The motivation behind the project

1. The detection of cheating in academic communities is significant to properly address the contribution of each researcher so working towards a finer approach is required
2. Similarly, this could be used in public or private online coding contests whether done in coding interviews or in official coding training contests to detect the cheating of applicants or contestants
3. This facilitates solving issues related to cheating in academic, work, and open source environments. Also, it can help detect the authors of malware software all over the world.

Nature of the project - Research cum development 1.

An end-to-end Neural Network Framework for Text Clustering -

Analysis - In this paper, we have seen some models using LSTM and word vectors with a new loss function. Though the main thing we got

from the paper is not the proposed architecture why not to use clustering options because of high dimensions around 180-d and also no way to use temporal and contextual knowledge of the word vectors

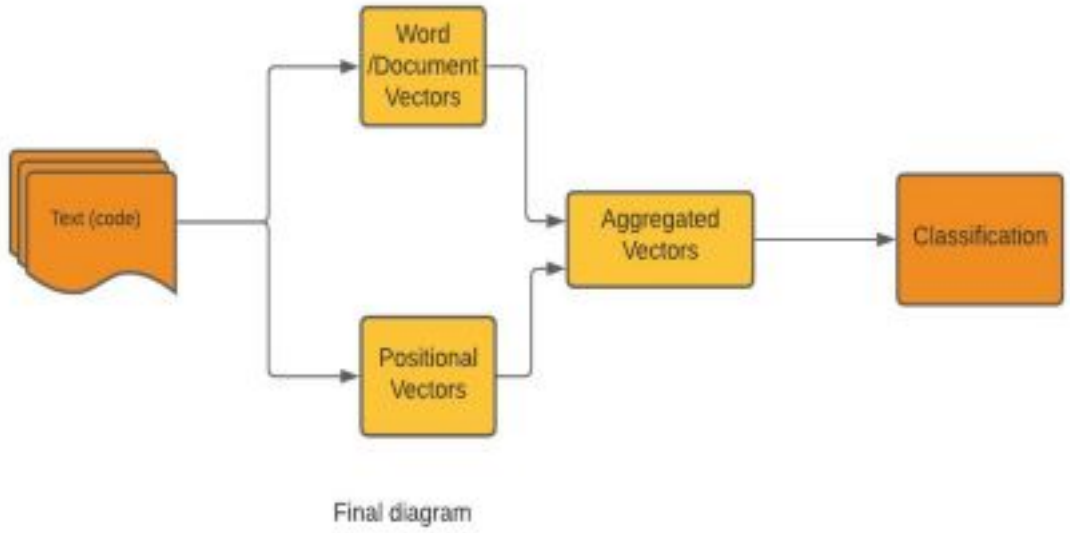
Summary - This paper shows us how deep learning could be used on the other side of k means and DBSCAN for modeling the task with superior accuracy and cluster affinity.

2. Attention Is All You Need

Analysis - In this paper, we have seen some Novel models like Transformers using only self-attention, not any word vectors or conventional approach. Though the main thing we got from the paper is not the proposed architecture of why and how to use Positional embeddings that adds a lot to the word, document vectors and also adds contextual knowledge.

Summary - the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. We have only accessed the positional embedding part of the paper

The overall design of the project



Final diagram

The above diagram is a clear depiction of our final project that we are submitting. The pipeline goes in the way is

The **text(code)** is first tokenized and then convert these tokens into the **Word/document vectors** and **Positional vectors**, These vectors were averaged and aggregated into one set of vectors thus adding positional knowledge to our word vectors that do not include it which gave us very good results that approve our hypothesis that word vectors are not very good features for the particular task and after adding positional embeddings to it we have got some great results on it.

Features around our project

1. In our dataset, we selected 1,000 users and collected 75 source codes from each one. So, the total number of source codes is 75,000. All collected source codes are correct, bug-free, compile-ready, and written using the C++ programming language

using different versions. For each user, all collected source codes are from unique problems.

2. We have developed the models on **word2vec** and **doc2vec** to get vectors of text(code). The weights can be provided if asked

3. We have developed the Positional embeddings of our Text(code) from our custom positional embedding class.

4. Next comes the classification task here we have trained 4 models a. KNN - Simple model but plagued with high bias very good for baseline results.
   b. Random forest - A complex model but very resistant to overfitting but not very good for the high number of classes c. SVM - Great model for Non-linear classification with kernel transformations
   d. NN - Though we have just experimented with it on the classification of data with and without positional embeddings

*More are provided in the results section

Methodology

Our methodology is very simple as our way of processing the data is

described by doing

1. **Text preparation and cleaning** - We have not performed much of the cleaning as we have to save the writing structure of the user by not removing the stop words that we do if we have another type of data but we consider not to alter the form of the text(code)

2. **Tokenizing and Vocab generation** - This part is simple as we have just segregated the text(code) file into the set of unique words in the file; these unique words are then aggregated and create a general vocab over all of the data.

3. **Word and Doc vectors** - After the vocab generation we have used the word2vec(skip-gram) model and averaging these word2vec data embeddings. While we have also used a separate Doc2vec model (word2vec + specific text ids or Paragraph Ids)

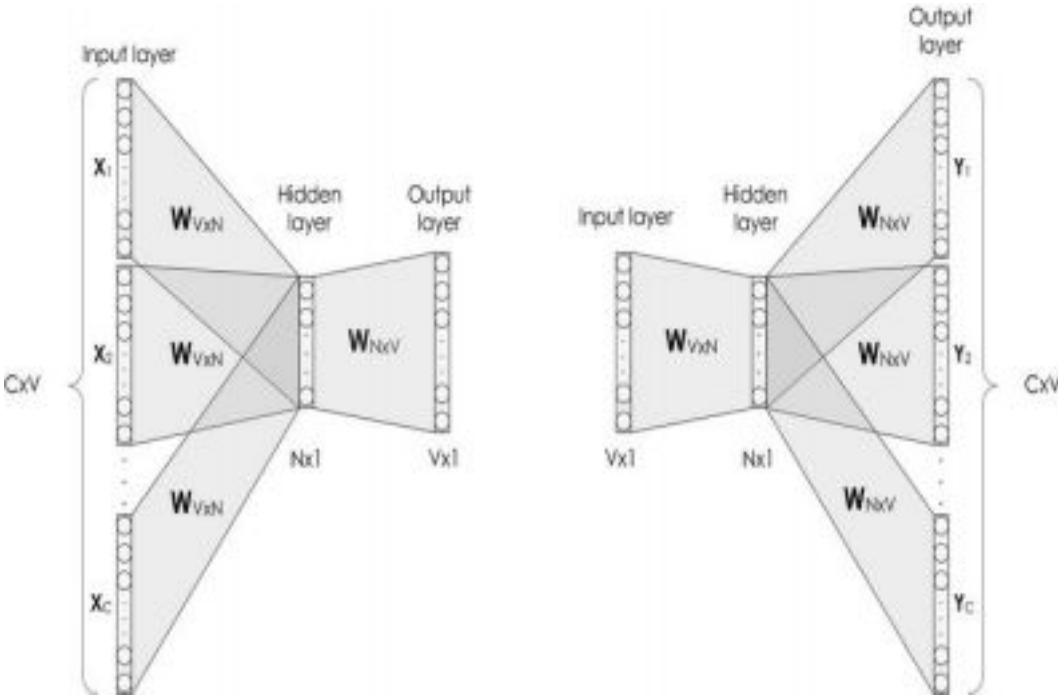
4. **Positional vectors** - This is the new addition that we have used in modeling our task. We have used this method to get the embedding of every unique word in each of the documents; these embeddings are then averaged and added with the doc2vec embeddings thus adding contextual and temporal features. This leads us with the final data of 180 length dimensions. Now finally

we do classification.

Algorithms on the Work

Here are some of the important Algorithms that we have used in the task

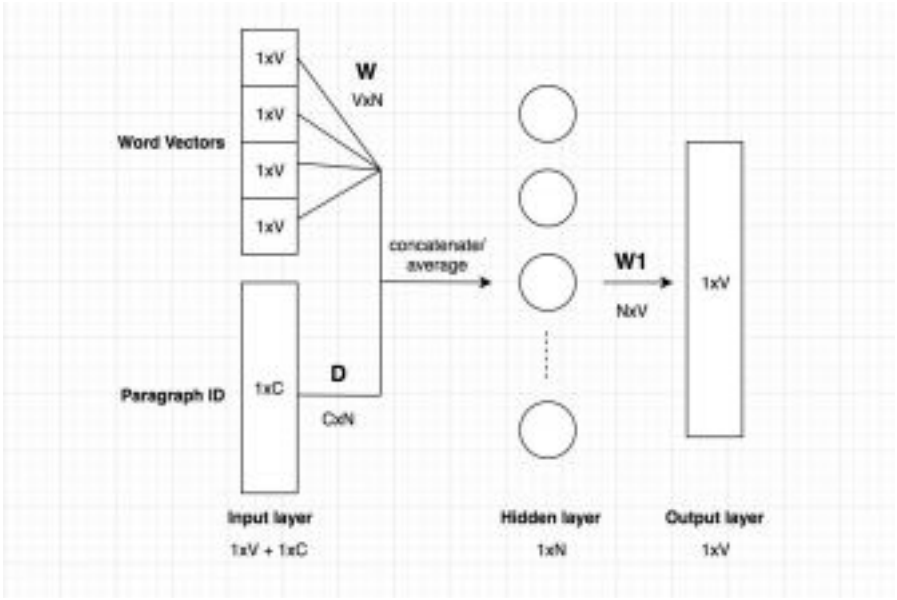
1. **Word2vec**:- To give you an insight let's first talk about it abstractly. Word2Vec uses a simple neural network with a single hidden layer to learn the weights. Different from most other machine learning models, we are not interested in the predictions this neural network could make. Instead, we only care about the weights of the hidden layer, since these weights are the word embeddings/vectors we are about to learn. It can be obtained using two methods: Skip Gram and Common Bag Of Words (CBOW)



This picture above shows the process of using a specific sequence of words to predict the next or target word. These words are then made as a one-hot vector after the one-hot vector we will pass these to the hidden layer which is then output to a softmax layer of 180 dimensions in length.

Skip Gram- For the Skip-Gram model, the task of the simple neural network is: Given an input word in a sentence, the network will predict how likely it is for each word in the vocabulary is that input word's nearby word.

Doc2vec- The role of positional embeddings is to supply information regarding the position of each token. This allows the attention layer to compute context-dependent results, that is, two tokens with the same value in the input sentence would get different representations.



The above diagram is based on the CBOW model, but instead of using

just nearby words to predict the word, we also added another feature vector, which is document-unique. So when training the word vectors W, the document vector D is trained as well, and at the end of the training, it holds a numeric representation of the document.

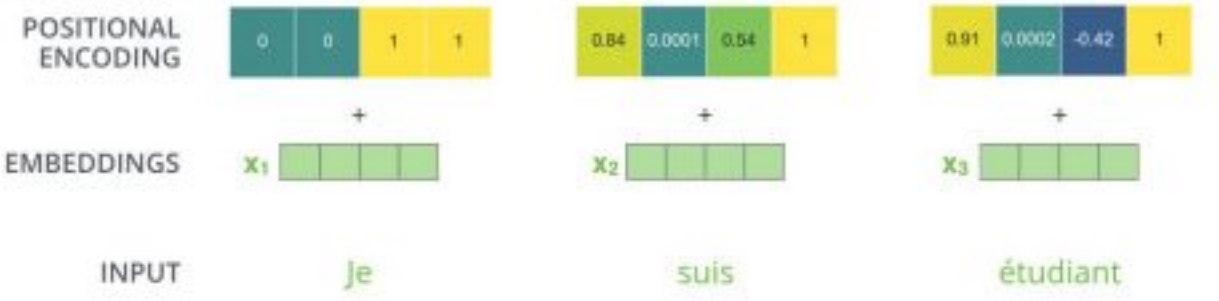
2. Positional vectors:-

The position and order of words are essential parts of any language. They define the grammar and thus the actual semantics of a sentence. Recurrent Neural Networks (RNNs) inherently take the order of words into account; They parse a sentence word by word in a sequential manner. This will integrate the words' order in the backbone of RNNs.

Positional Embeddings are introduced for recovering position information. In the paper, two versions of positional embeddings are mentioned, learned positional embeddings and sinusoidal and cosine positional embeddings respectively, and both are said to produce similar results. The role of positional embeddings is to supply information regarding the position of each token. This allows the classifier to compute context-dependent results, that is, two tokens with the same value in the input sentence would get different representations.

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$

$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$



Although the words used are absolutely the same, the meanings are opposite, information of the order is required to distinguish different meanings. sinusoidal positional embeddings generate embeddings using **sin** and **cos** functions. By using the equation shown above, the author hypothesized it would allow the model to learn the relative positions.

3. **SVM:-** SVMs are inherently two-class classifiers. The traditional way to do multiclass classification. In particular, the most common technique in practice has been to build one-versus-rest classifiers (commonly referred to as ``one-versus-all'' or OVA classification) and to choose the class which classifies the test datum with the greatest margin.
   Another strategy is to build a set of one-versus-one classifiers and

to choose the class that is selected by the most classifiers.

While this involves building classifiers, the time for training classifiers may decrease, since the training data set for each classifier is much smaller. We have used a non - linear method of kernelising the data with the use of the RBF kernel.

What RBF kernel SVM does is to create non-linear combinations of your features to uplift your samples onto a higher dimensional feature space where you can use a linear decision boundary to separate your classes

$$\begin{aligned} K\left(x^{(i)}, x^{(j)}\right) &= \phi(x^{(i)})^T \phi(x^{(j)}) \\ &= \exp\left(-\gamma \left\| x^{(i)} - x^{(j)} \right\|^2\right), \qquad \gamma > 0 \end{aligned}$$

In this equation, **gamma** specifies how much a single training point has on the other data points around it.

$$\begin{aligned} \exp\left(-\frac{1}{2}\|\mathbf{x} - \mathbf{x}'\|^2\right) &= \sum_{j=0}^{\infty} \frac{(\mathbf{x}^\top \mathbf{x}')^j}{j!} \exp\left(-\frac{1}{2}\|\mathbf{x}\|^2\right) \exp\left(-\frac{1}{2}\|\mathbf{x}'\|^2\right) \\ &= \sum_{j=0}^{\infty} \sum_{\sum n_i = j} \exp\left(-\frac{1}{2}\|\mathbf{x}\|^2\right) \frac{x_1^{n_1} \cdots x_k^{n_k}}{\sqrt{n_1! \cdots n_k!}} \exp\left(-\frac{1}{2}\|\mathbf{x}'\|^2\right) \frac{x'^{n_1}_1 \cdots x'^{n_k}_k}{\sqrt{n_1! \cdots n_k!}} \end{aligned}$$

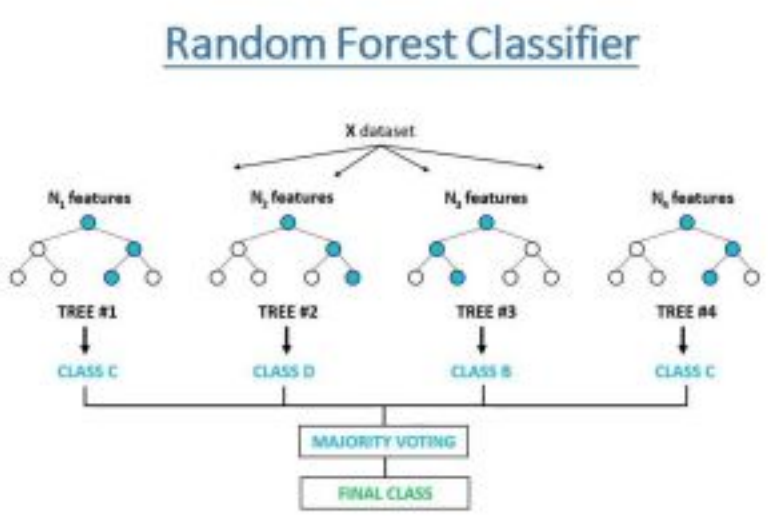
4. Random Forest:-

Random forest algorithm intuition can be divided into two stages. In the first stage, we randomly select “k” features out of the total m features and build the random forest. In the first stage, we proceed as follows:-

1. Randomly select k features from a total of m features where k < m.
2. Among the k features, calculate the node d using the best split point.
3. Split the node into daughter nodes using the best split. 4. Repeat 1 to 3 steps until the number of nodes has been reached. 5. Build a forest by repeating steps 1 to 4 several times to create several trees.

In the predicted stage, we make predictions using the trained random forest algorithm.

1. We take the test features and use each randomly created decision tree to predict the outcome and store the predicted outcome. 2. Then, we calculate the votes for each predicted target. 3. Finally, we consider the high voted predicted target as the final prediction from the random forest algorithm.



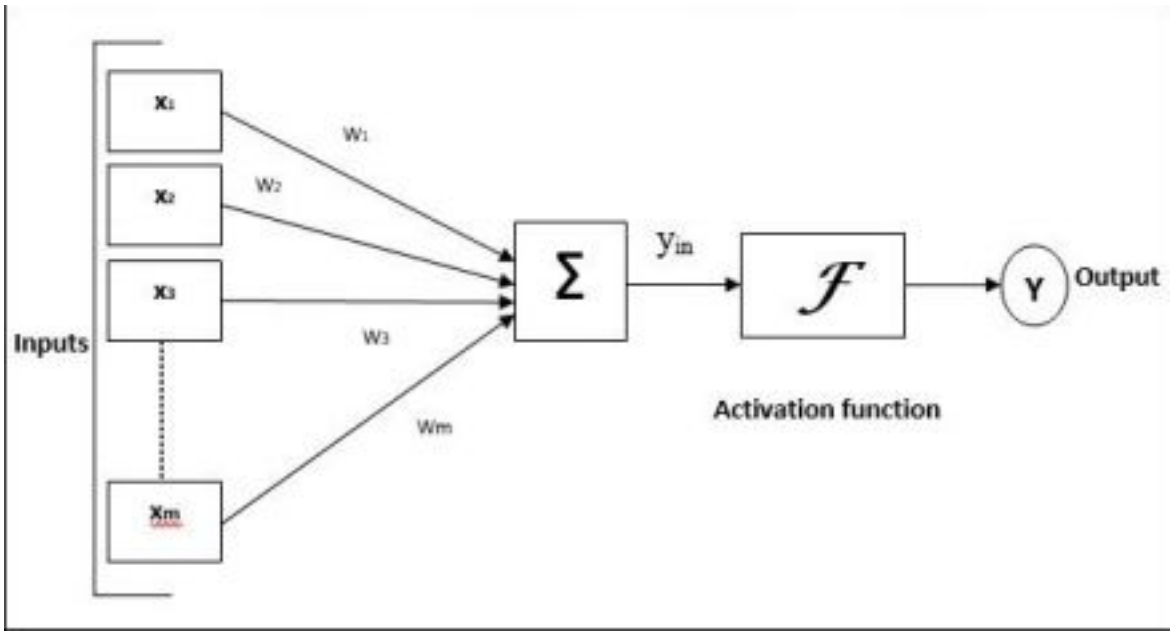
As the internal go random forest is a collection of trees that distribute the data for classification using information gain. Information gain is made up of entropy. Entropy is nothing but the **measure of disorder.** (You can think of it as a measure of purity as well. The more the information gained the better the feature column classifies the data.

$$E(S) = \sum_{i=1}^{c} - p_i \log_2 p_i$$

$$IG(Y, X) = E(Y) - E(Y|X)$$

And last but not least we have done some experiments on ANN or feed-forward Neural Network but though we were told by our supervisor to not do much experimentation on it as it is of deep learning paradigm it is very good for fast experimenting and showing

results in the further section.



The problem that we have faced in every classification problem is that a very high number of classes(1000) classes with around 75 samples per class because The number of categories a classifier could classify with good precision/recall, is how distinct each category is? (More on it in next section)

Results:

KNN (K - Nearest Neighbor)

1. Accuracy - 66%
2. F1 score - 67%

Random Forest

1. Accuracy - 52%

2. F1 score - 53%

SVM (Support Vector Machine)

1. Accuracy - 81.8%
2. F1 score - 82%

ANN (Feedforward Network)

With our New data of Doc2vec + Positional embeddings

Accuracy - 76.8%

With conventional Doc2vec

Accuracy - 35%

Discussion on KNN results

As we have said our KNN model with a hyperparameter of using 10 neighbors around every point to classify itself in a particular section with such a high number of classes we got a preliminary good accuracy of 66% in classifying that this type of **text(code)** that a particular participant had written in a problem that is solved by people before him is copied or might be copied with a score of 66% sufficiency to tell us that the code written by us might be modified and copied from an earlier participant. One more point to be discussed is that KNN is a very naive model with high bias and low variance that is why we have used another method of F1 score. This not only gives us a Precision and recall

of the whole model but with every class too

Precision-Recall

accuracy 0.66 75000

macro avg F1 0.72 0.66 0.67 75000

weighted avg F1 0.72 0.66 0.67 75000

This shows us an interesting conclusion that the data might be in a cluster form not very close but a packet loose points of every class

Discussion on Random Forest results

Surprisingly we have seen very different results on using random forests. We have thought that after KNN gave us decent results on the problem we were very hopeful on the random forest results but we have got only 52% accuracy on a particular parameter set though we can think that this might increase but this is a very long process of finding a good parameter set. The reason behind the results might be because of the very nature of our document embedding as the data is more clustered the more it will be difficult to classify it with decision trees and entropy.

Precision-Recall

accuracy 0.52 75000

macro avg 0.71 0.52 0.53 75000

weighted avg 0.71 0.52 0.53 75000

Accuracy : 0.5249066666666666

Discussion on SVM results

The results from SVM are very good as this gave us accuracy and an f1 score of 82%. These results also gave us a reason to support our hypothesis that it is difficult to classify data with some linear models and random forest trees as the decision boundaries are very difficult to map in nature and somewhat the orientation of each data point of 180 dimensions in the cartesian plane is not very good as in the SVM we have mapped this data in some other dimension using RBF kernel after this conversion the model does a very good task of classifying the data.

Precision-Recall

accuracy 0.82 75000

macro avg 0.85 0.82 0.83 75000

weighted avg 0.85 0.82 0.83 75000

Discussion on ANN experiments

After these results from SVM, we have thought that we should try ANN for the classification task as the Feedforward network could formulate the high dimension embeddings into the model for prediction very easily so we trained only a deep model with one hidden layer. We have also used Dropout with a degree of 50% percent neuron being turned off during backpropagation this not only reduces overfitting but training time too. The dimension of the middle layer is 600 and the outer layer is 1000(number of classes)

Layer (type) Output Shape Param #

==========

Input_dense (Dense) (None, 300) 54300

---

dropout (Dropout) (None, 300) 0

---

dense_1 (Dense) (None, 600) 180600

---

dropout_1 (Dropout) (None, 600) 0

Output (Dense) (None, 1000) 601000

==========

Total params: 835,900

We have trained the model on 5 fold(Stratified). The results that we achieve is an average of 76% accuracy which is better than Random forest and KNN in the cartesian plane. This also beholds our idea that the data has very complex classification planes between the data classes. The training time on GPU is around 3 minutes and 20 minutes on CPU.

Now because of very low training time, we have tested it on naively trained document vectors and it produces an accuracy of 34%.

Work Distribution

Mostly the work on the project is associated with the whole group but the work on word2vec and doc2vec are mostly associated with Aditya and Aman.

While the positional encoding part is done by me in a broader perspective. In the modeling part, the work is mostly distributed in KNN, Random forest to Aditya and Aman while the work on SVM is mostly done by me.

Future work

There is a lot that can be done with this topic in the latter half as more concepts of attention with most commonly Transformers and self-attention could play an important role in magnifying the classification accuracy. Also, work on some advanced Deep learning model that optimizes on a novel clustering loss could also be the way

this project could proceed. In the context of classical Machine learning, we can find good parameters of the models used in this task. In the end, more work on the Self-attention of transformers might be a good direction to extend our project into.

Conclusion

In the final part of our project, I would like to finish on the note that we have got much support from our supervisor on the particular task and our evaluation panel in improving our thinking by providing valuable insights on how we can increase the effectiveness of our project. We as a team have tried our best to perform well on the project either in any part of the task. In the context of our project, there is much improvement that could be done described above. If we get the chance we can improve our project and amplify our effectiveness to the real world if the constraints of using advanced techniques of the machine and deep learning are relieved.