Character-level Convolutional Networks for Text Classification

Text Classification

Text Classification is used to allocate predefined categories to free text. Almost any type of text can be organized, structured, and categorized using text classifiers, including texts, medical research, files, and web content.

Text Classification increases the Scalability as manually doing it is very much time-consuming. In just a few minutes, machine learning using text classification can help organize millions of surveys, comments, emails, etc. Real-time Analysis is processed.

Automated Text Classification can be approached by Machine learning-based systems.

Text classification using machine learning gains the ability to classify information based on prior observations. Machine learning algorithms can learn the various associations between textual fragments and that a specific output is expected for a specific input by using pre-labeled examples as training data.

Support vector machines (SVM), deep learning, and the Naive Bayes algorithms are a few of the most widely used text classification techniques.

Text Classification is an automated process of classifying text into categories. As we knew Text Classification can be done with the help of Different algorithms such as

- Naive Bayes
- Character level Neural Networks
- Character level CNN with LSTM

Text Classification using Character level CNN

Overview

- The paper discusses using character-level CNN for text classification.
- Character level CNN is no different from CNN (in CV) except that in the case of Image classification, we perform 2D convolution whereas, for text classification, we perform 1D convolution.
- The main problem when CNN is used for text is variable input size i.e, we find sentences of different lengths but CNN accepts input of fixed length. So, the text is transformed into fixed-length vectors, i.e, all documents now have the same dimensions.
- We pass the matrix (i.e, document) through 6 convolution layers and 3 fully connected layers. The output vector from the last fully connected layer has size m x 1 where m is the number of possible classes. The vector is normalized and the class corresponding to the maximum value in the vector is considered its underlying class.
- The model learns different weights using stochastic gradient descent and backpropagation

Character Quantization (Converting text to vectors)

- A character is represented using 1-of-m encoding (or One hot encoding). Here, m= 70. i.e, 70 possible characters are considered. These include letters, numbers (0-9), and a few special characters. Spaces and characters other than these are encoded as zero vectors.

- In this way, each character is encoded as a vector of length 70 and it encodes l characters. Any character exceeding l is ignored and less than l is padded with appropriate zero vectors. (l = 1014)

Model Design

- 2 ConvNets are designed.They are both 9 layers deep with 6 convolutional layers and 3 fully-connected layers.

- 2 dropout layers are included between fully connected layers to regularize with a probability of 0.5.

- ConvNet → a.256 filters →b. 1024 filters

layer	Input Size	ConvNet(#filters)	Kernel	Output	Pool	Output
1	70x1014x1	256	70x7	1x1008x256	1x3	1x336x256
2	1x336x256	256	1x7	1x330x256	1x3	1x110x256
3	1x110x256	256	1x3	1x108x256		
4	1x108x256	256	1x3	1x106x256		
5	1x106x256	256	1x3	1x104x256		
6	1x104x256	256	1x3	1x102x256	1x3	1x34x256

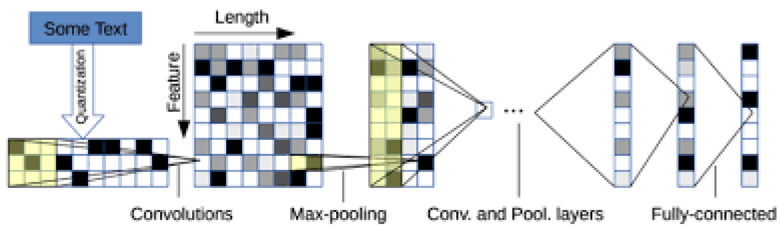
Then, the output from the sixth layer is passed through 3 fully connected layers.

- 2 Dropout layers are added to avoid overfitting.

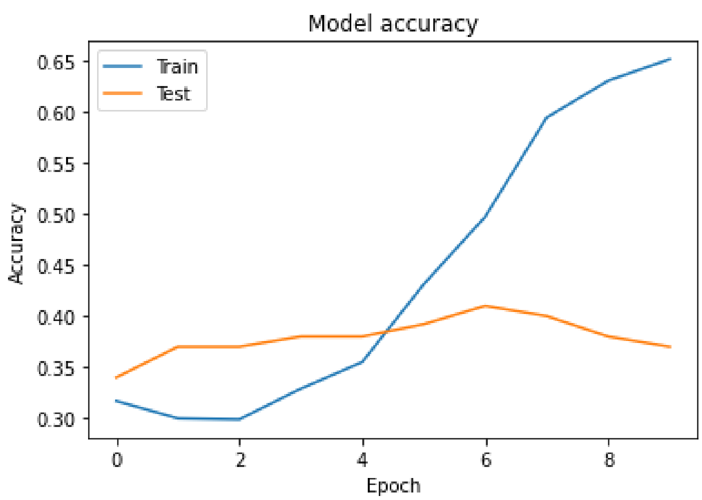
Input	No. of units	Output dimension
8074x1	1024	1024x1
1024x1	1024	1024x1
1024x1	M (num of classes)	mx1

The above table summarizes the dimensions of input at each layer.

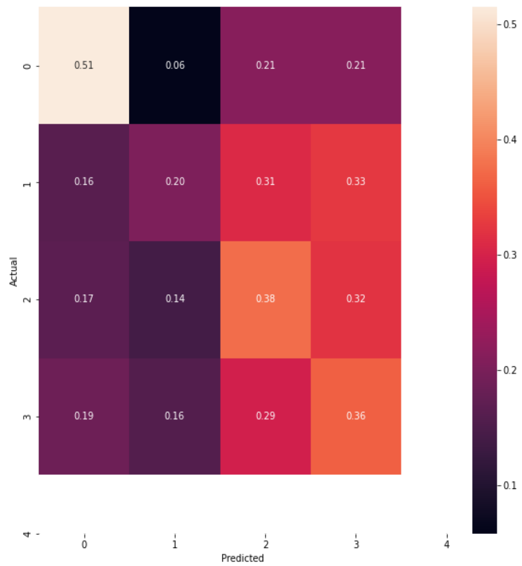
We get an mx1 vector, where each number represents the probability of belonging to that class (after applying softmax).



The above image depicts the character-level CNN architecture.



In the code, we have trained the model for 10 epochs and in the sixth epoch, we encountered higher accuracy values. We could also infer from the above plot that as we have trained the model for the number of epochs, the accuracy on the training data increased while the test accuracy decreased after reaching a peak as the model started to overfit when the number of epochs increased.



The above picture shows a normalized confusion matrix. we can infer that the model has correctly predicted class-0: 51% of times, correctly predicted class-1 : 20% of the times, correctly predicted class-3 : 38% of the times, correctly predicted class-4: 36% of the times.

Hyper Parameters:

Optimizer - Adam

Loss Function - Categorical Cross Entropy loss

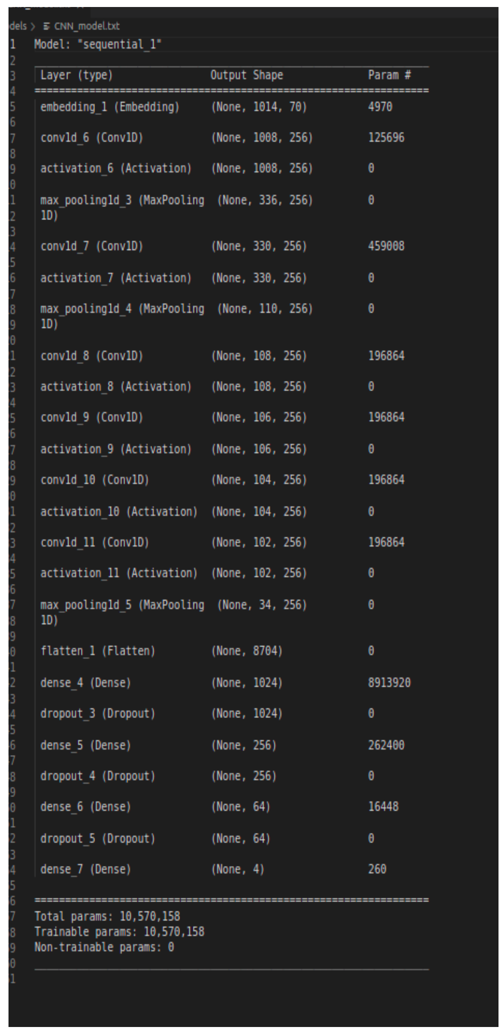
Batch Size - 128

Number of Epochs - 10

Convolution Layers - 6

Pooling layers - 3

Fully Connected Layers - 3



```
dels > CNN_model.txt
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_1 (Embedding)     (None, 1014, 70)          4970

 conv1d_6 (Conv1D)           (None, 1008, 256)         125696

 activation_6 (Activation)   (None, 1008, 256)         0

 max_pooling1d_3 (MaxPooling  (None, 336, 256)         0
 1D)

 conv1d_7 (Conv1D)           (None, 330, 256)          459008

 activation_7 (Activation)   (None, 330, 256)          0

 max_pooling1d_4 (MaxPooling  (None, 110, 256)         0
 1D)

 conv1d_8 (Conv1D)           (None, 108, 256)          196864

 activation_8 (Activation)   (None, 108, 256)          0

 conv1d_9 (Conv1D)           (None, 106, 256)          196864

 activation_9 (Activation)   (None, 106, 256)          0

 conv1d_10 (Conv1D)          (None, 104, 256)          196864

 activation_10 (Activation)  (None, 104, 256)          0

 conv1d_11 (Conv1D)          (None, 102, 256)          196864

 activation_11 (Activation)  (None, 102, 256)          0

 max_pooling1d_5 (MaxPooling  (None, 34, 256)          0
 1D)

 flatten_1 (Flatten)         (None, 8704)              0

 dense_4 (Dense)             (None, 1024)              8913920

 dropout_3 (Dropout)         (None, 1024)              0

 dense_5 (Dense)             (None, 256)               262400

 dropout_4 (Dropout)         (None, 256)               0

 dense_6 (Dense)             (None, 64)                16448

 dropout_5 (Dropout)         (None, 64)                0

 dense_7 (Dense)             (None, 4)                 260

=================================================================
Total params: 10,570,158
Trainable params: 10,570,158
Non-trainable params: 0
_________________________________________________________________
```

Text Classification using Character Naive Bayes

Overview

Naive Bayes depends on conditional probabilities, in general,Adam which are easy to implement and evaluate, which does not require an iterative process. NB supports binary classification as well as multinomial one. NB assumes that any feature value is independent of the value of the other features. Because of the independence assumption, NB doesn’t need to learn all possible correlations between the features. if N is the number of features, then a general algorithm requires 2N possible feature interactions, while NB only needs the order of N data points. Thus, NB classifiers can learn easier from small training data sets due to the class independence assumption.

Implementation:

Data Preprocessing: Basically transforming raw data into an understanding format. As the obtained raw data is always incomplete, incompatible, or lacking in certain behaviors. So in order to resolve all such issues we do preprocessing. Which will help in providing better results through various classification algorithms.

Techniques:

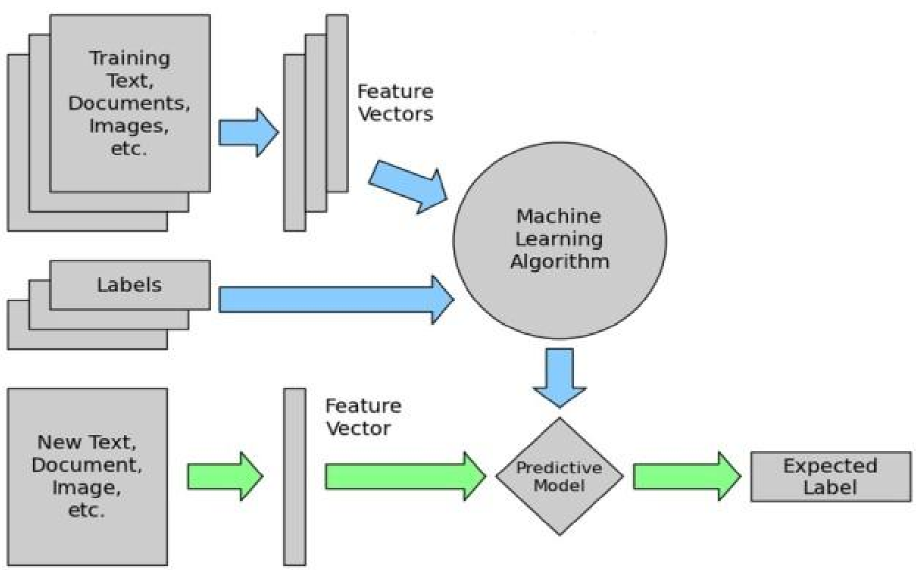
1. Tokenization is the division of text into tokens such as words, phrases, symbols, or other objects. For additional processing, the list of tokens serves as input. $word\_tokenize$ and $sent\_tokenize$ from the NLTK Library make it simple to separate a stream of text into a list of words or a list of sentences, respectively.
2. Lemmatization: each word's inflectional variants should be reduced to a single base or root. Stemming and lemmatization are closely linked. However, stemmers work on a single word without taking the context into account. Without that information, kids are unable to distinguish between words that have various meanings depending on the part of speech. Stemmers are often quicker and easier to deploy, therefore for some applications, the decreased accuracy may not be a factor.

When performing the data processing, remove the empty rows in the data, and will change all the text into lower letters, and perform the tokenization, and remove the non-alpha text, stop words, and deal with the word lemmatization.

Training and testing data sets: Splitting the data sets into training and testing sets, as we the training data set to perform fitting and test data set to perform the predictions.

Word Vectorization: it's a process of converting text documents into numerical feature vectors. One of the important methods by which this can be done is by TF-IDF (word frequency scores that try to highlight words which are important). With the help of this TF-IDF model we can construct a vocabulary of words , and will allot unique integer numbers to each of these words.

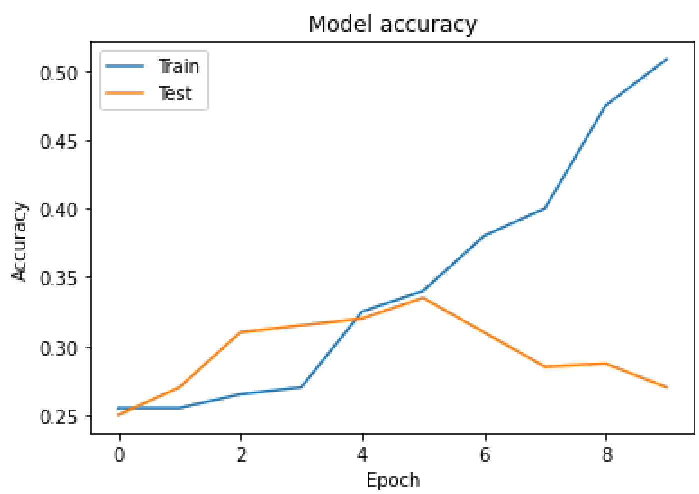
Upon performing all these methods we can obtain the vectorized data of a given input data set, which will be ready to perform different classification.

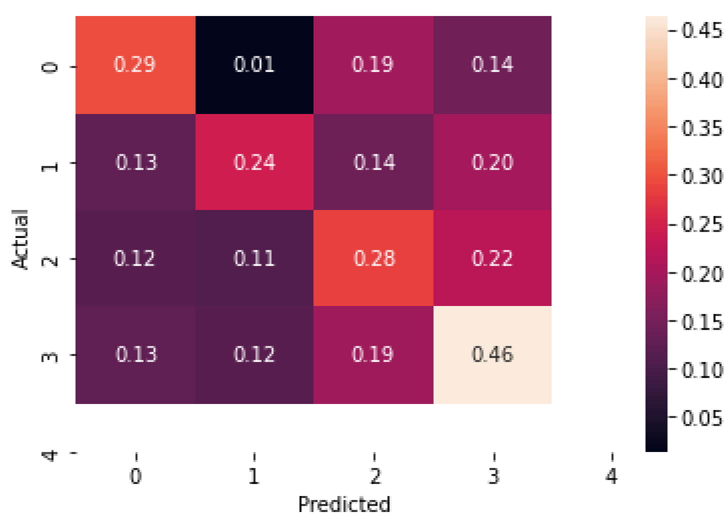


The architecture of Naive-Bayes model

I

In the code, we have trained the model for 10 epochs and in the fifth epoch, we encountered higher accuracy values. We could also infer from the above plot that as we have trained the model for more epochs, the accuracy on the training data increased while the test accuracy decreased after reaching a peak as the model started to overfit when the number of epochs increased.

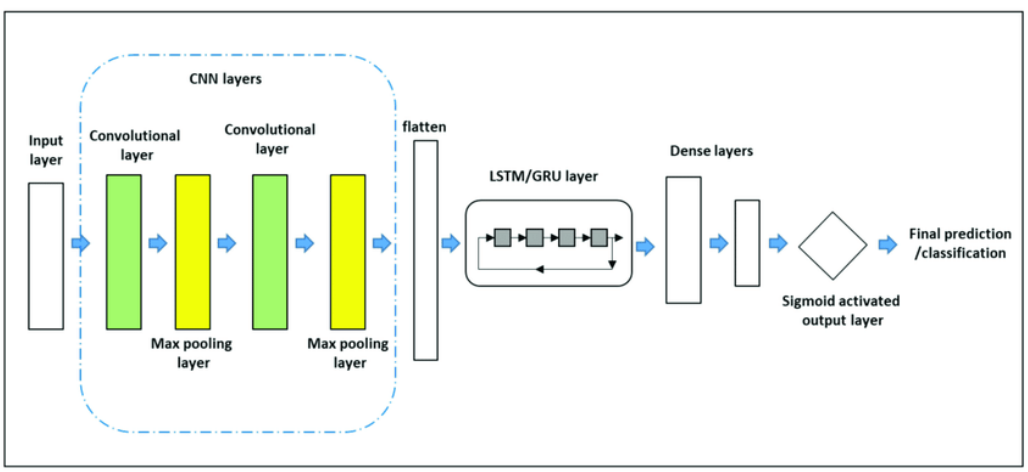




The above picture shows a normalized confusion matrix. we can infer that the model has correctly predicted class-0: 29% of the times, correctly predicted class-1: 24% of the times, correctly predicted class-3: 28% of the times, correctly predicted class-4: 46% of the times.

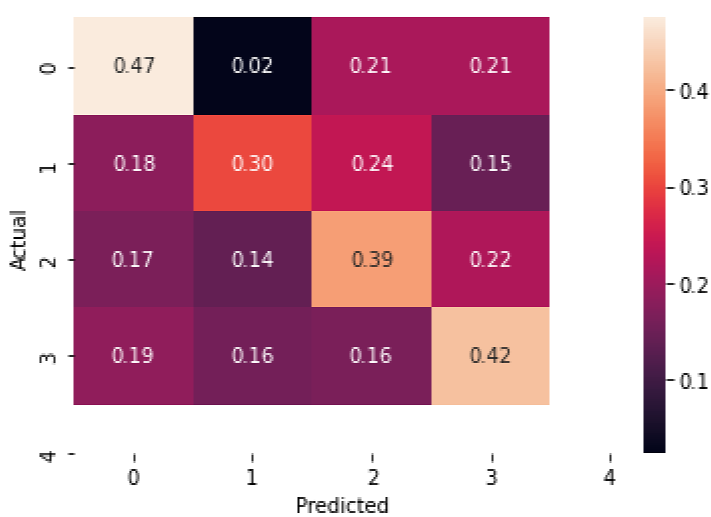
Text Classification using Character Level CNN and LSTM architecture

We have fine-tuned the baseline model of character level CNN by adding an LSTM layer on top of it in order to improve the accuracy and followed by the dense layers. Overall, we used CNN model for extracting features and LSTM architecture to interpret the features. Through the CNN model we are able to understand the patterns in the text like “It’s bad”, “I love” irrespective of their position in their sentence, so the text analysis through Convolutional Neural Network can extract the important features of text using pooling but it would be difficult to extract the contextual information using this CNN model, so in order to solve this issue we have fine-tuned the baseline model by adding LSTM layer as it captures the context. So the combination of these two sub-models CNN and LSTM architecture has improved the overall accuracy.

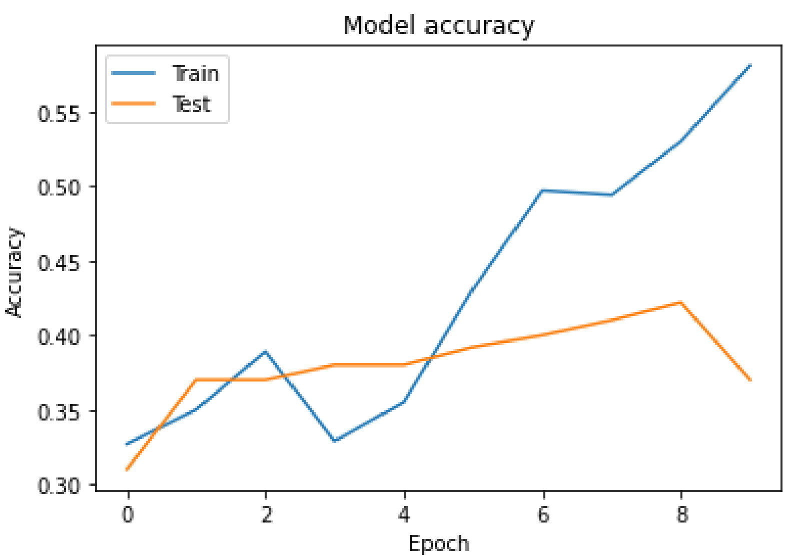


The architecture of Character level CNN with LSTM

```
dels > CNN_LSTM_model.txt
(None, 34, 256)
Model: "model_1"

 Layer (type)                Output Shape              Param #
=================================================================
 input (InputLayer)          [(None, 1014)]            0

 embedding_2 (Embedding)     (None, 1014, 70)          4970

 conv1d_9 (Conv1D)           (None, 1008, 256)         125696

 activation_9 (Activation)   (None, 1008, 256)         0

 max_pooling1d_6 (MaxPooling  (None, 336, 256)         0
 1D)

 conv1d_10 (Conv1D)          (None, 330, 256)          459008

 activation_10 (Activation)  (None, 330, 256)          0

 max_pooling1d_7 (MaxPooling  (None, 110, 256)         0
 1D)

 conv1d_11 (Conv1D)          (None, 108, 256)          196864

 activation_11 (Activation)  (None, 108, 256)          0

 conv1d_12 (Conv1D)          (None, 106, 256)          196864

 activation_12 (Activation)  (None, 106, 256)          0

 conv1d_13 (Conv1D)          (None, 104, 256)          196864

 activation_13 (Activation)  (None, 104, 256)          0

 conv1d_14 (Conv1D)          (None, 102, 256)          196864

 activation_14 (Activation)  (None, 102, 256)          0

 max_pooling1d_8 (MaxPooling  (None, 34, 256)          0
 1D)

 lstm_1 (LSTM)               (None, 128)               197120

 dense_3 (Dense)             (None, 32)                4128

 dropout_2 (Dropout)         (None, 32)                0

 dense_4 (Dense)             (None, 16)                528

 dropout_3 (Dropout)         (None, 16)                0

 dense_5 (Dense)             (None, 4)                 68

=================================================================
Total params: 1,578,974
Trainable params: 1,578,974
Non-trainable params: 0
```



The above picture shows a normalized confusion matrix. we can infer that the model has correctly predicted class 0: 40% of the time, correctly predicted class 1: 30% of the time, correctly predicted class 3: 39% of the time, correctly predicted class 4: 42% of the time



In the code, we have trained the model for 10 epochs and in the eight epochs, we encountered higher accuracy values. We could also infer from the above plot that as we have trained the model for more epochs, the accuracy on the training data increased after the fourth epoch while the test accuracy decreased after reaching a peak as the model started to overfit when the number of epochs increased.

Hyper Parameters:

Optimizer - Ada

Loss Function - Categorical Cross Entropy Loss

Batch Size - 12807

Number of Epochs - 10

Convolution Layers - 6

Pooling layers - 3

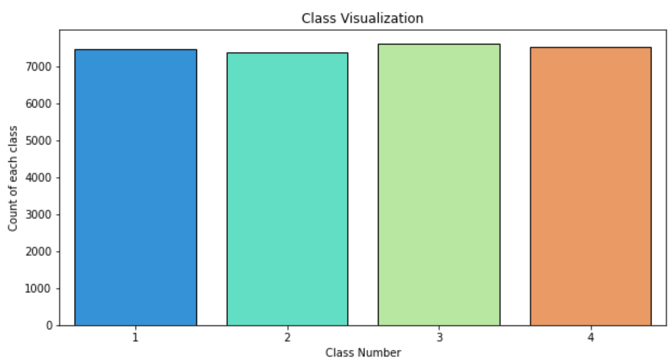
LSTM Layers - 1

Fully Connected Layers - 3

Dataset

To implement this, we have used AG news corpus dataset. Each AG's news sample can be segregated into one of the 4 classes. (Business, Sci/Tech, World, Sports) . Each class contains 30,000 training samples and 1,900 testing samples. The total number of training samples is 120,000 and testing is 7,600. This is the dimension of the original dataset.

We've considered 30,000 samples from the training data set (because it's difficult to train 1,20,000 samples on CPU). While taking 30K samples from the training data set, we've made sure that each class gets ~7k samples (sampling is performed on each class).



Comparison of models

When we compare the Naive Bayes, CNN(Baseline), CNN with LSTM. Accuracy increases from 32 to 39 to 43. CNN can be used to reduce the number of parameters we need to train without sacrificing performance. NB is comparatively fast but accuracy is low. LSTM requires less parameters. While slow to train, their advantage comes from being able to look at long sequences of inputs without increasing the network size.

Issued Faced:

As the given training data set is very huge, i.e, 1.5 lakh, 4.5 lakh, 5-6 lakh, we weren't able to perform training using our CPUs. So we tried using an ada account as it will be able to run larger simulations comfortably. But the installations of TensorFlow were giving many errors in anaconda. So, we decided to consider 30k of the total training data and run it in our local systems as that's the only way to get results.

Why is AG news DataSet used?

We used AG news Dataset for training and testing purposes in all the approaches that we have implemented since it has four classes which is not too many like DBPedia which has 14 classes and unlike Amazon Review Polarity which has only two classes. As we took 30k samples for training purposes, it would be difficult to accurately classify if there are more classes as our model can't interpret such a low number of training samples. And in addition to this, relatively the classes of AG News corpus are diverse which makes a model to understand the patterns as we are restricted to less samples in the training set.

Conclusion:

In this project, we explored text classification by using different models: Naive Bayes, CNN, and CNN with LSTM. From the models, the accuracies that we observed are, Naive Bayes < CNN < CNN with LSTM.

We use the CNN model to extract features, and LSTM to interpret the features. Although convolutional neural networks can extract contextual information through pooling, that would be difficult. Hence LSTM layer is added to capture context. Hence CNN with LSTM improved Overall Accuracy

Contribution:

Dataset collection - Greeshma, Ananya Sudi and Manaswini

Naive Bayes - Manaswini, Ananya Sudi and Nandini

CNN - Greeshma, Ananya Sudi and Manaswini

CNN with LSTM - Greeshma, Ananya Sudi and Nandini

Report - Greeshma, Ananya Sudi, Manaswini and Nandini