Applied Machine Learning and Data Science Internship 2020, IIT Kanpur.
Capstone Project Report
Tweets Sentiment Analysis

Jatin Mishra

Abstract

In this report we will be discussing the problem statement "Tweets Sentiment Analysis", feasible approaches, then we will dive into data preprocessing and various models used. In this project, we will be trying to practically implement all the important Machine Learning concepts we have learned through this internship. We will try to model our data first on simpler ML algorithms like Logistic Regression, Random Forest, Decision Tree, etc and then we will try to use Deep Learning approaches like LSTMs and CNNS.

1 Introduction

Sentiment Analysis is one of the most important problems approached in Natural Language Processing as it has numerous real-world use cases like analyzing movie reviews, product reviews, customer query tagging, etc the list goes on. It can be used pretty much anywhere the requirement is analyzing textual data. The textual data can be things like tweets (as our problem here) or it can be sources like blogs, reviews, forums, social media, news, etc.

With the advent of deep learning the accuracy of understanding and analyzing language has greatly increased to a point where it can be used extensively and reliably in the majority of use cases like machine translation, sentiment analysis, text generation, text summarization, etc.

But even the State Of The Art models of present time like OpenAI's GPT and BERT (Bidirectional Encoder Representation from Transformers) are not at human level language analysis. Although these Deep learning models have pushed the boundaries of NLP making NLP one of the most exciting fields in Machine Learning for both research and commercial applications.

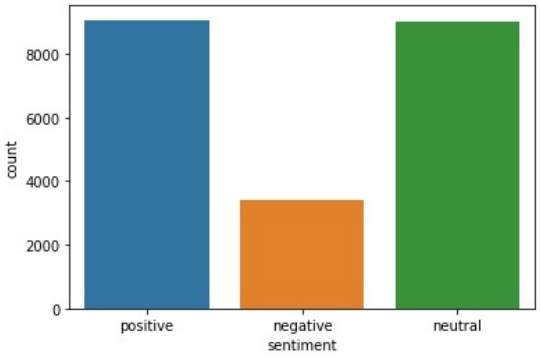
In this project, we will be trying to implement traditional Machine Learning algorithms like Logistic Regression, Decision Trees, Random Forest, etc and then we will model our data using Deep Learning approaches like LSTM (Long Short Term Memory) Networks, etc. We will also create a model combining CNNs (Convolutional Neural Networks) and LSTMs as CNNs famous for their application in Computer Vision have proved to perform very well in Natural Language Processing (Ye Zhang et al., 2017, Mathieu, 2017). We will discuss more about CNNs and their applications in NLP later. We will also be discussing the class imbalance problem in classification tasks, how it affects the model performance, approaches to tackle it like Oversampling (increasing the samples in classes with a lesser amount of sample), Downsampling (reducing the number of samples in the majority classes to match up with the number of samples in minority class). We will implement Random Over Sampler, this approach randomly duplicates samples from minority class and adds it to the training data. We will also briefly discuss about SMOTE (Synthetic Minority Over-Sampling Technique) (Chawla et al., 2002) "synthetic samples are generated in the following way: Take the difference between the feature vector (sample) under consideration and its nearest neighbor. Multiply this difference by a random number between 0 and 1, and add it to the feature vector under consideration. This causes the selection of a random point along the line segment between two specific features. This approach effectively forces the decision region of the minority class to become more general."

Finally, we will be comparing the results, we will be using the mean F score as the evaluation metric used in the Kaggle Competition for the same project.

2 Initial Data Inference

We will be using the training data provided in the associated Kaggle Competition.

The training data has 21465 samples divided among three classes 'neutral, 'positive', and 'negative'.



The positive class has 9064 samples, the neutral class has 9014 samples and the negative class has 3387 samples. The data shows class imbalance with the samples in negative class nearly 1/3 of the samples in positive and neutral classes. We will deal with this later.

3 Data Preprocessing.

Data preprocessing is a very important step in building a machine learning model because data preprocessing plays an important part in the performance of the model. Data preprocessing is essentially the process of modifying the data in such a form that it is best analyzable by the model. Also, to provide data to the model in a computationally efficient form. Generally, data preprocessing is the first step in process of building a model in NLP.

We used the following techniques in data preprocessing in this project:-

- Removing Unicode Characters.
- Convert text into lower case.
- HTML decoding (removing '&amp,' etc.)
- Remove Twitter mentions (@someone)
- Remove URLs (like https://..., http://....., www....)
- Remove stop words.

3.1 Removing Unicode Characters.

Unicode is an encoding standard followed all over the world for the consistent encoding, representation, and handling of text data. (Wikipedia)

Upon observing the training data we found that the samples contained Unicode characters like '\u002c', '\u2019t', etc. Since these characters do not contribute to the sentiment prediction we can remove them.

3.2 Lower casing.

The lower casing of the samples is one so that the model does not treat words like 'Good' and 'good' differently because both the words have equal importance in defining the sentiment.

3.3 HTML Decoding.

We will also try to decode the HTML characters like '&amp', '&lt', etc that may be present in the samples to their text form so that it reduces the noise in the data.

3.4 Removing @mentions.

In many tweets, users mention other users using the '@' character, although these can be used to analyze the tweets for certain information but it does not contribute to the sentiment of the tweet so we can safely remove it.

3.5 Removing URLs.

People may include URLs in their tweets to add some useful information but again this inclusion of these URLs will not contribute to our model because URL can help in analyzing sentiment in a piece of text.

3.6 Removing Stop Words.

Stop words are words that are commonly used like ' a', 'the', 'is', 'are', etc. We remove stop words because these are low information words that may be common in all classes and will add to the noise, removing these words will allow the model to focus on more important words.

4 Data Visualization

After cleaning the data using the above-mentioned steps, we will add a character length of the cleaned tweets as a new feature in our training data to get a better insight into the data.

4.1 Characters Length in Cleaned Tweets

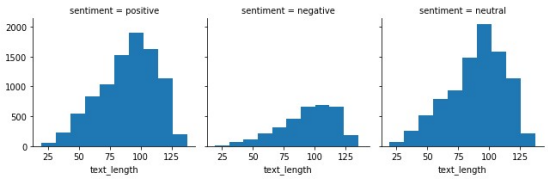
We will calculate the character length of the cleaned tweets and take a look at how it varies among the classes.

count	21465.000000
mean	67.402888
std	43.282333
min	0.000000
25%	53.000000
50%	68.000000
75%	81.000000
max	3652.000000

We have a min length of zero and max length of 3625 characters. As Twitter imposes a character length limit of 280, we can safely remove the ones more than that as they add to the noise.

count	21458.000000
mean	66.847003
std	19.411252
min	8.000000
25%	53.000000
50%	68.000000
75%	81.000000
max	132.000000

After removing the outlier tweets we get final cleaned 21458 samples with a min of 8 characters and a maximum of 132 characters.



This is the distribution of the tweets with respect to the text length among all the classes.

5 Setting Up a baseline.

Before actually beginning to create a model we should create a baseline so that we can know how better our models perform. We can use the python library Textblob for this task. In addition to providing great features like part-of-speech tagging, noun phrase extraction, classification, etc it also provides a basic sentiment analysis feature that we can use for our baseline. Then senitment() function of the TextBlob library returns a polarity in range[-1, 1] where -1 being most negative and 1 being most positive. We will check using TextBlob to predict three classes 'neutral', 'positive' and 'negative' classes on the test data and submitting on Kaggle to get mean F-score.

We got a mean F-score of 0.55210 on the test data using TextBlob's inbuilt sentiment analyzer function. This will be our baseline moving forward.

6 Traditional Machine Learning Algorithms

In present times deep learning has performed a lot better in almost all machine learning tasks outperforming the statistical machine learning approaches like Logistic Regression, Decision Trees, Random Forest Classifiers, etc. But we will try to implement and briefly describe these algorithms to get a better understanding of them.

TF-IDF Features

We will be using TF-IDF vectorizer to vectorize the sample before feeding it to the classifiers.

TF-IDF stands for Term Frequency-Inverse Document Frequency, and the TF-IDF weight is a weight often used in information retrieval and text mining. This weight is a statistical measure used to evaluate how important a word is to a document in a collection or corpus. The importance increases proportionally to the number of times a word appears in the document but is offset by the frequency of the word in the corpus.

6.1 Decision Tree Classifier.

Decision Tree is a very popular supervised learning technique used both for classification and regression tasks. But they tend to overfit on the data. Although they are not the best choice for the particular task of sentiment analysis due to the huge feature space involved but we can try it out and see how it performs.

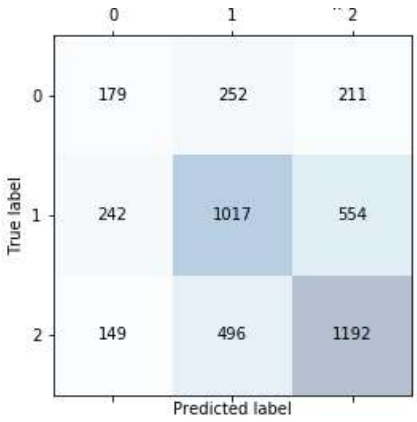
```
Accuracy 0.5563839701770736
F1-score 0.5595117881186039
              precision    recall  f1-score   support

    negative       0.31      0.28      0.30       642
     neutral       0.58      0.56      0.57      1813
    positive       0.61      0.65      0.63      1837

    accuracy                           0.56      4292
   macro avg       0.50      0.50      0.50      4292
weighted avg       0.55      0.56      0.55      4292
```

We get an accuracy of 0.55 and F1 score of 0.55 on validation data also they are near about the performance we got from TextBlob.

0 – Negative Class
1 – Neutral Class
2 – Positive Class



(Confusion Matrix for Decision Tree Classifier)

6.2 Random Forest Classifier.

Random Forest is an ensemble of multiple decision trees. Each decision tree predicts a class and then the class predicted by most decision tree becomes the final prediction.

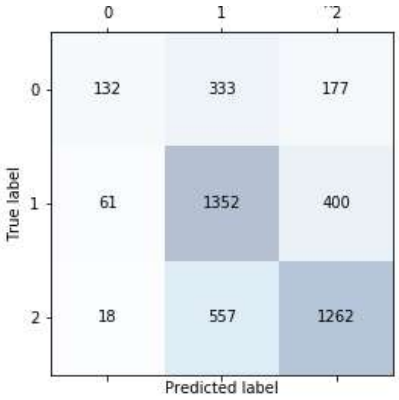
The main intuition behind Random Forest is trying to predict on multiple decision trees working uncorrelated results is a better performance overall.

```
Accuracy 0.6397949673811743
F1-score 0.6577412663615481
              precision    recall  f1-score   support

    negative       0.63      0.21      0.31       642
     neutral       0.60      0.75      0.67      1813
    positive       0.69      0.69      0.69      1837

    accuracy                           0.64      4292
   macro avg       0.64      0.55      0.55      4292
weighted avg       0.64      0.64      0.62      4292
```

The performance improved significantly with an accuracy of 0.63 and a F1 score of 0.65 on validation data.



(Confusion Matrix for Random Forest Classifier)

6.3 Logistic Regression Classifier.

Logistic regression is a classification algorithm used to assign observations to a discrete set of classes. Some of the examples of classification problems are Email spam or not spam, Online transactions Fraud or not Fraud, Tumor Malignant or Benign. Logistic regression transforms its output using the logistic sigmoid function to return a probability value.

It can be of two types:

- Binary (eg. Tumor Malignant or Benign)
- Multi-linear functions (eg. Multiple species of dogs.)

Below is an example of Logistic Regression Equation:

$$y = e^{(b0 + b1*x)} / (1 + e^{(b0 + b1*x)})$$

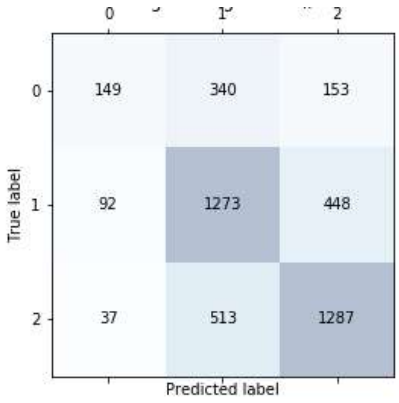
Where y is the predicted output, b0 is the bias or Intercept term and b1 is the coefficient for the single input value (x). Each column in your input data has an associated b coefficient (a constant real value) that must be learned from your training data. Let's see how it performs on our data.

```
Accuracy 0.6311742777260019
F1-score 0.6451126185619539
              precision    recall  f1-score   support

    negative       0.54      0.23      0.32       642
     neutral       0.60      0.70      0.65      1813
    positive       0.68      0.70      0.69      1837

    accuracy                           0.63      4292
   macro avg       0.61      0.54      0.55      4292
weighted avg       0.62      0.63      0.62      4292
```

We got an accuracy of 0.63 and F1 score of 0.64 which is slightly less than that of the Random Forest Classifier.



(Confusion Matrix for Logistic Regression Classifier)

We observe from the above confusion matrix that many of the negative samples are classified as neutral and positive classes, this is happening because of the fewer samples in the negative class. We can see if we can improve this by using handling the class imbalance problem using Oversampling or Downsampling (In this project will be checking Oversampling because downsampling will decrease the overall training data which is already very less comparatively.)

7 Handling Data Imbalance

In this section, we will briefly go through some techniques to handle data imbalance and implement some of them on our data to get a better understanding.

Data imbalance is the condition when some classes have a very less number of samples compared to other classes. If such kind of data is fed to a model the model's prediction is inclined towards the class having most samples ignoring the classes with a lesser number of samples.

On a high level, there are two techniques used for handling data imbalance.

Oversampling

In oversampling we simply increase the samples in the minority class either by randomly duplicating samples or by creating and adding synthetic or generated samples in the minority class. Random Over Sampler and SMOTE (Synthetic Minority Over-Sampling Technique), we will discuss more about them when we implement them.

Downsampling

As the name itself suggests this technique uses downsampling of data i.e it lowers the samples in the majority class to match up with the number of samples in the minority class. This prevents the model from being overwhelmed by the majority class.

7.1 Random Over Sampler.

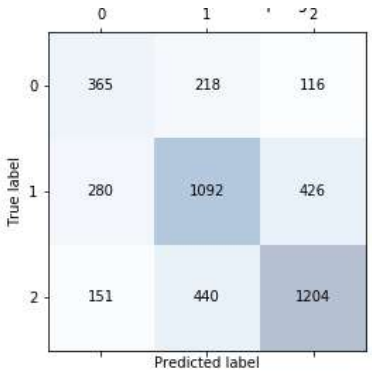
We will use the python library '**imbalanced-learn**' to implement RandomOverSampler. As the name implies Random Over Sampler is a method to duplicate random samples of the classes with a lesser number of samples to balance out with the majority classes. We used Logistic Regression with TF-IDF vectorizer in this case.

```
Accuracy 0.619990680335508
F1-score 0.618184541125921
              precision    recall  f1-score   support

    negative       0.46      0.52      0.49       699
     neutral       0.62      0.61      0.62      1798
    positive       0.69      0.67      0.68      1795

    accuracy                           0.62      4292
   macro avg       0.59      0.60      0.59      4292
weighted avg       0.62      0.62      0.62      4292
```

The accuracy got on the validation data using this technique is 0.62 and a F1 score of 0.62 on validation data which is slightly less than that of logistic regression.



(Confusion Matrix for Random Over Sampler with Logistic Regression.)

7.2 SMOTE (Synthetic Minority Over-Sampling Technique).

SMOTE is an oversampling approach in which the minority class is over-sampled by creating "synthetic" examples rather than by over-sampling using duplicates.

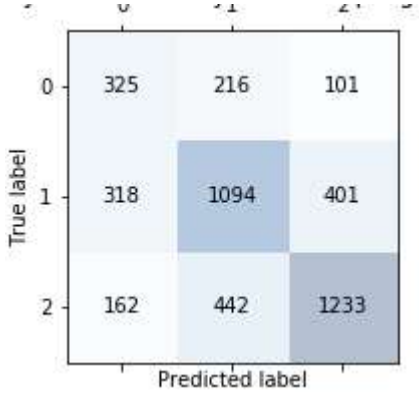
According to the original research paper (Chawla et al., 2002), "synthetic samples are generated in the following way: Take the difference between the feature vector (sample) under consideration and its nearest neighbor. Multiply this difference by a random number between 0 and 1 and add it to the feature vector under consideration. This causes the selection of a random point along the line segment between two specific features. This approach effectively forces the decision region of the minority class to become more general.
We used Logistic Regression with TF-IDF vectorizer in this case as well like RandomOverSampler to compare them consistently.

```
Accuracy 0.6178937558247903
F1-score 0.6138588459662743
              precision    recall  f1-score   support

    negative       0.40      0.51      0.45       642
     neutral       0.62      0.60      0.61      1813
    positive       0.71      0.67      0.69      1837

    accuracy                           0.62      4292
   macro avg       0.58      0.59      0.58      4292
weighted avg       0.63      0.62      0.62      4292
```

The accuracy got using SMOTE was 0.62 and the F1 score was 0.61 on the validation data. These values are very similar to that of Random Over Sampler.



(Confusion Matrix for SMOTE with Logistic Regression.)

8 Deep Learning Models.

After exploring some of the traditional machine learning algorithms like Decision Tree, Random Forest, and Logistic Regression, we will now try to model our data using neural networks. We will first use a vanilla neural network with single hidden layer then we will try out LSTMs (Long Short Term Memory) Networks which are more suited for tasks involving analysis of long term dependency data like textual data and in the last we will try to build a model using a combination of CNNs (Convolutional Neural Network) and LSTMs as in recent time CNNs have provided good results in NLP tasks such as sentiment analysis.

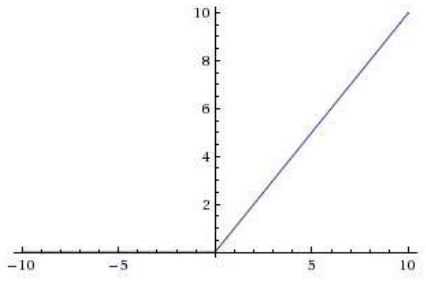
8.1 Simple Neural Network.

Our first deep learning model will consist of one input layer with 40000 nodes in which we will feed the samples after vectorizing them using the TF-IDF vectorizer, one hidden layer with 128 nodes and the output layer has 3 nodes representing the 3 classes we are predicting.

For the activation function in the hidden layer we be using 'RELU', it stands for the Rectified Linear Unit.

RELU

The Rectified Linear Unit or RELU is the most commonly used activation function in deep learning models. The function returns 0 if it receives any negative input, but for any positive value x it returns that value back. So it can be written as f(x)=max(0,x) .



For the activation function in the output layer, we are using softmax because we are essentially predicting the probabilities of the three classes.

Softmax

It is a function that basically normalizes the input into a vector of values that follows a probability distribution which sums up to 1.

$$\text{softmax}(y)_i = \frac{\exp(y_i)}{\sum_j \exp(y_j)}$$

And finally, the loss function we are using is the "categorical crossentropy".

Categorical CrossEntropy

This loss function is used in multi-class classification. It is most of the time used with softmax activation function as we are prediction probability of C classes.

$$-\frac{1}{N}\sum_{i=1}^{N}\log p_{model}\left[y_i \in C_{y_i}\right]$$

The mathematical formula for categorical cross-entropy.

We used *Adam* optimizer in this model for backpropagation.

Adam optimization is a stochastic gradient descent method that is based on an adaptive estimation of first-order and second-order moments.

According to (Kingma et al., 2014), the method is "computationally efficient, has little memory requirement, invariant to the diagonal rescaling of gradients, and is well suited for problems that are large in terms of data/parameters".

We also added a dropout of 20% to the hidden layer so that the model better generalizes on the data.

Dropout

The intuition behind dropout is randomly deactivating nodes in the layer so that the network does not overfits on the data.

According to the research paper "Improving neural networks by preventing co-adaptation of feature detectors" by Hinton et al. (2012), "*A good way to reduce the error on the test set is to average the predictions produced by a very large number of different networks. The standard way to do this is to train many separate networks and then to apply each of these networks to the test data, but this is computationally expensive during both training and testing. Random dropout makes it possible to train a huge number of different networks in a reasonable time.*"

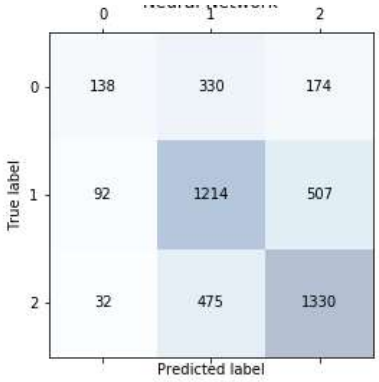
Let's see how this model performs.

```
Accuracy 0.624883504193849
F1-score 0.6405856214487828
              precision    recall  f1-score   support

           0       0.53      0.21      0.31       642
           1       0.60      0.67      0.63      1813
           2       0.66      0.72      0.69      1837

   micro avg       0.62      0.62      0.62      4292
   macro avg       0.60      0.54      0.54      4292
weighted avg       0.62      0.62      0.61      4292
 samples avg       0.62      0.62      0.62      4292
```

We get an accuracy of 0.62 and F1 score of 0.64 on the validation data which is an improvement over the SMOTE model.



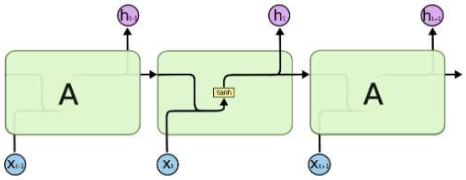
(Confusion Matrix for Simple Neural Network Classifier)

8.2 Long Short Term Memory Network.

Long Short Term Memory networks – usually just called "LSTMs" – are a special kind of RNN, capable of learning long-term dependencies. They were introduced by Hochreiter et al.,1997, and were refined and popularized by many people in the following work. They work tremendously well on a large variety of problems and are now widely used.

LSTMs are explicitly designed to avoid the long-term dependency problem. Remembering information for long periods of time is practically their default behavior, not something they struggle to learn!

All recurrent neural networks have the form of a chain of repeating modules of neural network. In standard RNNs, this repeating module will have a very simple structure, such as a single tanh layer.



A simple LSTM representation.

For this model, we use an Embedding layer with 40000 features and an LSTM layer with 128 nodes.

The Embedding Layer

Keras offers an Embedding layer that can be used for neural networks on text data.

It requires that the input data be integer encoded so that each word is represented by a unique integer. This data preparation step can be performed using the Tokenizer API also provided with Keras.

The Embedding layer is initialized with random weights and will learn an embedding for all of the words in the training dataset.

It is a flexible layer that can be used in a variety of ways, such as:

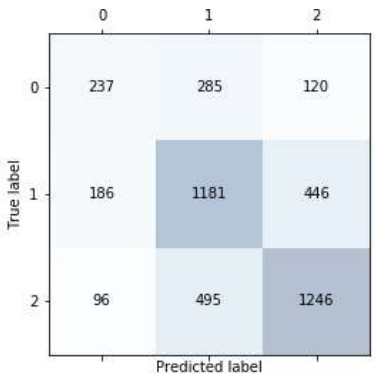
- It can be used alone to learn a word embedding that can be saved and used in another model later.
- It can be used as part of a deep learning model where the embedding is learned along with the model itself.
- It can be used to load a pre-trained word embedding model, a type of transfer learning.

```
Accuracy 0.6206896551724138
F1-score 0.6177379746056479
              precision    recall  f1-score   support

           0       0.46      0.37      0.41       642
           1       0.60      0.65      0.63      1813
           2       0.69      0.68      0.68      1837

    accuracy                           0.62      4292
   macro avg       0.58      0.57      0.57      4292
weighted avg       0.62      0.62      0.62      4292
```

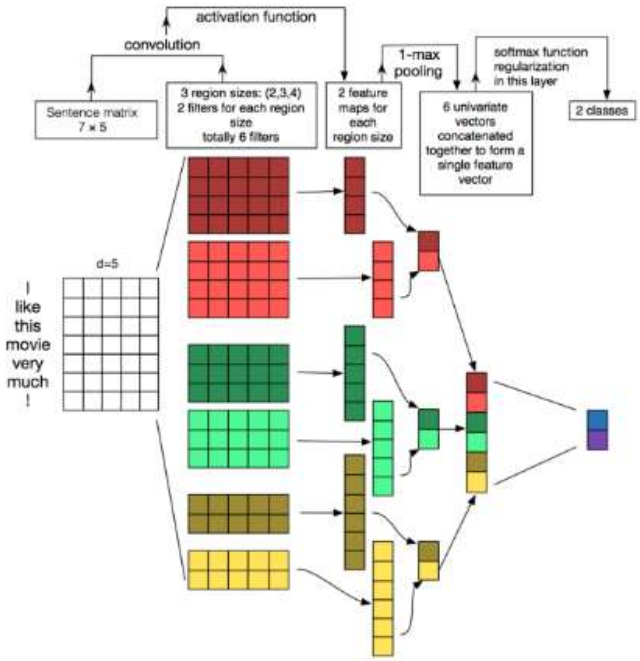
We got an accuracy of 0.62 and F1 score of .62 on the validation data which is slightly lower our neural network model.



(Confusion Matrix for LSTM Model)

8.3 CNN – LSTM Combination.

CNNs are mainly used in the world of Computer Vision but in many papers [*Kim Y. Convolutional Neural Networks for Sentence Classification. 2014*] we get to know that they show promising results in NLP also. The very intuition behind CNN is to extract patterns in a small group of data herein NLP we can think this like small phrases ("I like this"), ("It's bad"), etc. We can stack multiple CNN layers each with different kernel sizes (2, 3, 4, 5) referring to 2 words phrases, 3-word phrases, and so on. The following figure very clearly depicts the intuition behind the use of CNN in text classification. We mostly use the Convolutional layer to extract local features.



Zhang, Y., & Wallace, B. (2015) 'A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification

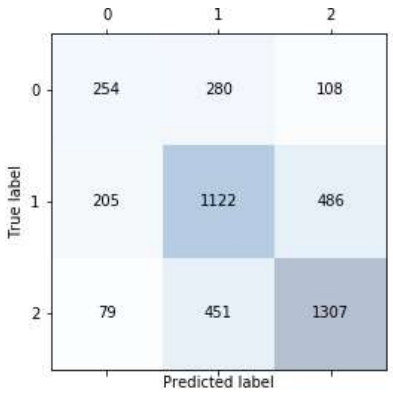
A combination of CNN and LSTM works well because CNN helps to extract local features and LSTMs are good for Long term dependency of data.
For our model, we are using 4 Convolutional Layer each with 128 filters and a kernel size of [2,3,4,5]. We also add a max pool layer with pool size 2 after each convolutional layer and then we add a LSTM layer with 264 units with a dropout of value 0.2 and also a recurrent dropout of 0.2 to lower overfitting and make the model better generalize on the data.

```
Accuracy 0.625116495806151
F1-score 0.6222661865588629
              precision    recall  f1-score   support

           0       0.47      0.40      0.43       642
           1       0.61      0.62      0.61      1813
           2       0.69      0.71      0.70      1837

    accuracy                           0.63      4292
   macro avg       0.59      0.58      0.58      4292
weighted avg       0.62      0.63      0.62      4292
```

We got and accuracy of 0.62 and F1 score of 0.62 on the validation data.



(Confusion Matrix for CNN-LSTM Model)

9 Model Comparison on Test Data

We will now compare the performance of the models on test data. The evaluation metric here will be the mean F score as used in the associated Kaggle Competition.

On comparison, we can see that the CNN-LSTM model performs the best and we improved from our baseline of 0.55210 to 0.66859. Although there is much, we can do to improve this performance.

Model	Mean F-Score
TextBlob (Baseline)	0.55210
LogisticRegression	0.65076
Logistic Regression with Random Over Sampler	0.65145
Logistic Regression with SMOTE	0.65470
Simple Dense Neural Network with TFIDF Vectors	0.63436
LSTM	0.64080
CNN-LSTM	0.66859

10 Future Improvements.

In the last section, we saw the CNN-LSTM model performs the best but still it is not very good. We will now discuss how we can use some approaches to improve performance.

Pre-Trained Word Embeddings.
Pre-Trained Word Embeddings like Word2Vec (Mikolov et al., 2013) and GloVe (Pennington et al., 2014) make the computations fast and improve the overall performance of the model.

Transformers
Transformer is an architecture which uses Encoder and Decoder to solve sequence to sequence tasks with handling long term dependencies (better than LSTMs). It was introduced in the famous paper "Attention is all you need" (Vaswani et al., 2017) and has been very successful in giving state of the art performances in various NLP tasks. It forms the base of the famous architecture made by Google called BERT (Bidirectional Encoder Representations from Transformers) which they claim improved their "Google Search" feature a lot.
Quoting from the paper (Vaswani et al., 2017)

"The Transformer is the first transduction model relying entirely on self-attention to compute representations of its input and output without using sequence-aligned RNNs or convolution."

BERT itself is the state-of-the-art model for NLP tasks such as summarization, sentiment analysis we can try this also on our training data and see it performs.

11 Tweelyzer (Web app)

Tweelyzer is a simple yet very effective web app that is built by embedding the best performing model described above and using the python framework called ‘flask’.

The user is required to enter keywords in the app and it fetches the latest 100 tweets using the ‘tweepy’ library and performs sentiment analysis on them. After is displays the result in a pie chart distributed as ‘positive’, ‘negative’, or ‘neutral’.

12 Conclusion.

In this project, we explored a lot of ML concepts including some statistical algorithms like Logistic Regression, Decision Trees, Random Forest, etc. We then used deep learning concepts like a basic dense neural network, LSTMs and CNNs to model our data. In the process we also covered important ML project building blocks like Data preprocessing, visualizing data, handling class imbalance, etc.
Learning and actually implementing these concepts was a very exciting journey. This being my first complete end-to-end ML project helped me learn a lot and increased my confidence in the domain. Also, building a real-life application that users can use and benefit based on the ML model is a very great feeling.
Thanks to the IITK faculty for building a very great internship program. It helped us gain a lot of knowledge in this domain in very little time.

References

Ian Goodfellow and Yoshua Bengio and Aaron Courville, 2016. The Deep Learning Book.

Ye Zhang, Byron C. Wallace. 2017. A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification

Mathieu Cliche. 2017. BB twtr at SemEval-2017 Task 4: Twitter Sentiment Analysis with CNNs and LSTMs

N. V. Chawla, K. W. Bowyer, L. O. Hall, W. P. Kegelmeyer . 2002. SMOTE: Synthetic Minority Over-sampling Technique.

Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, Ruslan R. Salakhutdinov. 2012. Improving neural networks by preventing co-adaptation of feature detectors.

Diederik P. Kingma, Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization.

Sepp Hochreiter, Jurgen Schmidhuber. 1997. LONG SHORT-TERM MEMORY

Ye Zhang, Byron Wallace. 2015. A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification

Yann LeCun Leon Bottou Yoshua Bengio and Patrick Haffner. 1998. GradientBased Learning Applied to Document Recognition

Tomas Mikolov, Kai Chen, Greg Corrado, Jeffrey Dean 2013. Efficient Estimation of Word Representations in Vector Space

Jeffrey Pennington, Richard Socher, Christopher Manning 2014. GloVe: Global Vectors for Word Representation

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, Illia Polosukhin 2017. Attention Is All You Need

Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova 2018. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding