

# **Automated Essay Grading Using Machine Learning**

**Submitted in partial fulfilment of the requirements**

**of the degree of**

**Bachelor of Engineering**

**by**

**Priyanka Kedar      36**

**Neha Mishra        47**

**Sarthak Gupta      26**

**Supervisor:**

**Prof. Uday Nayak**



**UNIVERSITY OF MUMBAI**

# **Automated Essay Grading Using Machine Learning**

**Submitted in partial fulfilment of the requirements**

**of the degree of**

**Bachelor of Engineering**

**by**

**Priyanka Kedar      36**

**Neha Mishra        47**

**Sarthak Gupta      26**

**Supervisor:**

**Prof. Uday Nayak**



**Department of Information Technology**

**Don Bosco Institute of Technology**

**Vidyavihar Station Road, Mumbai - 400070**

**2019-2020**

# DON BOSCO INSTITUTE OF TECHNOLOGY

Vidyavihar Station Road, Mumbai - 400070

Department of Information Technology

## CERTIFICATE

This is to certify that the project entitled “**Automated Essay Grading Using Machine Learning**” is a bonafide work of

<b>Sarthak Gupta</b>	<b>26</b>
<b>Priyanka Kedar</b>	<b>36</b>
<b>Neha Mishra</b>	<b>47</b>

submitted to the University of Mumbai in partial fulfilment of the requirement for the award of the degree of **Undergraduate** in **Bachelor of Information Technology**

Date:     /     /

(Prof. Uday Nayak)  
Supervisor

(Prof. Prasad P.)  
HOD, IT Department

(Dr. Prasanna Nambiar)  
Principal

# **DON BOSCO INSTITUTE OF TECHNOLOGY**

**Vidyavihar Station Road, Mumbai - 400070**

**Department of Information Technology**

## **Project Report Approval for B.E.**

This project report entitled “**Automated Essay Grading Using Machine Learning**” by **Priyanka Kedar, Neha Mishra, Sarthak Gupta** is approved for the degree of **Bachelor of Engineering in Information Technology**

**(Examiner's Name and Signature)**

1. \_\_\_\_\_

2. \_\_\_\_\_

**(Supervisor's Name and Signature)**

1. \_\_\_\_\_

**(Chairman)**

1. \_\_\_\_\_

**Date:**

**Place:**

# **DON BOSCO INSTITUTE OF TECHNOLOGY**

**Vidyavihar Station Road, Mumbai - 400070**

## **Department of Information Technology**

### **Declaration**

I declare that this written submission represents my ideas in my own words and where others' ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea / data / fact / source in my submission. I understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

(\_\_\_\_\_ )  
**(Priyanka Kedar 36)**

(\_\_\_\_\_ )  
**(Neha Mishra 47)**

(\_\_\_\_\_ )  
**(Sarthak Gupta 26)**

**Date:**

# Abstract

The project aims to build an automated essay scoring system using a data set of 16173 essays from kaggle.com. The essay set contained of 8 different types of essays based on different contexts. We extracted features such as total word count per essay, removed Stopwords, word Lemmatizer, etc from the training set essays. We used RNN for defining layers and XGBClassifier model to learn from these features and generate parameters for testing and validation. We used Bi-directional LSTM for better model performance. Further, we performed testing on dataset for 3 times each time dropping different parameters for better accuracy. The Grading style used 4 parameters i.e. worst, average, above average, excellent for grading essays based on clarity and coherence. Categorical encoders were used for encoding these categorical parameters into numeric values. Sequence matrix was used convert tokenized text of max length 50 into matrix. We get an array of four values which are nothing but probability scores for values from 0 to 3 and the highest value is considered as graded score for the particular essay.

**Keywords:** Essay Grading, Machine Learning

# Contents

1	Introduction	1
1.1	Problem Statement .....	1
1.2	Scope of the Project .....	2
1.3	Current Scenario .....	2
1.4	Need for the Proposed System .....	2
1.5	Summary of the results and task completed .....	2
2	Review of Literature	3
2.1	Summary of the investigation in the published papers .....	3
2.2	Comparison between the tools / methods / algorithms .....	5
3	Analysis and Design	6
3.1	Methodology / Procedure adopted .....	6
3.2	Analysis .....	6
3.2.1	Software / System Requirement Specification – IEEE format . . Attach as Appendix .....	8
3.3	System Architecture .....	9
4	Implementation	11
4.1	Setting up Implementation Environment	12
4.2	Implementation Plan for Sem – 8 .....	11
4.3	Implementation Of the System .....	12
4.4	Roadmap Of The System .....	14
4.5	Testing .....	16
4.4.1	Testing Methods .....	16
4.6	Coding Standards .....	17
5	Result and Analysis	22
5.1	Result .....	22
5.2	Output .....	22

6 Conclusion and Future Scope	25
6.1 Conclusion .....	25
6.2 Future Scope .....	25
 Appendix	 26
Installation Procedure - Development Software .....	26
 References	 27
 Technical Paper in IEEE Format	 28



# List of Figures

2.1 Comparison of different existing systems . . . . .	5
3.1 Agile Methodology . . . . .	7
3.2 System Architecture . . . . .	10
3.3 Snapshot of Dataset. . . . .	11
4.1 Gantt Chart of AEG . . . . .	11
4.1 Front Page of Jupyter Notebook. . . . .	14
4.2 Creating new python3 Notebook. . . . .	15
4.3 Changing Runtime of Notebook. . . . .	16
4.4 Loading Dataset in Notebook. . . . .	18
4.5 Workflow of Score Reporter . . . . .	19
4.6 Input essay. . . . .	19
4.7 Embedding matrix explanation. . . . .	22
4.8 Sequence Matrix Generation. . . . .	22
4.9 Gantt Chart of AEG . . . . .	23
4.10 Flow of the AEG System . . . . .	24
4.11 Roadmap of the System . . . . .	27
4.12 Snapshot of imports . . . . .	29
4.13 Keras Lstm layer . . . . .	31
4.14 Pre-processing. . . . .	37
4.15 Output of Pre-processing . . . . .	37
4.16 Snap shot of Dataset. . . . .	38
4.17 Average score of human grades. . . . .	38
4.18 Sequence matrix. . . . .	39
4.19 Output of Sequence matrix. . . . .	39
4.20 Model Definition. . . . .	39
4.21 Model Fitting. . . . .	41
4.22 Model Prediction. . . . .	42
4.23 Snapshot of XGBoost Classifier. . . . .	43

4.23	Advantages of XGBoost. ....	43
4.24	Classification Report. ....	44
4.25	Snapshot of Output csv file. ....	44
5.1	Output of Manually inserted Data ....	46
5.2	Snapshot of system at end-user ....	46
5.3	Input Essays ....	47
5.4	Classifier Report ....	47
5.5	Before Target Encoding ....	48
5.6	Output After Target Encoding ....	48
5.7	Number of Essays VS Essay Scores(Variation in grades). ...	48
5.8	Output of manually inserted data ....	49

# **Chapter 1**

## **Introduction**

### **1.1 Problem Statement**

In today's era of digital transformation, the educational sector faces a couple of challenges such as unqualified teachers in tier two and three city schools of India, less facilities and lack of implementation of modern technology posing to be an important hindrance for substantial growth in the educational sector. In the quick need for changing the way we learn at schools and institutions the schools have started prioritizing the need to build students for industrial sustainability. A major hindrance in this transformation is unnecessary human interference in grading essays and paragraphs which is often a very frequent activity in modern day schools and professional exams such as the TOEFL and IELTS.

### **1.2 Scope of the Project**

- Accurately perceive data sets and assess ungraded data with no or minimal variance from human judgement.
- Easy to use for a person without any technical knowledge or background functionality of the algorithm.
- An efficient, cross platform compatible software for performing the above.

- Users without a technical knowledge background will be able to understand the working of our software and easily be able to use functionalities within.

### **1.3 Current Scenario**

Even in today's times of modern technological era, school and college teachers spend a large amount of time reading long articles such as essays and paragraphs for grading student's essay which is time consuming and requires a lot of human effort. This leads to a lot of avoidable financial expenditure as well. Manual effort is wasted where computers can solve the need using Artificial intelligence. This is a problematic situation in cases where thousands of essays need to be assessed such as IELTS or TOEFL examinations. There has been a lot of work done previously a few of which have been stated along below. Our motive will be to counter every issue that has come up previously.

### **1.4 Need for the Proposed System**

- Manpower which can be utilized elsewhere is unnecessarily wasted in evaluation of essays and paragraphs.
- Computers are overruling today in many fields. Using them for this application will surface their involvement in this field.
- Computers are becoming more and more popular day by day even in remote areas thereby making the functionality easy i.e everyone would want a system which can reduce human effort and is feasible.
- Examinations like GRE and TOEFL have thousands of applicants and it's a tedious task to moderate so many answer sheets.
- There are institutions which have a financial constraint of having many faculties. Such institutions can implement this for correcting answer sheets and continue to save costs. However this is a future scope

### **1.5 Summary of the results and task completed**

The result we get in the end is a grade between 0 to 3. also the feedback for the same. User can re-enter the input based on the feedback provided.

## **Chapter 2**

### **Review of Literature**

#### **2.1 Summary of the investigation in the published papers**

##### **1. Design Of An Automated Essay Grading (AEG) System In Indian Context**

In his paper ‘Region Effects in AEG human discrepancies of TOEFL score’ Attali (2005) mentioned Asian Students show higher organization scores and poor grammar, usage and mechanics scores compare to other students. More over local languages influence them. Serious work in the area of AEG can bring significant changes in this direction and also can give a new shape to Indian NLP Machine Learning research work. Future plans - In near future the following things will be taken into consideration so that some solutions can be given as - Solution for machine translated essays (how to recognize them?), Capturing the mental status of the student writing essay (psychometric models will be considered).

##### **2. Automated Essay Grading Using Features Selection -**

This research shows that automated essay grading will help writers to know their own level of competency. However, there are many areas of improve-ment especially with the complexity for languages around the world even in

this system some of the critical grammatical errors are quite impossible to register they have gone unnoticed. Nonetheless, it is quite evident what machine learning can do if the agent can be trained properly with large set of relevant data and reasonably good algorithm will even make the learning easier. This research suggests that, although statistically significant differences existed between instructor- and AES grading, it would be quite preferable to use this interface. Consequently, depending on the intent of individual instructors for their chosen assignment, these systems may be more acceptable as a formative, as opposed to summative, assessment of student learning

### **3. ASAP++: Enriching the ASAP Automated Essay Grading Dataset with Essay Attribute Scores -**

In this paper, we present a manually annotated dataset for automated essay grading. The annotation was done for different attributes of the essays. Most of the essays were annotated by a single annotator. However, about a sixth of them were annotated by a second annotator. These annotations can be used as a gold standard for future experiments in predicting different attribute scores. The resource is available online at <https://cfilt.iitb.ac.in/egdata/>. The resource is available for non-commercial research use under the Creative Commons Attribution-Share Alike License<sup>8</sup>.

### **4. Neural Network Approach To Automated Essay Scoring**

In this paper, we have proposed an approach based on recurrent neural networks to tackle the task of automated essay scoring. Our method does not rely on any feature engineering and automatically learns the representations required for the task. We have explored a variety of neural network model architectures for automated essay scoring and have achieved significant improvements over a strong open-source baseline. Our best system outperforms the baseline by 5.6 an analysis of the network has been performed to get an insight of the recurrent neural network model and we show that the method effectively utilizes essay content to extract the required information for scoring essays.

## 2.2 Comparison between the tools / methods / algorithms

Sr No.	PAPER TITLE	AUTHORS	TECHNOLOGY USED	ISSUES
1.	Design of an Automated Essay Grading (AEG) System in Indian Context	Siddhartha Ghosh Dr. Sameen S Fatima	Natural language processing , ML	Can take input of only 300 words
2.	AUTOMATED ESSAY GRADING USING FEATURES SELECTION	Y.Harika , I.Sri Latha , V.Lohith Sai , P.Sai Krishna , M.Suneetha	NLP with stemming and tokenization	Grammatical errors are hard to identify
3.	ASAP++: Enriching the ASAP Automated Essay Grading Dataset with Essay Attribute Scores	Abeer Zaroor, Mohammed Maree Muath , Pushpak Bhattacharyya	QWK and random forest classifier	Compares expected essays and observed essays .
4.	E-rater used by GRE	J. Burstein, J. Tetreault	Supervised Learning Algorithm, CNN QWK	Does not count regional words as input and directly discard them
5.	A neural network approach to automated essay scoring	Kaveh taghipoor , Hwee tou Ng	RNN , QWK	Processing time is too slow and requires a lot of space

Figure 2.1: Comparison of different existing systems

## **Chapter 3**

### **Analysis and Design**

#### **3.1 Methodology / Procedure adopted**

The development methodology used is Agile Methodology. We conduct meetings every 2 days in a week. Topics told in the previous meeting are studied and implemented and then discussed in the meeting. After the given task is done, new tasks are divided among the team to study and implemented. The progress of the work is written in the project booklet.



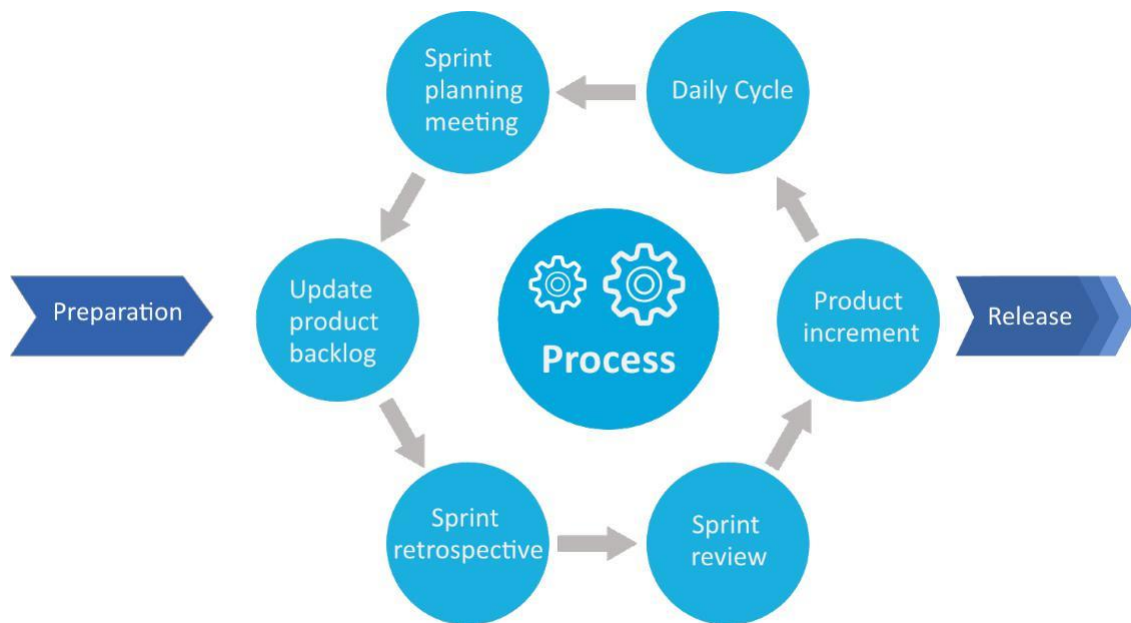


Figure 3.1: Agile Methodology

## 3.2 Analysis

As per the university timetable, we were granted a total of 8 hours in college per week to work together under the guidance of our supervisor Prof. Uday Nayak. Along with the discussions during these meeting hours, each of us discussed and shared the remaining work in case left. After the spread of corona and government lockdown, we used team viewer and zoom video conferencing calls to finish the last deliverables a part of which is this whole report document.

As mentioned, we used agile methodology as it was best supported by our schedule and college timetable. However, learnt in our course and intern-ships we understood that the deliverables achieved were the only parameters of achievements in the information technology industry. So, we intended on keep-ing up with our progress stated in the Gantt chart as per which we produced 100

The feasibility study consists of a five-point model known as TELOS and our research related to the below mentioned are stated — this includes the following components:

### *Technical*

We used JUPYTER for offline code compilation, GOOGLE COLAB for online code compilation and both of them are open source applications. Since

our project requires processing of high number of data sets, a computer system with a high computing power especially RAM is required.

### ***Economic***

No economic drawbacks were observed in our study.

### ***Legal***

The applications used were completely legal and open source and thereby free for any amount of use and distribution had no legal obligations.

### ***Operational***

No operational drawbacks were found. Our system requires little to no maintenances leaving regular updates of open source applications carried out by the licensor.

### ***Schedule***

No drawbacks were observed.

Requirements that were modified.

As stated in our previous presentation reports, we intended on making a our model grade essays on factors such as Grammar, Lexical Complexity, Mechanics , Usage , Organization , Vocabulary usage , Style , Discourse analysis , Ideas and plagiarism. However, during presentations, interactions with our guide and during ongoing work, its feasibility was difficult and we decided on keeping it restricted only to clarity and coherency.

## **3.2.1 Software / System Requirement Specification**

*Libraries used are below: -*

[1] *NUMPY* -NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.

[2] *KERAS*- Keras is an open-source neural-network library written in Python. It is capable of running on top of TensorFlow, Microsoft Cognitive Toolkit,

Theano, or Plaid ML. Designed to enable fast experimentation with deep neural networks, it focuses on being user-friendly, modular, and extensible.

[3] *NLTK*-The Natural Language Toolkit, or more commonly NLTK, is a suite of libraries and programs for symbolic and statistical natural language processing for English written in the Python programming language.

[4] *SKLEARN*-The Natural Language Toolkit, or more commonly NLTK, is a suite of libraries and programs for symbolic and statistical natural language processing for English written in the Python programming language.

[5] *MATPOLIB*-The Natural Language Toolkit, or more commonly NLTK, is a suite of libraries and programs for symbolic and statistical natural language processing for English written in the Python programming language.

[6] *XG-BOOST*- It is an open-source software library which provides a gradient boosting framework for C++, Java, Python, R, Julia, Perl, and Scala. It works on Linux, Windows, and macOS. From the project description, it aims to provide a “Scalable, Portable and Distributed Gradient Boosting Library”

### 3.3 System Architecture

The two major components of the system are Score Reporter And Feedback generator.

Score Reporter

Feedback generator

1. Input - The input of the training dataset is given to the model, the input consists of 1700 different essays of 10 different topics i.e. Essay sets. And then there are 4 columns which are Score1, Score2, Score3 and Score4 which are the actual grades of the essays. Then there are two different scores i.e. clarity and coherence.

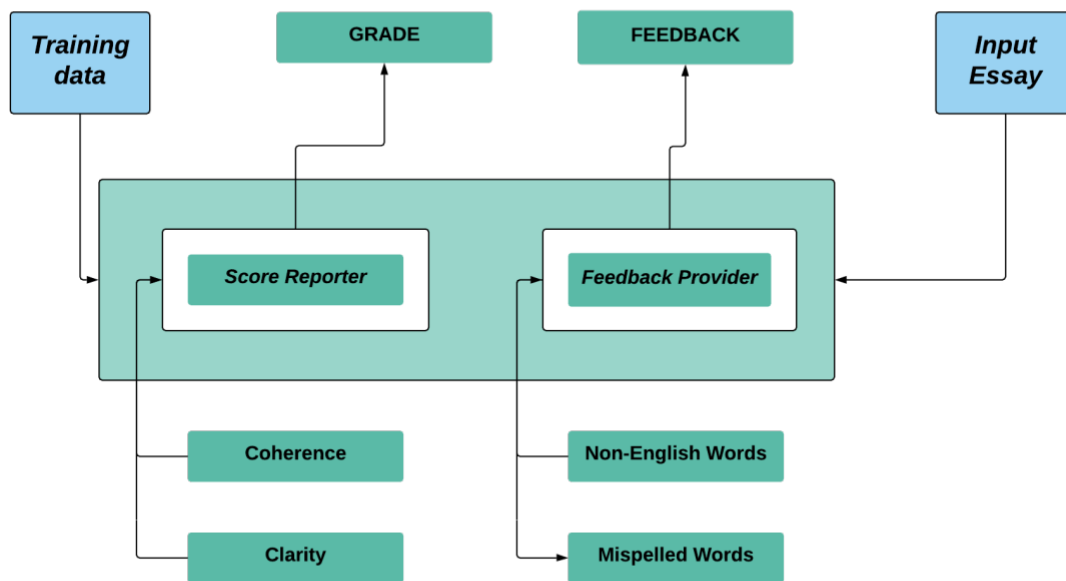


Figure 3.2: System Architecture

2. Score reporter - Score reporter takes clarity and coherence value as the input and also the average score of human graded essay and then generates the output in the form of 0 to 3 where 0 is the lowest number and 3 is the highest.
3. Feedback Provider - Feedback provider works on very simple logic , It takes out the misspelled words from and the essay and asks the user to re-enter it , It also takes out the non-English words from the essays and displays them for correction.
4. Input essay - The essay should be of maximum 2500words and the user can enter it in the textbox displayed

**Training data** -The dataset is picked up from kaggle and it has 12 columns and 27588 rows. The columns are [Id, Essayset, score\_1, score\_2, score\_3, score\_4, score\_5, min\_score, max\_score, clarity, coherence, EssayText]. There are total 10 different essay set i.e. 10 types of essays.

	ID	Essayset	min_score	max_score	score_1	score_2	score_3	score_4	score_5	clarity	coherent	EssayText
0	1	1.0	0	3	1	1	1.0	1.0	1.0	average	worst	additional information would need replicate exl..
1	3	1.0	0	3	1	1	1.0	1.0	1.5	worst	above_average	need trial control set exact amount vinegar po...
2	4	1.0	0	3	0	0	0.0	0.0	1.0	worst	worst	student list rock better rock worse procedure
3	5	1.0	0	3	2	2	2.0	2.5	1.0	above_average	worst	student able make replicate would need tell us...
4	6	1.0	0	3	1	0	0.0	0.0	0.0	worst	worst	would need information would let different sam...

Figure 3.3: Snapshot of Dataset

Let us have a look at it ,

1. **ID** - Unique ID given to each essay.
2. **Essayset** - There are 10 different essay sets i.e. 10 different topic.
3. **Min\_score** - the minimum score i.e. 0
4. **Max\_score** - the maxim score i.e. 1
5. **Score\_1 , Score\_2.....Score\_5** - the essays are judged by 5 different people.
6. **Clarity**- Essays are judged on the basis of clarity.
7. **Coherence** - Also , they are judged based on coherence.
8. **Essaytext** - the dataset consists of 16000 essays on 10 different topics.

The most important component of our system is the score reporter and it's development is divided into several phases , Now we will see how it works , how does it take input and what exactly it gives us the output.

**Background** One of the key roadblocks to advancing school-based curricula focused on critical thinking and analytical skills is the expense associated with scoring tests to measure those abilities. For example, tests that require essays and other constructed responses are useful tools, but they typically are hand scored, commanding considerable time and expense from public agencies. So, because of those costs, standardized examinations have increasingly been limited to using “bubble tests” that deny us opportunities to challenge our students with more sophisticated measures of ability.

Recent developments in automating assessment of student essays and other response types are promising. And, states are showing increasing interest in them. To meet their needs, The William and Flora Hewlett Foundation (Hewlett) is sponsoring a series of competitions through the Automated Student Assessment Prize(ASAP), in which new participants will compete to demonstrate their capabilities for automating the grading process.

Demonstrations will be based on scoring actual student essays that were graded by experts, to determine whether or not competing solutions can deliver the same accuracy and reliability. The first prize focuses on “long-form constructed response” (essays), but it will be followed by other prizes in the months ahead using other forms of graded student content. Hewlett intends to drive innovation in this sector, at a time when state departments of education are working towards adopting new and more sophisticated student assessments.

Hewlett believes that breakthroughs will lead to adoption of these models at a time when the need for solutions is critical to improving education. Hewlett believes that there is – more than ever before – vital interest among a large base of commercial interest. We are asking for your participation, both to improve the conditions that are inhibiting public education and to present viable and effective business solutions among a strongly motivated base of investors.

The **score reporter** takes clarity, coherence and the essay as an input

**Clarity** - **Clarity means** clearness. Clean water running down a mountain **has clarity**. So **does** a lovely singing voice: it's clear and pure. If you bring **clarity** to a situation, you help people see what really happened by clearing up misunderstandings or giving explanations.

**Coherence** - Coherence is one of the two qualities that give a written or spoken text unity and purpose. The other is cohesion. Coherence refers to the general sense that a text makes sense through the organisation of its content. In writing, it is provided by a clear and understood structuring of paragraphs and sentences in writing. **Example** - A learner's argument essay is coherent because it has a structure that gives unity and follows an accepted form. It begins with a statement of belief, gives the opposing arguments, refutes these, and summarises in a final paragraph.

**Feedback provider**- Once the model is trained the input essayset is given to the model for prediction and the feedback.the system generates a feedback using Non English words used in the essay and the misspelled words in the essay and the feedback is given based on these factors.The feedback is for the essay writer to improve his essay skills and understand what mistakes she/he is making.

## Chapter 4

### Implementation

#### 4.1 Setting up Implementation Environment

Googlecolab was used for implementation of the Project so as to develop the AEG system fluently. If you have a GPU or a good computer creating a local environment with anaconda and installing packages and resolving installation issues are a hassle.

Colaboratory is a free Jupyter notebook environment provided by Google where you can use free GPUs and TPUs which can solve all these issues.

#### Installation procedure

To start working with Colab you first need to log in to your google account, then go to this link <https://colab.research.google.com>.

#### Opening Jupyter Notebook:

On opening the website, you will see a pop-up containing following tabs –

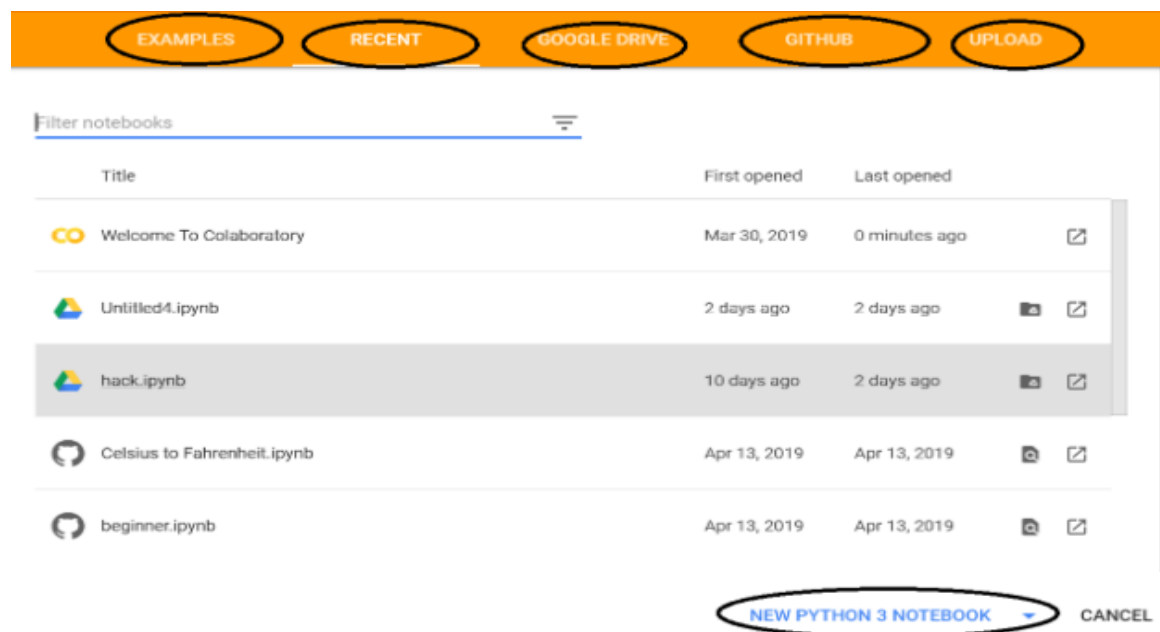


Figure 4.1: Frontpage of Jupyter notebook



**EXAMPLES:** *Contain a number of Jupyter notebooks of various examples.*

**RECENT:** *Jupyter notebook you have recently worked with.*

**GOOGLE DRIVE:** *Jupyter notebook in your google drive.*

**GITHUB:** *You can add Jupyter notebook from your GitHub but you first need to connect Colab with GitHub.*

**UPLOAD:** *Upload from your local directory.*

Else you can create a new Jupyter notebook by clicking New Python3 Notebook or New Python2 Notebook at the bottom right corner.

### Notebook's Description:

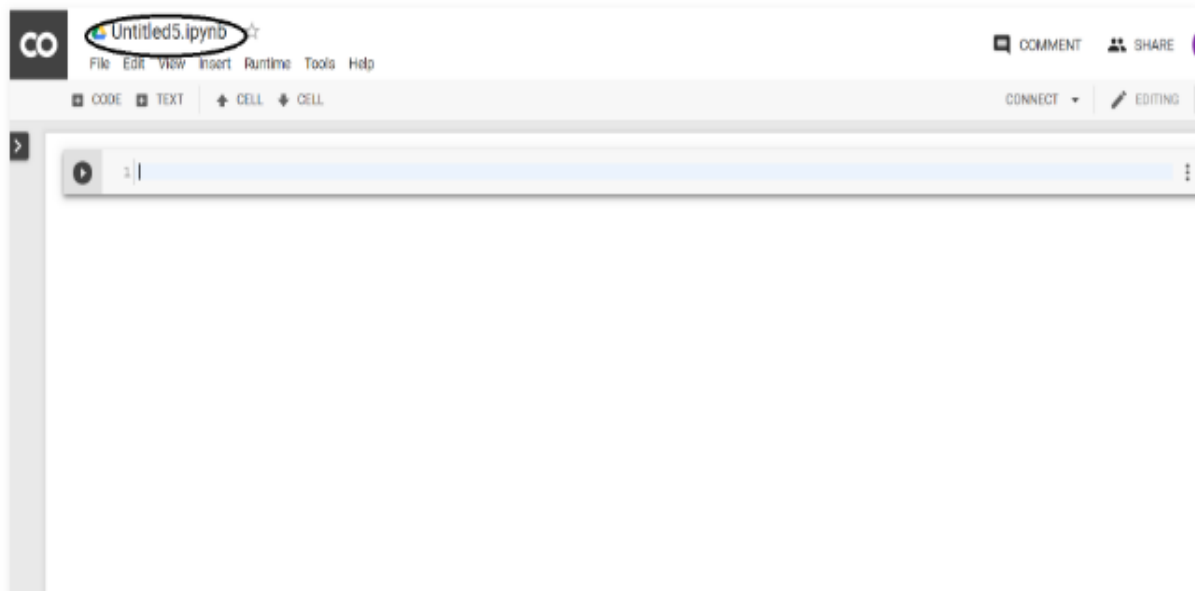


Figure 4.2: Creating New Python3 Notebook

On creating a new notebook, it will create a Jupyter notebook with `Untitled0.ipynb` and save it to your google drive in a folder named **Colab Notebooks**. Now as it is essentially a Jupyter notebook, all commands of Jupyter notebooks will work here. Though, you can refer the details in [Getting started with Jupyter Notebook](#)

**Change Runtime Environment:**

Click the “**Runtime**” dropdown menu. Select “**Change runtime type**”. Select python2 or 3 from “**Runtime type**” dropdown menu.

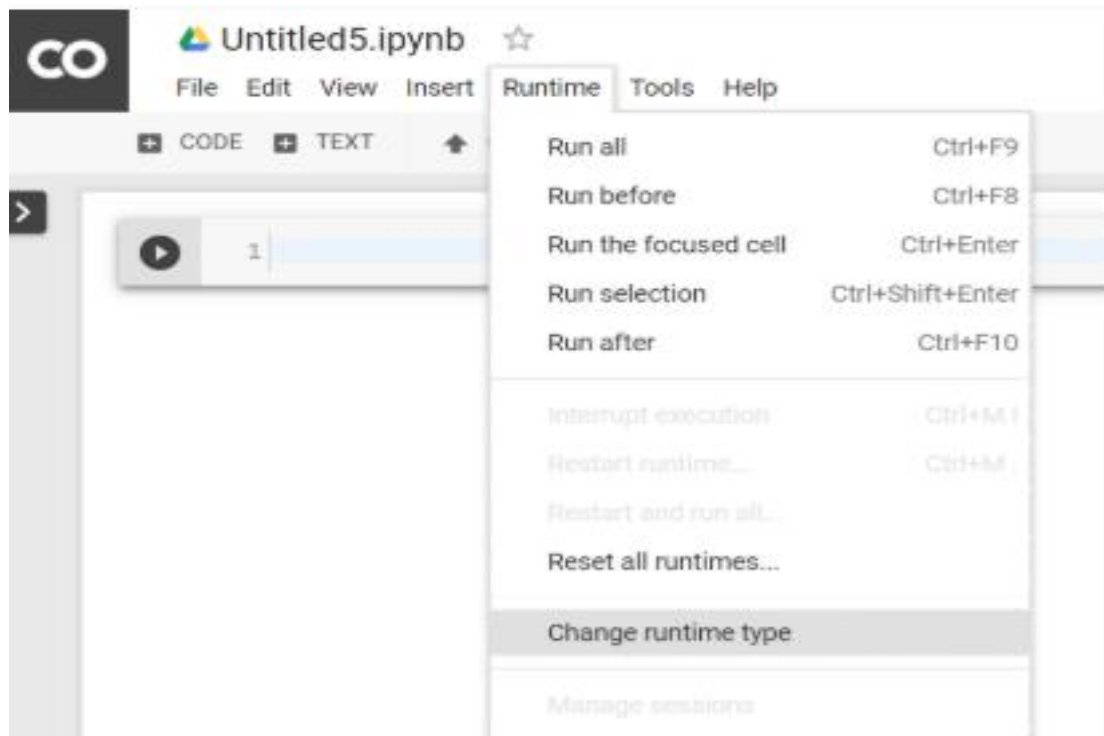


Figure 4.3: Changing runtime of notebook

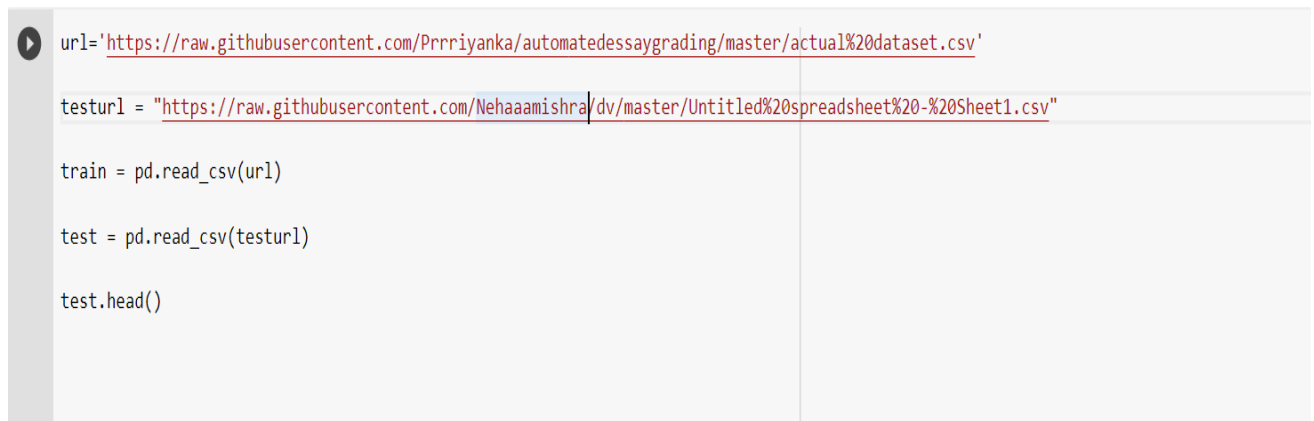
## Setting up the environment for coding of website

### Install Xampp:

1. Open the XAMPP website. Go to <https://www.apachefriends.org/index.html> in your computer's web browser.
2. Click XAMPP for Windows. It's a grey button near the bottom of the page.  
Depending on your browser, you may first have to select a save location or verify the download.
3. Double-click the downloaded file. This file should be named something like xampp-win32-7.2.4-0-VC15-installer, and you'll find it in the default downloads location (e.g., the "Downloads" folder or the desktop).
4. Click Yes when prompted. This will open the XAMPP setup window.
5. Click Next. It's at the bottom of the setup window.
6. Select aspects of XAMPP to install. Review the list of XAMPP attributes on the left side of the window; if you see an attribute that you don't want to install as part of XAMPP, uncheck its box.
7. Click Next. It's at the bottom of the window.
8. Select an installation location. Click the folder-shaped icon to the right of the current installation destination, then click a folder on your computer.
9. Click OK. Doing so confirms your selected folder as your XAMPP installation location. Click Next. You'll find it at the bottom of the page.
10. Uncheck the "Learn more about Bitnami" box, then click Next. The "Learn more about Bitnami" box is in the middle of the page.
11. Begin installing XAMPP. Click Next at the bottom of the window to do so. XAMPP will begin installing its files into the folder that you selected.
12. Click Finish when prompted. It's at the bottom of the XAMPP window. Doing so will close the window and open the XAMPP Control Panel, which is where you'll access your servers. Select a language. Check the box next to the American flag for English, or check the box next to the German flag for German.
13. Click Save. Doing so opens the main Control Panel page. Start XAMPP from its installation point. If you need to open the XAMPP Control Panel in the future, you can do so by opening the folder in which you installed XAMPP, right-clicking the orange-and-white xampp-control icon, clicking Run as administrator, and clicking Yes when prompted.

When you do this, you'll see red X marks to the left of each server type (e.g., "Apache"). Clicking one of these will prompt you to click Yes if you want to install the server type's software on your computer.

Counterintuitively, double-clicking the xampp start icon doesn't start XAMPP.



```
url='https://raw.githubusercontent.com/Prriyanka/automatedessaygrading/master/actual%20dataset.csv'

testurl = "https://raw.githubusercontent.com/Nehaaamishra/dv/master/Untitled%20spreadsheet%20-%20Sheet1.csv"

train = pd.read_csv(url)

test = pd.read_csv(testurl)

test.head()
```

Figure 4.4: Loading Dataset in notebook

Initially when the system was being developed, we had to take input in the form of a csv file, since we used google colab to run the system we had to pre-store the data on GitHub. Then we got the raw GitHub URL and from that URL the system was taking the input working on it and generating a score.

But now we have made it very easy we came up with a web portal where user can give the input essay click on submit , after clicking on submit the Input gets stored in a csv file , our system reads that csv file , works on it generates the output and stores it in the same row of the essay under the column name **score**

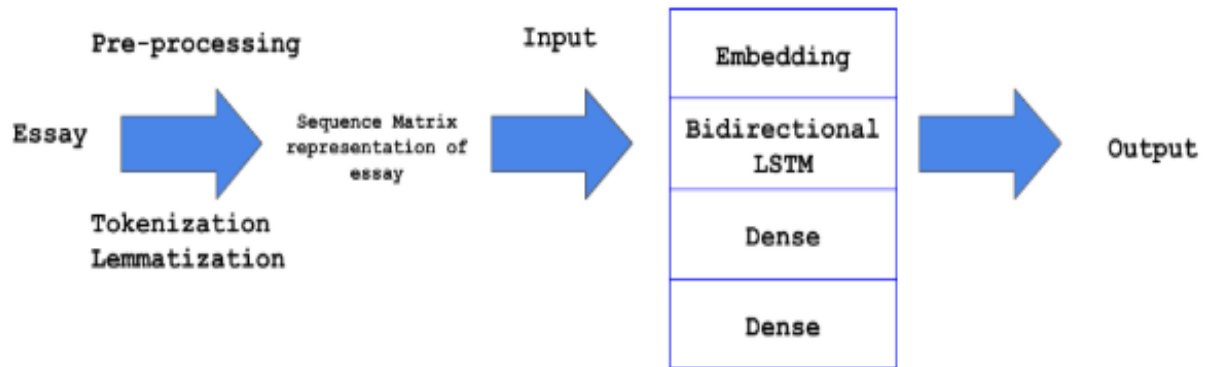


Figure 4.5: Workflow of score reporter of AEG system.

### Step 1. Input essay

Enter your details

Thank you

Enter Essayset

Enter essay

a computer is a device that accepts the message by the imputer and processes this message and stores the information at the storage devices and later gives

Submit

Figure 4.6: Input essay

## Step 2 Sequence matrix generation

Let's start to formalize the problem of learning a good word embedding. When you implement an algorithm to learn a word embedding, what you end up learning is an embedding matrix. Let's take a look at what that means. Let's say, as usual we're using our 10,000-word vocabulary. So, the vocabulary has A, Aaron, Orange, Zulu, maybe also unknown word as a token. What we're going to do is learn embedding matrix  $E$ , which is going to be a 300 dimensional by 10,000 dimensional matrix, if you have 10,000 words vocabulary or maybe 10,001 is our word token, there's one extra token. And the columns of this matrix would be the different embeddings for the 10,000 different words you have in your vocabulary. So, Orange was word number 6257 in our vocabulary of 10,000 words. So, one piece of notation we'll use is that  $O_{6257}$  was the one-hot vector with zeros everywhere and a one in position 6257. And so, this will be a 10,000-dimensional vector with a one in just one position. So, this isn't quite a drawn scale. Yes, this should be as tall as the embedding matrix on the left is wide. And if the embedding matrix is called capital  $E$  then notice that if you take  $E$  and multiply it by just one-hot vector by  $O$  of 6257, then this will be a 300-dimensional vector.

So,  $E$  is 300 by 10,000 and  $O$  is 10,000 by 1. So, the product will be 300 by 1, so with 300-dimensional vector and notice that to compute the first element of this vector, of this 300-dimensional vector, what you do is you will multiply the first row of the matrix  $E$  with this. But all of these elements are zero except for element 6257 and so you end up with zero times this, zero times this, zero times this, and so on. And then, 1 times whatever this is, and zero times this, zero times this, zero times and so on. And so, you end up with the first element as whatever is that elements up there, under the Orange column. And then, for the second element of this 300-dimensional vector we're computing, you would take the vector  $O_{6257}$  and multiply it by the second row with the matrix  $E$ . So again, you have zero times this, plus zero times this, plus zero times all of these are the elements and then one times this, and then zero times everything else and add that together. So you end up with this and so on as you go down the rest of this column. So, that's why the embedding matrix  $E$  times this one-hot vector here winds up selecting out this 300-dimensional column corresponding to the word Orange. So, this is going to be equal to  $E_{6257}$  which is the notation we're going to use to represent the embedding vector that 300 by one dimensional vector for the word Orange. And more generally,  $E$  for a specific word  $W$ , this is going to be embedding for a word  $W$ . And more

generally,  $E$  times  $O$  substitute  $J$ , one-hot vector with one that position  $J$ , this is going to be  $E_J$  and that's going to be the embedding for word  $J$  in the vocabulary. So, the thing to remember from this slide is that our goal will be to learn an embedding matrix  $E$  and what you see in the next video is you initialize  $E$  randomly and you're straight in the sense to learn all the parameters of this 300 by 10,000 dimensional matrix and  $E$  times this one-hot vector gives you the embedding vector.

Now just one note, when we're writing the equation, it'll be convenient to write this type of notation where you take the matrix  $E$  and multiply it by the one-hot vector  $O$ . But if when you're implementing this, it is not efficient to actually implement this as a matrix vector multiplication because the one-hot vectors, now this is a relatively high dimensional vector and most of these elements are zero.

So, it's actually not efficient to use a matrix vector multiplication to implement this because if we multiply a whole bunch of things by zeros and so the practice, you would actually use a specialized function to just look up a column of the Matrix  $E$  rather than do this with the matrix multiplication. But writing the map, it is just convenient to write it out this way. So, in Cara's for example there is an embedding layer and we use the embedding layer then it more efficiently just pulls out the column you want from the embedding matrix rather than does it with a much slower matrix vector multiplication. the notations were used to describe algorithms to learning these embeddings and the key terminology is this matrix capital  $E$  which contain all the embeddings for the words of the vocabulary.

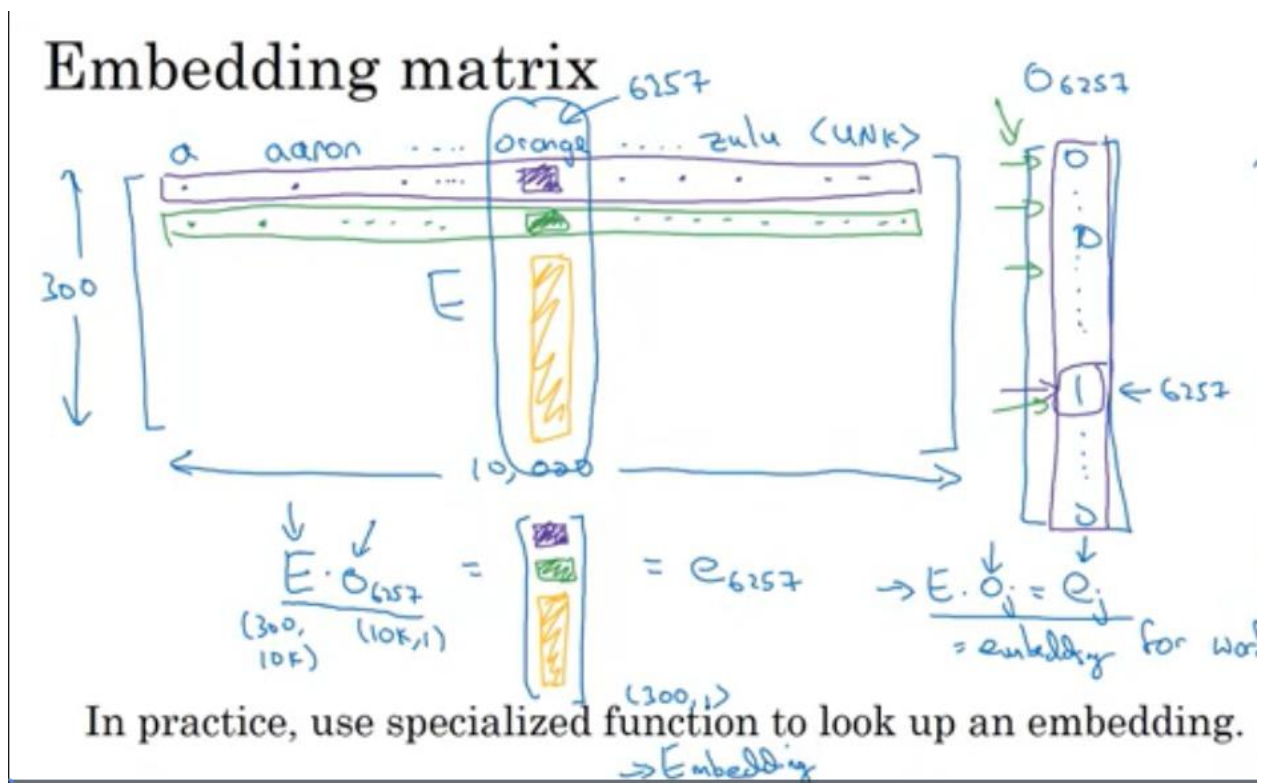


Figure 4.7: Embedding matrix explanation

```
print(sequences_matrix)
Y1=train2.avg_score
```

```
[
  [ 0  0  0 ... 26 415 52]
  [ 0  0  0 ... 439 785 316]
  [ 0  0  0 ... 734 1756 121]
  ...
  [ 0  0  0 ... 267 181 130]
  [ 0  0  0 ... 267 65 58]
  [ 0  0  0 ... 399 43 82]]
```

Figure 4.8: Sequence Matrix generation



## 4.2 Implementation Plan for Sem – 8



Figure 4.9: Gantt Chart of AEG

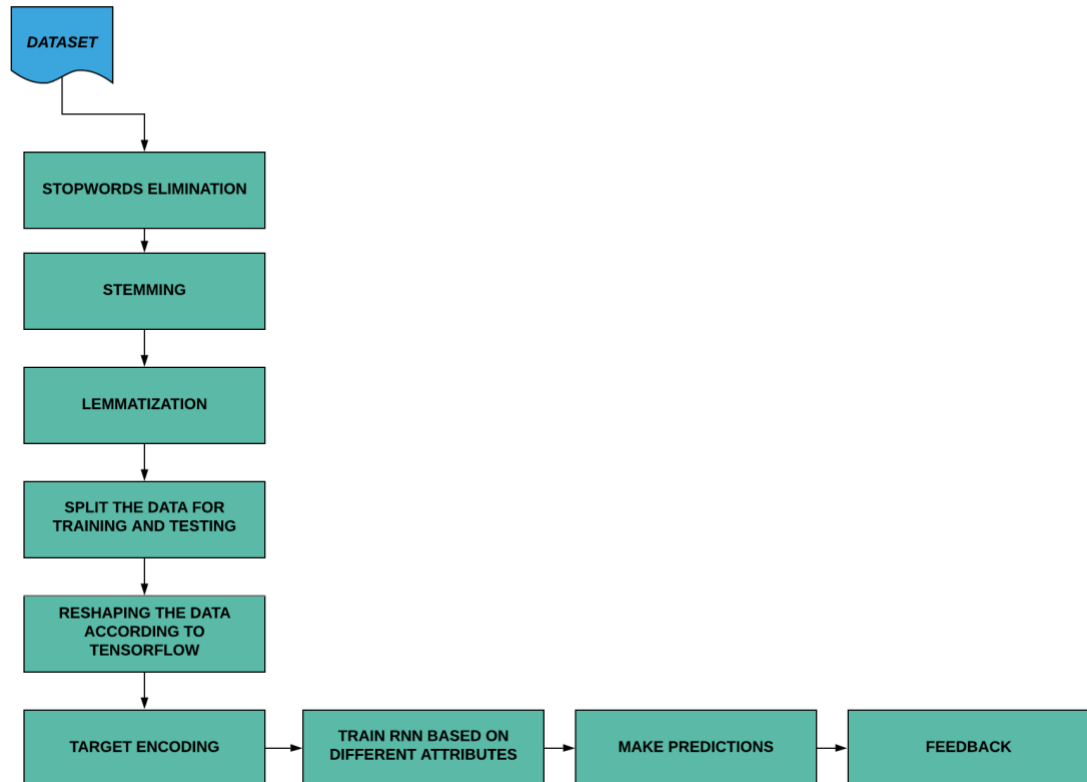


Figure 4.10: Flow of the AEG System

## 4.2 Implementation Of the System

1. **Stop words Elimination / Pre-processing-** Stop words are a set of commonly used words in any language. For example, in English, “the”, “is” and “and”, would easily qualify as stop words. In NLP and text mining applications, stop words are used to eliminate unimportant words, allowing applications to focus on the important words instead.
2. **Stemming** - Stemming is the process of reducing a word to its word stem that affixes to suffixes and prefixes or to the roots of words known as a lemma. Stemming is important in natural language understanding (NLU) and natural language processing (NLP). ... When a new word is found, it can present new research opportunities.
3. **Tokenization and Stemming-** We have defined maximum length as 2500 i.e. system can take essays with input up to 2500 words. In this section we try to divide the whole essay into tokens. After generating the tokens. A matrix known as a sequence matrix is generated to be fed in the model.

4. **Splitting the data into train and testing** -Here, we split the data into the training part and testing part. test size is 0.2 which denotes that 20 % of data will be used to test the accuracy of the model. Now , in the training part we put the tokenized matrix on x axis and on y axis we put the average score.
5. **Defining the model**- A recurrent neural network (RNN) is a class of artificial neural network where connections between units form a directed graph along a sequence. This allows it to exhibit dynamic temporal behaviour for a time sequence. Unlike feedforward neural networks, RNNs can use their internal state (memory) to process sequences of inputs. This makes them applicable to tasks such as unsegmented, connected handwriting recognition or speech recognition. Recurrent Neural Network comes into the picture when any model needs context to be able to provide the output based on the input. Sometimes the context is the single most important thing for the model to predict the most appropriate output. Let's understand this by an analogy. Suppose you are watching a movie, you keep watching the movie as at any point in time, you have the context because you have seen the movie until that point, then only you are able to relate everything correctly. It means that you remember everything that you have watched. Similarly, RNN remembers everything. In other neural networks, all the inputs are independent of each other. But in RNN, all the inputs are related to each other. Let's say you have to predict the next word in a given sentence, in that case, the relation among all the previous words helps in predicting the better output. The RNN remembers all these relations while training itself. In order to achieve it, the RNN creates the networks with loops in them, which allows it to persist the information.
6. **Target encoding**- It is the process of replacing a categorical value with the mean of the target variable. Any non-categorical columns are automatically dropped by the target encoder model. Category encoders are used to convert the categorical value of clarity and coherence columns into numeric value which fits into the model
7. **Model Predictions** - The model will Predict 4 probabilities. The greatest one will be considered. Suppose the output of model is [0.33,0.22,0.11,0.44] then the best possible score of the essay is 3.

8. **Feedback-** This module will provide feedback about the essay if any mis-spelled or non-English words are there then it'll ask the user to re-enter input with corrections

### 4.3 Roadmap of The System

The system was developed in five phases namely,

**Identify-** In this phase the problem statement was identified. The requirements were gathered. Identification of problem statement -

A problem statement is a concise description of an issue to be addressed or a condition to be improved upon. It identifies the gap between the current (problem) state and desired (goal) state of a process or product. Focusing on the facts, the problem statement should be designed to address the Five W's.

Requirement gathering in this phase most of the requirements related to project were gathered and then what exactly needs to be done, what software and how much computation is required were collected.

Planning a literature survey

In this phase the current systems need to be studied, so the plan for the literature survey was made and executed.

**Analyse** - In this phase the current systems were studied and literature survey took place.

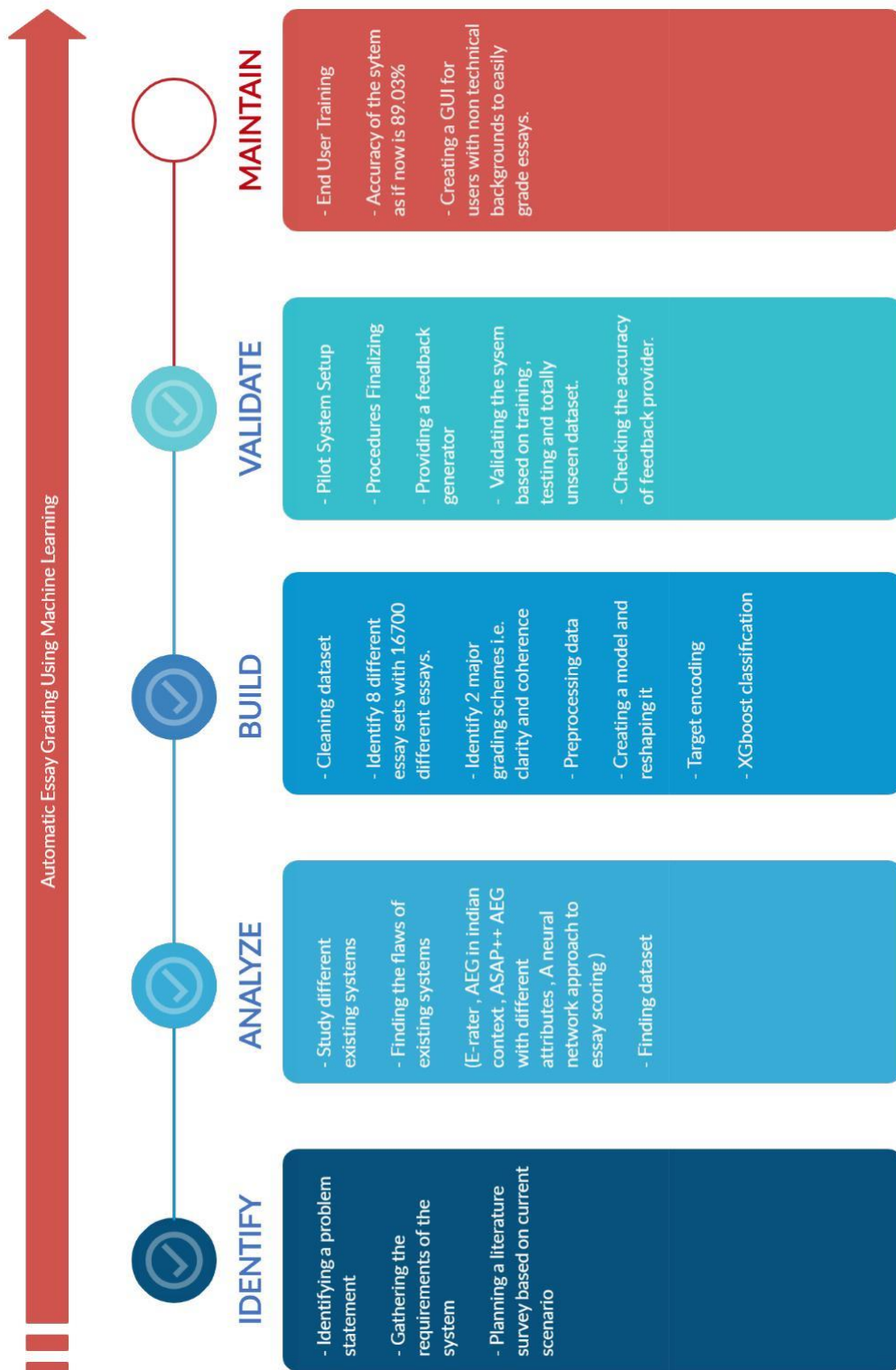
Current systems where studied thoroughly and the flaws were identified. the system which are studied under this phase are E-rater , AEG-grader , E-rater used in TOEFL. the dataset was the most important part of the project. so the accurate dataset was found out and it was executed.

**Build** - In this phase the actual system was designed and developed. This was the most important phase. the major building processes were decided in this phase. the two major functionalities of the system were decided and system was developed accordingly as we have used agile methodology so every week the system was upgraded accordingly. the model training and other processes were majorly done here

**Validate** - In this phase modules of the systems were tested. the accuracy of the system was checked. and various testing were done on this phase and the system was improved accordingly

**Maintain** - In this phase the UI of the algorithm will be improved and the end user training will be given to the people.

Figure 4.11: Roadmap of the system



## 4.4 Testing

### 4.4.1 Testing Methods

Blackbox Testing was done on the system, Blackbox testing is testing the functionality of an application without knowing the details of its implementation including internal program structure, data structures, etc. Test cases for black-box testing are created based on the requirement specifications. Therefore, it is also called as specification-based testing. The following diagram represents the Blackbox testing:

There were different parameters based on which the system was tested which are mentioned below

Model performance- The accuracy of the model is tested and improved.

Metamorphic Testing- Testing model performance is about testing the models with the test data/new data sets and comparing the model performance in terms of parameters such as accuracy/recall etc., to that of pre-determined accuracy with the model already built and moved into production. This is the most trivial of different techniques which could be used for Blackbox testing.

Dual Coding -With dual coding technique, the idea is to build different models based on different algorithms and comparing the prediction from each of these models given a particular input data set. Let's say, a classification model is built with different algorithms such as random forest, SVM, neural network. All of them demonstrate a comparative accuracy of 90% or so with random forest showing the accuracy of 94%

Testing with different data slices-So there are two types of data that were tested first when the essay was totally known to the model one of the dataset and the other one which is totally new to the system.

## 4.5 Coding Standards

### Implementation of the system with code, Explanation and output.

```
%tensorflow_version 1.x
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from keras.models import Model
from keras.layers import LSTM, Activation, Dense, Dropout, Input,
Embedding, Bidirectional, BatchNormalization
from keras.optimizers import RMSprop
from keras.preprocessing.text import Tokenizer
from keras.preprocessing import sequence
from keras.utils import to_categorical
from keras.callbacks import EarlyStopping
from keras.utils import to_categorical
%matplotlib inline
```

Figure 4.12: Snapshot of Imports

*%tensorflow\_version 1.x* -is a "magic" command ("magic spell") in GoogleColab that instructs the Colab environment to use the newest stable release of Tensorflow version 1

**Pandas** - Pandas is an open source Python package that provides numerous tools for data analysis. The package comes with several data structures that can be used for many different data manipulation tasks.

It is used to present data in a way that is suitable for data analysis via its **Series** and **DataFrame** data structures.

**Numpy** - NumPy is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays. It is the fundamental package for scientific computing with Python.

**Matplotlib.pyplot** -matplotlib.pyplot is a collection of command style functions that make matplotlib work like MATLAB. Each pyplot function makes some change to a figure: e.g., creates a figure, creates a plotting area in a figure, plots some lines in a plotting area, decorates the plot with labels, etc.

**Scikit-learn** - is a free machine learning library for Python. It features various algorithms like support vector machine, random forests, and k-neighbours, and it also supports Python numerical and scientific libraries like NumPy and SciPy

**Sklearn.model\_selection.train\_test\_split**-Split arrays or matrices into random train and test subsets. Quick utility that wraps input validation and `next(ShuffleSplit().split(X, y))` and application to input data into a single call for splitting (and optionally subsampling) data in a one liner.

**Sklearn.preprocessing**- the sklearn.preprocessing package provides several common utility functions and transformer classes to change raw feature vectors into a representation that is more suitable for the downstream estimators.

**LabelEncoder** -Encode target labels with value between 0 and `n_classes-1`. This transformer should be used to encode target values, *i.e.* `y`, and not the input `X`. It is used to transform non-numerical labels (as long as they are hashable and comparable) to numerical labels.

**Keras**-is TensorFlow's high-level API for building and training deep learning models. It's used for fast prototyping, state-of-the-art research, and production, with three key advantages:

- *User-friendly*  
Keras has a simple, consistent interface optimized for common use cases. It provides clear and actionable feedback for user errors.
- *Modular* and *composable*  
Keras models are made by connecting configurable building blocks together, with few restrictions.
- *Easy* to *extend*  
Write custom building blocks to express new ideas for research. Create new layers, metrics, loss functions, and develop state-of-the-art models.

**Model**(in Keras)-There are two main types of models available in Keras: the Sequential model, and the Model class used with the functional API.

These models have a number of methods and attributes in common:

- `model.layers` is a flattened list of the layers comprising the model.
- `model.inputs` is the list of input tensors of the model.
- `model.outputs` is the list of output tensors of the model.
- `model.summary()` prints a summary representation of your model. For layers with multiple outputs, `multiple` is displayed instead of each individual output shape due to size limitations. Shortcut for `utils.print_summary`



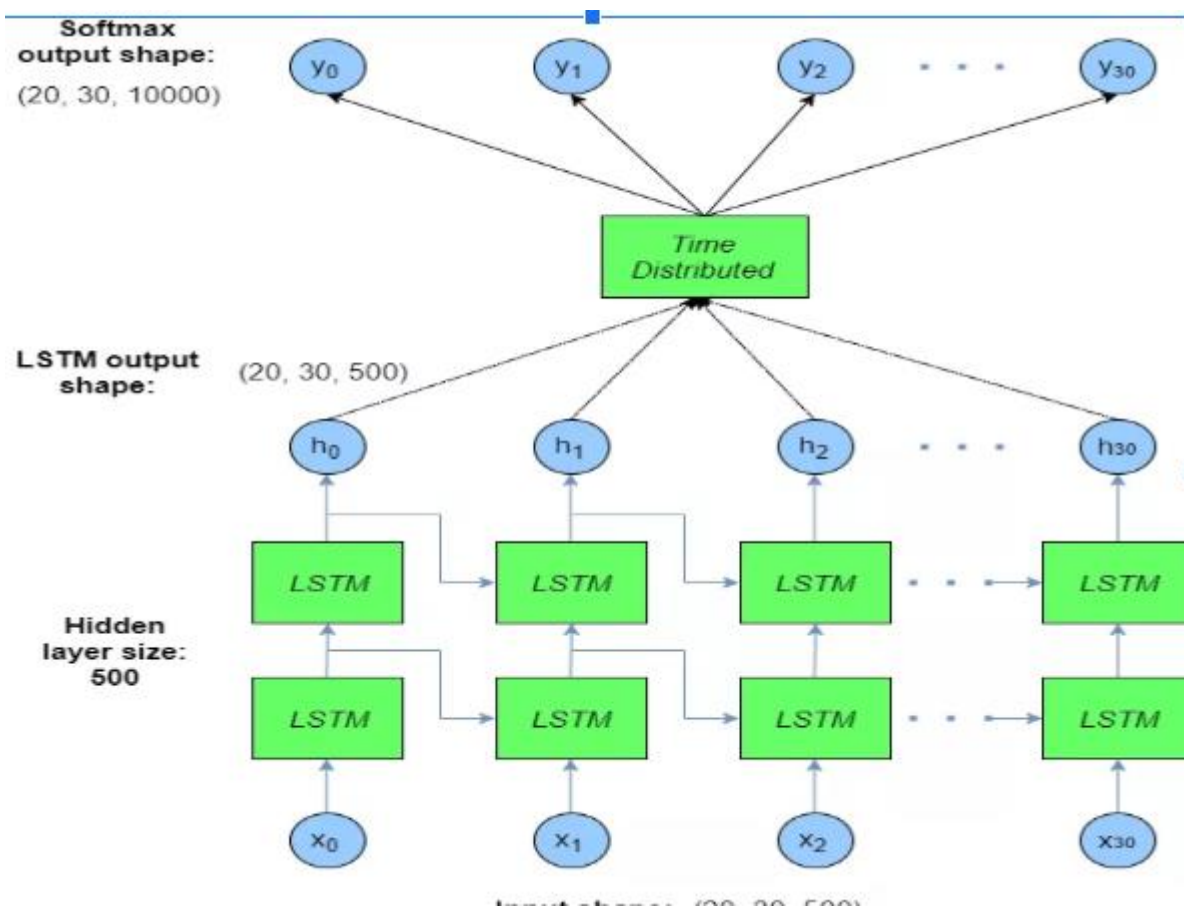
*Keras.layers.LSTM-*

Figure 4.13: Keras lstm layers

The input shape of the text data is ordered as follows : (batch size, number of time steps, hidden size). In other words, for each batch sample and each word in the number of time steps, there is a 500-length embedding word vector to represent the input word. These embedding vectors will be learnt as part of the overall model learning. The input data is then fed into two “stacked” layers of LSTM cells (of 500 length hidden size) – in the diagram above, the LSTM network is shown as unrolled over all the time steps. The output from these unrolled cells is still (batch size, number of time steps, hidden size). This output data is then passed to a Keras layer called Time Distributed, which will be explained more fully below. Finally, the output layer has a *softmax* activation applied to it. This output is compared to the training  $y$  data for each batch, and the error and gradient backpropagation is performed from there in Keras. The training  $y$  data in this case is the input  $x$  words advanced one-time step – in other words, at each time step the model is trying to predict the very next word in the sequence. However, it does this at *every* time step – hence the output layer has the same number of time steps as the input layer.

*keras.optimizers (RMSprop)*-An optimizer is one of the two arguments required for compiling a Keras model.

**RMSProp optimizer** --The RMSprop optimizer restricts the oscillations in the vertical direction. Therefore, we can increase our learning rate and our algorithm could take larger steps in the horizontal direction converging faster. The difference between RMSprop and gradient descent is on how the gradients are calculated.

#### Arguments

- **learning\_rate**: float  $\geq 0$ . Learning rate.
- **rho**: float  $\geq 0$ .

**Text Preprocessing Tokenizer**-Text tokenization utility class.

This class allows to vectorize a text corpus, by turning each text into either a sequence of integers (each integer being the index of a token in a dictionary) or into a vector where the coefficient for each token could be binary, based on word count

#### Arguments

- **num\_words**: the maximum number of words to keep, based on word frequency. Only the most common **num\_words**-1 words will be kept.
- **filters**: a string where each element is a character that will be filtered from the texts. The default is all punctuation, plus tabs and line breaks, minus the ' character.
- **lower**: boolean. Whether to convert the texts to lowercase.
- **split**: str. Separator for word splitting.
- **char\_level**: if True, every character will be treated as a token.
- **oov\_token**: if given, it will be added to word\_index and used to replace out-of-vocabulary words during text\_to\_sequence calls

***Pad sequence***- Pads sequences to the same length. This function transforms a list of num\_samples sequences (lists of integers) into a 2D Numpy array of shape (num\_samples, num\_timesteps). num\_timesteps is either the maxlen argument if provided, or the length of the longest sequence otherwise.

Sequences that are shorter than num\_timesteps are padded with value at the end. Sequences longer than num\_timesteps are truncated so that they fit the desired length. The position where padding or truncation happens is determined by the arguments padding and truncating, respectively.

Converts a class vector (integers) to binary class matrix.  
E.g. for use with categorical\_crossentropy.

### Arguments

- **y**: class vector to be converted into a matrix (integers from 0 to num\_classes).
- **num\_classes**: total number of classes.
- **dtype**: The data type expected by the input, as a string (float32, float64, int32...)

### Returns

A binary matrix representation of the input. The classes axis is placed last.

**Keras.callbacks** (*EarlyStopping*)-A call back is a set of functions to be applied at given stages of the training procedure. You can use call-backs to get a view on internal states and statistics of the model during training. You can pass a list of call-backs (as the keyword argument call-backs) to the .fit() method of the Sequential or Model classes. The relevant methods of the call-backs will then be called at each stage of the training

A problem with training neural networks is in the choice of the [number of training epochs](#) to use. Too many epochs can lead to overfitting of the training dataset, whereas too few may result in an underfit model. Early stopping is a method that allows you to specify an arbitrary large number of training epochs and stop training once the model performance stops improving on a hold out validation dataset.

**EarlyStopping** -Stop training when a monitored quantity has stopped improving. Keras supports the early stopping of training via a callback called *EarlyStopping*.

This callback allows you to specify the performance measure to monitor, the trigger, and once triggered, it will stop the training process.

The *EarlyStopping* callback is configured when instantiated via arguments.

The “*monitor*” allows you to specify the performance measure to monitor in order to end training. Recall from the previous section that the calculation of measures on the validation dataset will have the ‘*val\_*’ prefix, such as ‘*val\_loss*’ for the loss on the validation dataset

```
es = EarlyStopping(monitor='val_loss')
```

Based on the choice of performance measure, the “*mode*” argument will need to be specified as whether the objective of the chosen metric is to increase (maximize or ‘*max*’) or to decrease (minimize or ‘*min*’).

**Word\_Tokenize()**- We use the method word\_tokenize() to split a sentence into words. The output of word tokenization can be converted to Data Frame for better text understanding in machine learning applications. It can also be provided as input for further text cleaning steps such as punctuation removal, numeric character removal or stemming. Machine learning models need numeric data to be trained and make a prediction. Word tokenization becomes a crucial part of the text (string) to numeric data conversion.

**Lemmatization**-Lemmatization is the process of converting a word to its base form. The difference between stemming and lemmatization is, lemmatization considers the context and converts the word to its meaningful base form, whereas stemming just removes the last few characters, often leading to incorrect meanings and spelling errors. For example, lemmatization would correctly identify the base form of ‘caring’ to ‘care’, whereas, stemming would cut off the ‘ing’ part and convert it to car.

‘Caring’ -> Lemmatization -> ‘Care’

‘Caring’ -> Stemming -> ‘Car’

**Wordnet** is a large, freely and publicly available lexical database for the English language aiming to establish structured semantic relationships between words. It offers lemmatization capabilities as well and is one of the earliest and most commonly used lemmatizers. NLTK offers an interface to it, but you have to download it first in order to use it.

**Punkt** Sentence Tokenizer- This tokenizer divides a text into a list of sentences, by using an unsupervised algorithm to build a model for abbreviation words, collocations, and words that start sentences. It must be trained on a large collection of plaintext in the target language before it can be used for example:

```
nltk. download("punkt") Download "punkt" tokenizer.  
sentence = "Think and wonder, wonder and think."  
words = nltk. word_tokenize(sentence)  
new_words= [word for word in words if word.isalnum()]  
print(new_words)
```

Output: ['Think', 'and', 'wonder', 'wonder', 'and', 'think'].

```
stopwords = nltk.corpus.stopwords.words('english')
```

This code is used to remove stop words from the essays in English language

#### list of stop words

```
{'ourselves', 'hers', 'between', 'yourself', 'but', 'again', 'there', 'about', 'once', 'during',  
'out', 'very', 'having', 'with', 'they', 'own', 'an', 'be', 'some', 'for', 'do', 'its', 'yours',  
'such', 'into', 'of', 'most', 'itself', 'other', 'off', 'is', 's', 'am', 'or', 'who', 'as', 'from',  
'him', 'each', 'the', 'themselves', 'until', 'below', 'are', 'we', 'these', 'your', 'his',  
'through', 'don', 'nor', 'me', 'were', 'her', 'more', 'himself', 'this', 'down', 'should',  
'our', 'their', 'while', 'above', 'both', 'up', 'to', 'ours', 'had', 'she', 'all', 'no', 'when',  
'at', 'any', 'before', 'them', 'same', 'and', 'been', 'have', 'in', 'will', 'on', 'does',  
'yourselves', 'then', 'that', 'because', 'what', 'over', 'why', 'so', 'can', 'did', 'not', 'now',  
'under', 'he', 'you', 'herself', 'has', 'just', 'where', 'too', 'only', 'myself', 'which',  
'those', 'i', 'after', 'few', 'whom', 't', 'being', 'if', 'theirs', 'my', 'against', 'a', 'by',  
'doing', 'it', 'how', 'further', 'was', 'here', 'than'}
```

#### RE-Regular Expression:

A **regular expression** is “instruction” given to a function on what and how to match or replace a set of strings. Let's start with some basics in **regular** expressions, that is some basic syntax you should know. Brackets [] are used to specify a disjunction of characters.

**Regular expressions** also called **regex**.

It is a very powerful programming tool that is used for a variety of purposes such as feature extraction from text, string replacement and other string manipulations. A regular expression is a set of characters, or a **pattern**, which is used to find sub strings in a given string. for ex. extracting all hashtags from a tweet, getting email id or phone numbers etc..from a large unstructured text content.

## Preprocessing

```

nltk.download('stopwords')
nltk.download('punkt')
nltk.download('wordnet')
stopwords = nltk.corpus.stopwords.words('english')

def clean_text(text):
    text=re.sub(r"^[a-zA-Z]", " ",text)
    text=text.lower()
    text=re.sub(r"i'm", "i am",text)
    text=re.sub(r"he's", "he is",text)
    text=re.sub(r"she's", "she is",text)
    text=re.sub(r"that's", "that is",text)
    text=re.sub(r"what's", "what is",text)
    text=re.sub(r"where's", "where is",text)
    text=re.sub(r"\ 'll", " will",text)
    text=re.sub(r"\ 've", " have",text)
    text=re.sub(r"\ 're", " are",text)
    text=re.sub(r"\ 'd", " would",text)
    text=re.sub(r"won't", "will not",text)
    text=re.sub(r"can't", "cannot",text)
    text=re.sub(r"[- () \\"#/@;:<>{}+=~|.?,]", "",text)

```

Figure 4.14: Pre-processing

“The computer runs on a three-step cycle namely input, process, and output. Also, the computer follows this cycle in every process it was asked to do. In simple words, the process can be explained in this way. The data which we feed into the computer is input, the work CPU do is process and the result which the computer give is output. The simple computer basically consists of CPU, monitor, mouse, and keyboard. Also, there are hundreds of other computer parts that can be attached to it. These other”

## Output –

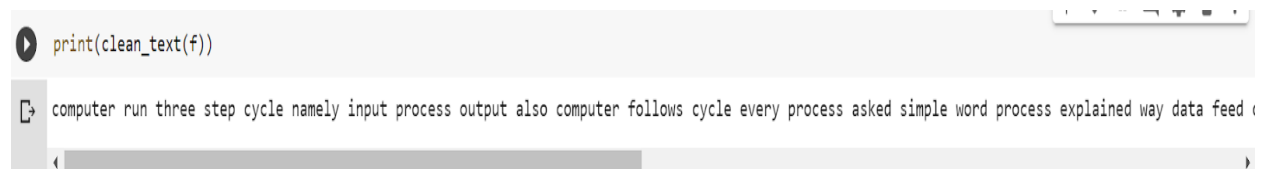


Figure 4.15: Output of clean text

Output looks like:

*'computer run three step cycle namely input process output also computer follows cycle every process asked simple word process explained way data feed computer input work cpu process result computer give output simple computer basically consists cpu monitor mouse keyboard also hundred computer part attached'*

Now let us have the look at the dataset

	ID	Essayset	min_score	max_score	score_1	score_2	score_3	score_4	score_5	clarity	coherent	EssayText
0	1	1.0	0	3	1	1	1.0	1.0	1.0	average	worst	additional information would need replicate ex...
1	3	1.0	0	3	1	1	1.0	1.0	1.5	worst	above_average	need trial control set exact amount vinegar po...
2	4	1.0	0	3	0	0	0.0	0.0	1.0	worst	worst	student list rock better rock worse procedure
3	5	1.0	0	3	2	2	2.0	2.5	1.0	above_average	worst	student able make replicate would need tell us...
4	6	1.0	0	3	1	0	0.0	0.0	0.0	worst	worst	would need information would let different sam...

Figure 4.16: Snapshot of Dataset

As you can see there are 12 columns and 16,000 rows.

1. **ID** - Unique ID given to each essay.
2. **Essay set** - There are 10 different essay sets i.e. 10 different topic.
3. **Min\_score** - the minimum score i.e. 0
4. **Max\_score** - the maxim score i.e. 1
5. **Score\_1, Score\_2 ..... Score\_5** - the essays are judged by 5 different people
6. **Clarity**- Essays are judged on the basis of clarity.
7. **Coherence** - Also, they are judged based on coherence.

**Essaytext** - the dataset consists of 16000 essays on 10 different topics.

In the next step we're taking the average of the human grades and we will put this average score in the model to train the dataset.

```
train['avg_score']=np.round((train.score_1+train.score_2+train.score_3)/3)
```

ID	Essayset	min_score	max_score	clarity	coherent	EssayText	avg_score
1	1.0	0	3	average	worst	additional information would need replicate ex...	1.0
3	1.0	0	3	worst	above_average	need trial control set exact amount vinegar po...	1.0
4	1.0	0	3	worst	worst	student list rock better rock worse procedure	0.0
5	1.0	0	3	above_average	worst	student able make replicate would need tell us...	2.0
6	1.0	0	3	worst	worst	would need information would let different sam...	0.0

Figure 4.17: Average Score of human grades



We have defined maximum length as 2500 i.e. system can take essays with input up to 2500 words. In this section we try to divide the whole essay into tokens. After generating the tokens, A matrix known as a sequence matrix is generated to be fed in the model.

```
max_words = 2500
max_len = 50
tok = Tokenizer(num_words=max_words)
tok.fit_on_texts(train2.EssayText)
sequences = tok.texts_to_sequences(train2.EssayText)
sequences_matrix = sequence.pad_sequences(sequences,maxlen=max_len)
Y1=train2.avg_score
```

Figure 4.18: Sequence Matrix

The sequence matrix is generated.

```
[[ 0  0  0 ... 26 415 52]
 [ 0  0  0 ... 439 785 316]
 [ 0  0  0 ... 734 1756 121]
 ...
 [ 0  0  0 ... 267 181 130]
 [ 0  0  0 ... 267 65 58]
 [ 0  0  0 ... 399 43 82]]
```

Figure 4.19: Output of Sequence matrix

## Splitting the data

Here, we split the data into training part and testing part. test size is 0.2 which deities that 20% of data will be used to test the accuracy of the model.

Now, in the training part we put the tokenized matrix on x axis and on y axis we put the average score. italicized text

Now, we will define the model

## Model Definition

```
def RNN():
    inputs = Input(name='inputs', shape=[max_len])
    layer = Embedding(max_words,50,input_length=max_len)(inputs)
    layer = Bidirectional(LSTM(64))(layer)
    layer = Dense(64,name='FC1')(layer)
    layer = Activation('relu')(layer)
    layer = Dropout(0.5)(layer)
    layer = Dense(64,name='FC2')(layer)
    layer = Activation('relu')(layer)
    layer = Dropout(0.5)(layer)
    layer = Dense(4,name='out_layer')(layer)
    layer = Activation('softmax')(layer)
    model = Model(inputs=inputs,outputs=layer)
    return model
```

Figure 4.20: Model Definition

**Model Definition:**

The input layer consists of an embedding layer, bidirectional lstm.

**Embedding layer-** The Embedding layer is used to create word vectors for incoming words. It sits between the input and the LSTM layer, i.e. the output of the Embedding layer is the input to the LSTM layer

**Bidirectional lstm-** It involves duplicating the first recurrent layer in the network so that there are now two layers side-by-side, then providing the input sequence as-is as input to the first layer and providing a reversed copy of the input sequence to the second. The idea is to split the state neurons of a regular RNN in a part that is responsible for the positive time direction (forward states) and a part for the negative time direction (backward states)

**FC1 and FC2:**

**Dense layer** - A fully connected layer that often follows LSTM layers and is used for outputting a prediction is called Dense ().

**Activation-** Activation functions determine the output of a deep learning model, its accuracy, and also the computational efficiency of training a model—which can make or break a large-scale neural network

**Relu-** The purpose of the Rectified Linear Activation Function (or ReLU for short) is to allow the neural network to learn nonlinear dependencies. Specifically, the way this works is that ReLU will return input directly if the value is greater than 0. If less than 0, then 0.0 is simply returned

**Softmax-** Softmax is an activation function which converts its inputs – likely the logits, a.k.a. the outputs of the last layer of your neural network when no activation function is applied yet – into a discrete probability distribution over the target classes.

**Dropout-** Dropout is a regularization method where input and recurrent connections to LSTM units are probabilistically excluded from activation and weight updates while training a network. This has the effect of reducing overfitting and improving model performance.

```

model = RNN()
model.compile(loss='categorical_crossentropy',optimizer=RMSprop(),metrics=
['accuracy'])
model.fit(X_train,Y_train,batch_size=64,epochs=2,validation_data=(X_test,Y
_test))

```

Figure 4.21: Model Fitting

## Model Fitting

**Categorical cross entropy** is a loss function that is used for single label categorization. This is when only one category is applicable for each data point. In other words, an example can belong to one class only. The block before the Target block must use the activation function SoftMax.

**Optimizer**-An optimizer is one of the two arguments required for compiling a Keras model.

**RMSProp optimizer** --The RMSprop optimizer restricts the oscillations in the vertical direction. Therefore, we can increase our learning rate and our algorithm could take larger steps in the horizontal direction converging faster. The difference between RMSprop and gradient descent is on how the gradients are calculated.

**Metrics**- The Keras library provides a way to calculate and report on a suite of standard metrics when training deep learning models.

In addition to offering standard metrics for classification and regression problems, Keras also allows you to define and report on your own custom metrics when training deep learning models. This is particularly useful if you want to keep track of a performance measure that better captures the skill of your model during training.

### **Keras Classification Metrics.**

- **Binary Accuracy:** binary\_accuracy, acc
- **Categorical Accuracy:** categorical\_accuracy, acc
- **Sparse Categorical Accuracy:** sparse\_categorical\_accuracy
- **Top k Categorical Accuracy:** top\_k\_categorical\_accuracy (requires you specify a k parameter)
- **Sparse Top k Categorical Accuracy:** sparse\_top\_k\_categorical\_accuracy (requires you specify a k parameter)

**Model prediction -**

```

y_pred_train=model.predict(sequences_matrix)
ttl=pd.DataFrame(y_pred_train,columns=['out1','out2','out3','out4'])
train4=pd.concat([train3,ttl],axis=1)
train4.head()

```

	Essayset	clarity	coherent	avg_score	out1	out2	out3	out4
0	1.0	average	worst	1.0	0.068568	0.192579	0.370063	0.368791
1	1.0	worst	above_average	1.0	0.540235	0.447043	0.012596	0.000126
2	1.0	worst	worst	0.0	0.888665	0.110655	0.000677	0.000003
3	1.0	above_average	worst	2.0	0.215757	0.347087	0.312097	0.125060
4	1.0	worst	worst	0.0	0.918013	0.080249	0.001716	0.000022

Figure 4.22: Model Prediction

**Model Prediction:**

A recurrent neural network (RNN) is a class of artificial neural network where connections between units form a directed graph along a sequence. This allows it to exhibit dynamic temporal behaviour for a time sequence. Unlike feedforward neural networks, RNNs can use their internal state (memory) to process sequences of inputs. This makes them applicable to tasks such as unsegmented, connected handwriting recognition or speech recognition. Recurrent Neural Network comes into the picture when any model needs context to be able to provide the output based on the input. Sometimes the context is the single most important thing for the model to predict the most appropriate output. Let's understand this by an analogy. Suppose you are watching a movie, you keep watching the movie as at any point in time, you have the context because you have seen the movie until that point, then only you are able to relate everything correctly. It means that you remember everything that you have watched. Similarly, RNN remembers everything. In other neural networks, all the inputs are independent of each other. But in RNN, all the inputs are related to each other. Let's say you have to predict the next word in a given sentence, in that case, the relation among all the previous words helps in predicting the better output. The RNN remembers all these relations while training itself. In order to achieve it, the RNN creates the networks with loops in them, which allows it to persist the information.

## XGBOOST CLASSIFICATION

```

from xgboost import XGBClassifier
xgb=XGBClassifier(colsample_bytree=0.4, gamma=0,
  learning_rate=0.01,
  max_depth=4,
  min_child_weight=0.5,
  n_estimators=10000,
  reg_alpha=0.75,
  reg_lambda=0.45,
  subsample=0.6,
  seed=42)

```

## Output

```

out.to_csv('SCORES.csv',index=None)
nn=pd.read_csv('SCORES.csv')

```

Figure 4.23: Snapshot of XGBoost classifier

XGBoost is a decision-tree-based ensemble Machine Learning algorithm that uses a gradient boosting framework. In prediction problems involving unstructured data (images, text, etc.) artificial neural networks tend to outperform all other algorithms or frameworks. However, when it comes to small-to-medium structured/tabular data, decision tree-based algorithms are considered best-in-class right now. Please see the chart below for the evolution of tree-based algorithms over the years.



Figure 4.24: Advantages of XGBoost

```
accuracy_score(Y1_test,yy_test)
```

```
0.7935085007727976
```

```
print(classification_report(Y1_test,yy_test))
```

	precision	recall	f1-score	support
0.0	0.84	0.87	0.86	1309
1.0	0.75	0.71	0.73	1053
2.0	0.80	0.82	0.81	749
3.0	0.56	0.48	0.52	124
accuracy			0.79	3235
macro avg	0.74	0.72	0.73	3235
weighted avg	0.79	0.79	0.79	3235

Figure 4.25: Classification Report

The output is generated in a csv file. The essay are graded and is seen in essay score with respective essay set and id.

```
[76] out.to_csv('SCORES.csv',index=None)
nn=pd.read_csv('SCORES.csv')
```

```
nn.head(10)
```

	id	essay_set	essay_score
0	1	1	1.0
1	3	1	1.0
2	4	1	0.0
3	5	1	2.0
4	6	1	0.0
5	7	1	0.0
6	8	1	3.0
7	9	1	3.0
8	10	1	2.0

Figure 4.26: Snapshot of output csv file

## **Chapter 5**

### **Result and Analysis**

#### **5.1 Result**

Data set was divided into train and test. Training set was used to train the model. And the testing set was used for prediction which gave prediction accuracy of 69. Now the manual input essay is fed to the system and score is generated. The essays that are input using the website are graded and score is stored in a .csv file. Testing And manual both the essays were graded with 69 Percent and 78 Percent of Accuracy. The essays are graded from 0 to 3 where 0 being the lowest and 3 being the highest rate. , the dataset is split into Train and Test Sets in a ratio of 0.7 i.e. 70% data for training and 30% for testing purpose .Category encoders are used to convert the categorical value of clarity and coherence columns into numeric value which fits into the model. Testing of dataset is done by dropping out various column for rigorous training. Different attributes are considered at a time for proper training and testing. Based on these training and testing is done and score with highest accuracy is predicted.

#### **5.2 Output**

Output for seen data -

```
out.to_csv('UNSEEN_SCORES.csv',index=None)
nn=pd.read_csv('UNSEEN_SCORES.csv')
```

```
nn.head()
```

	id	essay_set	essay_score
0	1	10	2.0
1	13	11	2.0

Figure 5.1: Output of Manually inserted Data

For prediction of score an input essay was taken from a website created for the same. These essays were stored in a csv file by the website. These csv files are fed to the system for predicting score of the essay. The csv file contains the fields 'ID','EssaySet'.EssayText. To check accuracy of the system we input two essays from the website one essay was completely written and the same essay with missing parameters and non-English words was fed. The first essay was graded as 2 and the modified essay was graded as 0.

## Enter your details

Thank you

Enter Essayset

Enter essay

Submit

Figure 5.2: Snapshot of system at end-user

This a snapshot of the website we created for taking essay input. These was made using php And for which xampp server is required. The fields in the website are Essayset and essay. Essayset does not contribute to the grading. The essay once written and submitted gets saved in a csv file .These csv file needs to be uploaded in to code for grading purpose.



	A	B	C
1	ID	Essayset	EssayText
2	1	10	computer is a device that accepts the message by the imputer and processes this message and stores the i
3	13	11	computer was related to a person who carries out calculations or computations and as such the word comp
4			
5			
6			

Figure 5.3: Input Essays

The essay which are input through website are stored in the csv above is the screenshot of same.

	precision	recall	f1-score	support
0.0	0.80	0.82	0.81	1309
1.0	0.63	0.61	0.62	1053
2.0	0.63	0.65	0.64	749
3.0	0.34	0.30	0.32	124
accuracy			0.69	3235
macro avg	0.60	0.59	0.60	3235
weighted avg	0.69	0.69	0.69	3235

Figure 5.4: Classifier Report

```
train4.head()
```

	Essayset	score_1	score_2	score_3	score_4	score_5	clarity	coherent	avg_score	out1	out2	out3	out4
0	1.0	1	1	1.0	1.0	1.0	average	worst	1.0	0.069076	0.280814	0.589117	0.060993
1	1.0	1	1	1.0	1.0	1.5	worst	above_average	1.0	0.566896	0.340163	0.079751	0.013190
2	1.0	0	0	0.0	0.0	1.0	worst	worst	0.0	0.882725	0.110984	0.005769	0.000522
3	1.0	2	2	2.0	2.5	1.0	above_average	worst	2.0	0.121137	0.437128	0.417713	0.024022
4	1.0	1	0	0.0	0.0	0.0	worst	worst	0.0	0.781279	0.191935	0.023125	0.003661

Figure 5.5: Before Target Encoding

	Essayset	score_1	score_2	score_3	score_4	score_5	clarity	coherent	avg_score	out1	out2	out3	out4
0	1.480649	1	1	1.0	1.0	1.0	0.496662	0.482423	1.0	0.069076	0.280814	0.589117	0.060993
1	1.480649	1	1	1.0	1.0	1.5	0.486640	1.944411	1.0	0.566896	0.340163	0.079751	0.013190
2	1.480649	0	0	0.0	0.0	1.0	0.486640	0.482423	0.0	0.882725	0.110984	0.005769	0.000522
3	1.480649	2	2	2.0	2.5	1.0	1.943451	0.482423	2.0	0.121137	0.437128	0.417713	0.024022
4	1.480649	1	0	0.0	0.0	0.0	0.486640	0.482423	0.0	0.781279	0.191935	0.023125	0.003661

Figure 5.6: Output After Target Encoding



Figure 5.7: Number of Essays VS Essay Scores (Variation in grades)

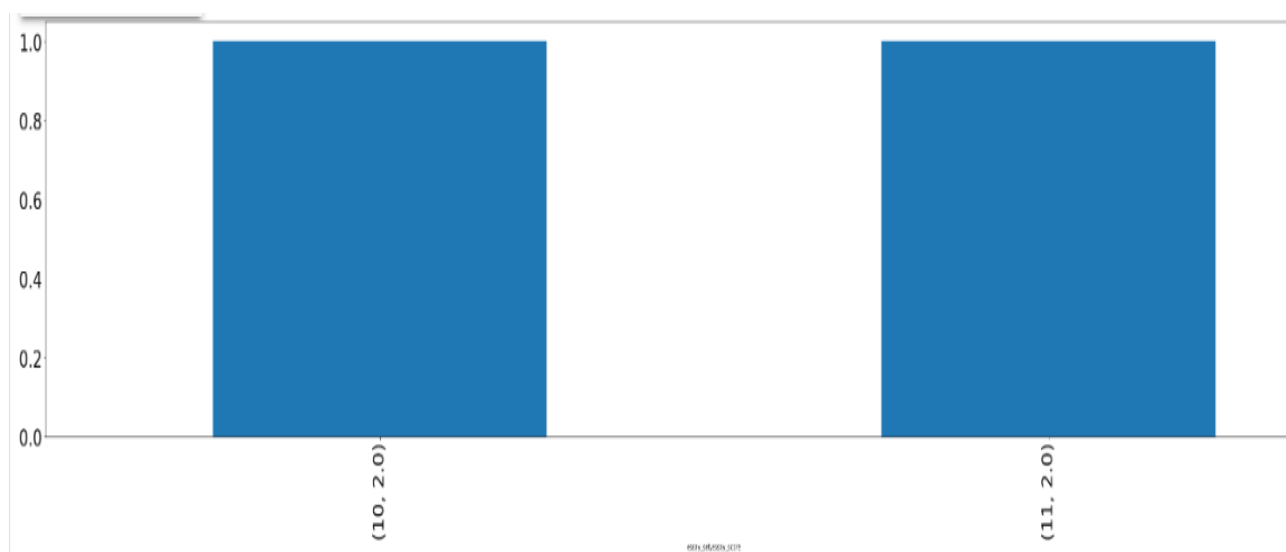


Figure 5.8: Output of manually inserted data

The two bars in the above diagram represent the score of newly inserted data as seen in figure 5.1 both the essays are graded as 2 hence in the plot two plots are displayed both of same height. The values on x-axis represent essayset and essay score.

## Chapter 6

### Conclusion and Future Scope

#### 6.1 Conclusion

A Software solution which when provided an essay grades it in between 0 to 3 where 0 being the lowest and 3 being the highest grade. It also provides the feedback where Non-English words are highlighted. As we have hypothesized, removing the non-English word contributes towards a Good prediction. Our Final model uses bidirectional LSTM in order to produce a good result. Using Decision tree classifier, the network tends to outperform, to deal with these we used XGBoost Classifier which helped to avoid over fitting problem. Our model works relatively better on non-context specific essays. Performance on context specific and richer essays can be improved by incorporating content and advanced NLP features.

#### 6.2 Future Scope

Even Though our project is using clarity and coherence as factors to judge further it can be extended to judging essays Grammatically, Usage Based(use of preposition , word usage),choice of word, Mechanics, etc. Automated Grammar checker could be added which will instantly find and report over 250 types of grammatical mistakes. Plagiarism Detection could be introduced which compare essay text to billions of web pages and major content databases. Hence Reduce chances of copying. We could also further improve the model by using more complex features. This would be particularly constructive for context specific essays in the data set.

# Appendix - I

## Installation Procedure - Development Software

### Installing Anaconda

1. Go to the following link: [Anaconda.com/downloads](https://anaconda.com/downloads)
2. Select operating system you want where the three operating systems are listed.
3. Download the most recent Python 3 release. At the time of writing, the most recent release was the Python 3.6 Version. Python 2.7 is legacy Python. For problem solvers, select the Python 3.6 version. If you are unsure if your computer is running a 64-bit or 32-bit version of Windows, select 64-bit as 64-bit Windows is most common.
4. may be prompted to enter your email. You can still download Anaconda if you click [No Thanks] and don't enter your Work Email address. The download is quite large (over 500 MB) so it may take a while to for Anaconda to download.
5. Once the download completes, open and run the .exe installer. At the beginning of the install, you need to click Next to confirm the installation. Then agree to the license.
6. the Advanced Installation Options screen, I recommend that you do not check" Add Anaconda to my PATH environment variable"
7. After the installation of Anaconda is complete, you can go to the Windows start menu and select the Anaconda Prompt.
8. This opens the Anaconda Prompt. Anaconda is the Python distribution and the Anaconda Prompt is a command line shell (a program where you type in commands instead of using a mouse). The black screen and text that makes up the Anaconda Prompt doesn't look like much, but it is really helpful for problem solvers using Python.

## References:

- [1] Siddhartha Ghosh ; Sameen S. Fatima, “Design of an Automated Essay Grading (AEG) System in Indian Context”, 27 January 2009.
- [2] Y.Harika,I.Sri Latha , V.Lohith Sai ,P.Sai Krishna,M.Suneetha NLP , “AUTOMATED ESSAY GRADING USING FEATURES SELECTION”.
- [3] Abeer Zaroor, Mohammed Maree Muath , Pushpak Bhattacharyya, “ASAP++: Enriching the ASAP Automated Essay Grading Dataset with Essay Attribute Scores”
- [4] Kaveh taghipoor , Hwee tou Ng, D. Klieger, M.(2015),”A neural network approach to automated essay scoring”.
- [5] J burstein, J. Tetreault, (2013), “The e-rater R Automated Essay Scoring System, Handbook of Automated Essay Scoring: Current Applications and Future Directions”
- [6] Attali, Y. and Burstein, J.: Automated Essay Scoring with e-rater V. 2.In: The Journal of Technology, Learning and Assessment. 4(3), 2006, p.3–29.
- [7] Elliot, S: IntelliMetric: From Here to Validity. In: AUTOMATED ESSAY SCORING: A CROSS- DISCIPLINARY PERSPECTIVE. 2003, pp. 71-86
- [8] Landauer, T., Laham, D. and Foltz, P.: THE INTELLIGENT ESSAY ASSESSOR. IEEE INTELLIGENT SYSTEMS.15 (5). 2000, p. 27–31

## Acknowledgements

We have taken efforts in this project however it would not have been possible without the kind support and help of many individuals. We would like to extend our sincere thanks to all of them.

We would like to express our gratitude towards our project Coordinator Prof. Sunantha Krishnan, Head of the Department IT, Prof. Prasad and Principal of DBIT, Dr. Prasanna Nambiar for their kind co-operation and encouragement which helped us in completion of this project

( \_\_\_\_\_ )  
(Priyanka Kedar 36)

( \_\_\_\_\_ )  
(Neha Mishra 47)

( \_\_\_\_\_ )  
(Sarthak Gupta 26)

**Date:**   /   /