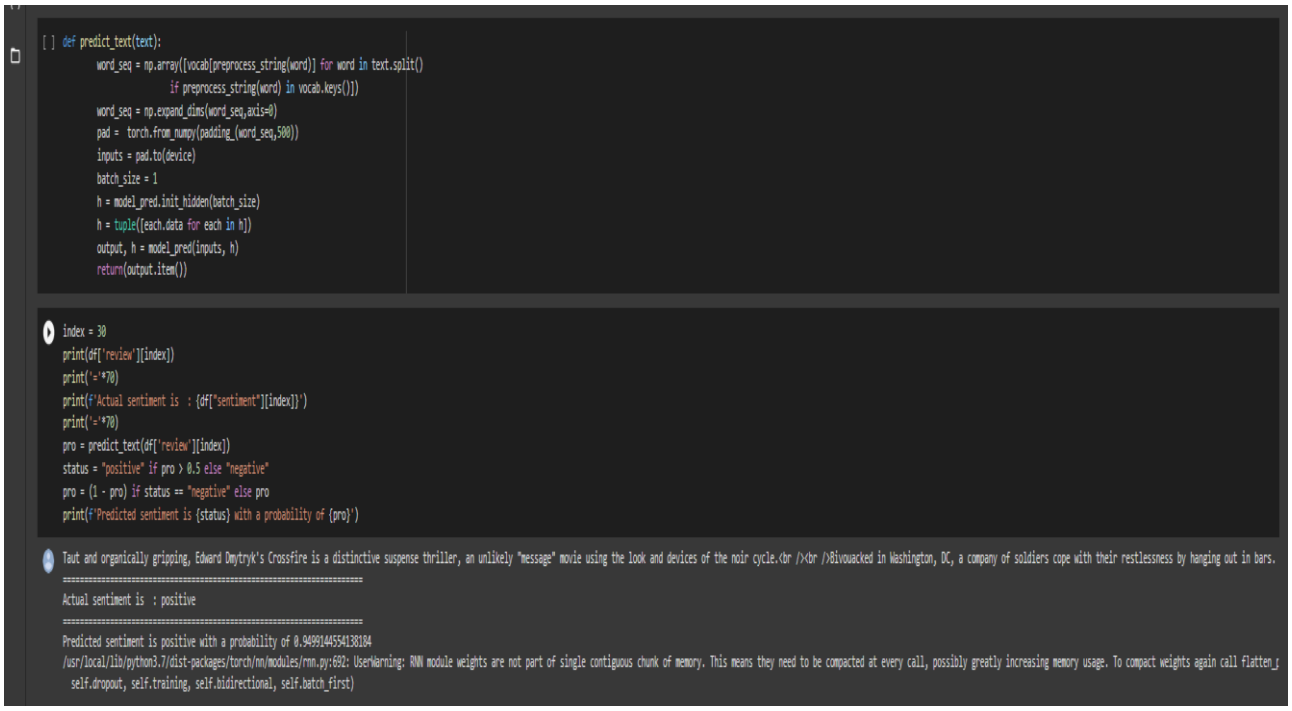
```
Epoch 1
train_loss : 0.5194273465275765 val_loss : 0.4437732826471329
train_accuracy : 74.13866666666667 val_accuracy : 79.656
Validation loss decreased (inf --> 0.443773).  Saving model ...
==================================================
Epoch 2
train_loss : 0.38362665901581444 val_loss : 0.3538558022379875
train_accuracy : 83.50666666666666 val_accuracy : 84.47200000000001
Validation loss decreased (0.443773 --> 0.353856).  Saving model ...
==================================================
Epoch 3
train_loss : 0.33476171412070593 val_loss : 0.33820891404151915
train_accuracy : 85.69866666666667 val_accuracy : 85.304
Validation loss decreased (0.353856 --> 0.338209).  Saving model ...
==================================================
Epoch 4
train_loss : 0.3001483160058657 val_loss : 0.3362339195907116
train_accuracy : 87.29866666666666 val_accuracy : 85.976
Validation loss decreased (0.338209 --> 0.336234).  Saving model ...
==================================================
Epoch 5
train_loss : 0.2696306763291359 val_loss : 0.3324641445279121
train_accuracy : 88.752 val_accuracy : 85.824
Validation loss decreased (0.336234 --> 0.332464).  Saving model ...
==================================================
Epoch 6
train_loss : 0.23086624428629876 val_loss : 0.35006691724061967
train_accuracy : 90.64533333333333 val_accuracy : 85.768
==================================================
Epoch 7
train_loss : 0.17408675122261047 val_loss : 0.384448705136776
train_accuracy : 93.22399999999999 val_accuracy : 84.736
==================================================
```

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 7.39 ⁜ Vol. 11, Issue 4, April 2022

DOI: 10.17148/IJARCCE.2022.11419



V. CONCLUSION

In this research, various methods were used to find the sentiments of the movie reviews. The algorithms performed were LSTM, KNN, Naïve Bayes, Decision Tree. The finest results were given by the LSTM. The LSTM classifier achieved 90.90% accuracy, the Naïve Bayes classifier achieved 85% accuracy, the Decision Tree classifier achieved 80.2% accuracy, and the KNN classifier achieved 60% accuracy.

As of now, only a few algorithms have been tested, it is necessary to test other algorithms or create mixed methods to increase the accuracy of the results. Finding the sentiments of the reviews can help in a variety of domains. Smart systems can be developed which can provide the users with a complete overview of movies, products, services, etc. without requiring the user to review individual reviews, he or she can make direct decisions based on the results provided by the smart systems.

VI. ACKNOWLEDGEMENT

Thanks to all the persons who have contributed directly or indirectly to the research.

VI. REFERENCES

- C. D. Santos and M. Gatti. “Deep convolutional neural networks for sentiment analysis of short texts.” In Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers, 2014, pp. 69-78.
- I. Hemalatha, G. P. S. Varma, and A. Govardhan. “Preprocessing the informal text for efficient sentiment analysis.” International Journal of Emerging Trends & Technology in Computer Science (IJETTCS) 1, no. 2: pp.58-61. 2012.
- A. Kennedy and D. Inkpen. “Sentiment classification of movie reviews using contextual valence shifters.” Computational intelligence, Vol. 22, no. 2, pp.110-125. 2006
- Joscha Markle-Hub, Stefan Feuer Riegel, Helmut Preminger. Improving Sentiment Analysis with Document-Level Semantic Relationships from Rhetoric Discourse Structures, Proceedings of the 50th Hawaii International Conference on System Sciences, 2017.
- A. Hogeboom, F. Frasnian, F. de Jong, and U. Kaymak, Using Rhetorical Structure in Sentiment Analysis, Communications of the ACM, vol. 58, no. 7, pp. 69–77, 2015.
- Dhiraj Murthy, Twitter, and elections: are tweets, predictive, reactive, or a form of buzz Information, Communication & Society, 18:7, 816-831, DOI:10.1080/1369118X.2015.1006659, 2016.
- Bo Pang and Lillian Lee and Shivakumar Vaithyanathan “Thumbs up Sentiment Classification using Machine Learning Techniques”, Language Processing (EMNLP), Philadelphia, pp. 79-86, July 2002

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 7.39 ⁜ Vol. 11, Issue 4, April 2022

DOI: 10.17148/IJARCCE.2022.11419

- Kamal, Subjectivity Classification using Machine Learning Techniques for Mining Feature Opinion Pairs from Web Opinion Sources. International Journal of Computer Science Issues 10(5), 191- 200, A. 2013.
- H. Dai, J. Hao, "Mining social media data for opinion polarities about electronic cigarettes" Tobacco control, 26(2), 175-180, 2017, DOI:1136/Tobacco Control 2015-052818.
- Akshay Amolik, Niketan Jivani, Mahavir Bhandari, Dr.M. Venkatesan, Twitter Sentiment Analysis of Movie Reviews Using Machine Learning Techniques, School of Computer Science and Engineering, VIT University, Vellore.
- Humera Shaniya, Kavitha, Raniah Zaheer, 2015, Text Categorization of Movie Reviews for Sentiment International Journal of Computer Applications (0975 – 8887) Volume 179 – No.7, Analysis, International Journal of Innovative Research in Science, Engineering and Technology, Vol. 4, Issue11, December 2017.

