Group - 9

Mini Project – 1

Predicting Human Reading Behavior

Project Report

Project Supervisor : Dr . Sunil Saumya

Group Members :

Sumith Sai Budde , 18BCS101

Syed Sufyan Ahmed , 18BCS103

MP Bharath , 18BCS057

Chandhan P , 18BCS063

Indian Institute of Information Technology Dharwad

Computer Science and Engineering

CERTIFICATE

We hereby certify that the work which is being presented in the Mini Project 1 report entitled "Predicting Human Reading Behavior" in partial fulfillment of the requirements for the award of B.Tech degree and submitted to the Department of Computer Science and Engineering of Indian Institute of Information Technology Dharwad , Karnataka is an authentic record of our own work carried out during the period from January 2021 to April 2021 under the supervision of Dr. Sunil Saumya , Asst.Professor, IIIT Dharwad.

The matter proposed in this report has not been published earlier and has never been submitted by us for the award of any other degree elsewhere.

Name of the Students :

Sumith Sai Budde	18BCS101
Syed Sufyan Ahmed	18BCS103
MP Bharath	18BCS057
P Chandhan	18BCS063

Dr.Sunil Saumya	Dr. B.Jayalaxmi	Dr. Arun Chauhan
Project Supervisor	Project Coordinator	Head of the Department

Table of Contents :

1. Abstract :

Eye movement data during reading is a useful source of information for understanding language comprehension processes. The ability of accurately modeling eye-tracking features is crucial to advance the understanding of language processing. Eye tracking provides millisecond-accurate records on where humans look when they are reading, shedding lights on where humans pay attention during their reading and comprehension phase. The task is to predict eye tracking-based metrics recorded during English sentence processing. In this project we present five models, two are the conventional Machine Learning Models – Linear Regression and Ridge Regression and the other three models are Deep Learning Models - LSTM, BiLSTM and RoBERTa for solving the task.

2. Introduction :

Human reading behaviour is not uniform. It varies from person to person. The reason that eye fixation data provide a rich base for a theoretical model of language processing is that reader's pauses on various words of a text are distinctly non-uniform. Some words are looked at very briefly, while others are gazed at for one or two seconds. And also each person may fixate different amount of time on a particular word. The longer pauses are associated with a need for more computation.

Reading is complex cognitive behavior and the underlying cognitive process occurs only in the brain. Modeling such behavior requires obtaining some explicit indicators via methods such as eye tracking.

When reading a text, the eyes of a skilled reader do not move continuously over the lines of text. Instead, reading proceeds by alternating between fixations and rapid eye movements called saccades. This behavior is determined by the physiological structure of the human retina. Most of the optic nerve cells are concentrated in the fovea and only when the visual image falls in this area can it be "seen" clearly. Unfortunately, the fovea only provides about a 5- degree field of view. Therefore, the reader needs to change the fixation point through successive saccades so that the next content falls on the fovea region of the retina.

By analyzing eye movements during reading, we can quantify the reader's actions and model for reading. Eye tracking helps researchers to determine where and how many times subjects focus on a certain word, along with their eye movement sequences from one word to another. The ability of accurately modeling eye-tracking features is crucial to advance the understanding of language processing. The benefits of utilizing eye movement data have been noticed in various domains, including natural language processing and computer vision. Eye tracking data can also be used in neuro science research , gaining inferences on task specific reading , normal reading and language understanding.

This type of eye tracking data has recently been leveraged for uses in natural language processing: it can improve performance on a variety of downstream tasks, such as part-of-speech tagging (Barrett et al., 2016), dependency parsing (Strzyz et al., 2019), and for cognitively-inspired evaluation methods for word embeddings (Søgaard, 2016) etc.

Although these eye movement models typically have parameters that are fit to empirical data, their predictions are tested on unseen data. Moreover, their predictions are usually averaged over a group of readers, while eye movement patterns vary significantly between individuals. Predicting the actual eye movements that an individual will make while reading a new text is arguably a challenging problem.

This Project aims to add to our understanding of how language models can relate to eye movement features.

3.Methodology :

3.1. Dataset Description :

We will use the eye-tracking data of the Zurich Cognitive Language Processing Corpus (ZuCo 1.0 and ZuCo 2.0) recorded during normal reading. The training data will contain 800 sentences, and the test set 191 sentences. The data provided will contain scaled features in the range between 0 and 100 to facilitate evaluation via the mean absolute error (MAE). The features nFix and fixProp are scaled separately, while FFD, GPT and TRT are scaled together since these are all dependent and measured in milliseconds. The features are averaged across all readers. The data was scaled and randomly shuffled before splitting into training and test data.

A prominent feature of this dataset is the personal reading speed. The sentences were presented to the subjects in a naturalistic reading scenario, where the complete sentence is presented on the screen and the subjects read each sentence at their own speed, i.e. the reader determines him/herself for how long each word is fixated and which word to fixate next.

The Dataset contains 8 columns : sentence_id , word_id , word , nFix , FFD , GPT , TRT , fixProp. sentence_id and word_id are the id's of sentences and words respectively. End of a sentence is given by <EOS> symbol meaning end of the sentence. (Table.1)

Features:

1. number of fixations (nFix): Total number of fixations on the current word.
2. first fixation duration (FFD): The duration of the first fixation on that particular word.
3. total reading time (TRT): The sum of all fixation durations on the current word.
4. go-past time (GPT): The sum of all fixations prior to progressing to the right of the current word.
5. fixation proportion (fixProp): The proportion of participants that fixated the current word (proxy for how likely a word is to be fixated).

For a detailed description of the dataset please refer to the original publications (Hollenstein et al., 2018 and Hollenstein et al., 2020).

	sentence_id	word_id	word	nFix	FFD	GPT	TRT	fixProp
0	0	0	Carlucci	28.39757	4.642973	6.190631	10.34359	94.11765
1	0	1	was	12.98174	3.534385	5.263977	4.565348	76.47059
2	0	2	deputy	25.15213	5.809781	9.926357	9.145251	100.0000
3	0	4	defense	20.28398	5.261551	8.092455	8.359293	88.23529
…	…	…	…	…	…	…	…	…
15732	799	5	patient	23.75478	5.230420	7.826157	8.412660	100.0000
15733	799	6	wife	18.3908	3.674811	8.61198	6.669187	77.77778
15734	799	7	of	14.55939	4.380448	11.89044	6.114758	77.77778
15735	799	8	Homer.<EOS>	2.298851	0.882046	2.978338	0.882046	16.66667

Table 1. Training Dataset

3.2. Data Preprocessing for Conventional Machine Learning Model :

We are using an encoding technique called TF-IDF(Term Frequency - Inverse Data Frequency) encoding for giving input to the model.

Term Frequency: Is the occurrence of the current word in the current sentence w.r.t the total number of words in the current sentence.

$$tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{i,j}}$$

Inverse Data Freqency: Log of Total number of words in the whole data corpus w.r.t the total number of sentences containing the current word.

$$idf(w) = log(\frac{N}{df_t})$$

Example :

Let's say we have 5 sentences : ["this is good phone" , "this is bad mobile" , "she is good cat" , "he has bad temper" , "this mobile phone is not good"]

Data Corpus: ["bad" , "cat" , "good" , "has" , "he" , "is" , "mobile" , "not" , "phone" , "she" , "temper" , "this"]

TF("this") : "this" in sentence1 : Number of "this" word in sentence1 / total number of words in sentence1

IDF("this") : log(total number of words in the whole data corpus / total number of sentences having "this" word)

TF : 1 / 4 = 0.25 , IDF : loge(12 / 3) = 1.38629

TF-IDF = TF*IDF = 0.25*1.38629 = 0.3465725

So we associate "this" in sentence 1: 0.3465725 ; similarly we can find out TF-IDF for every word in that sentence.

For this we are using a function called TfidfVectorizer() from sklearn module and we convert a collection of raw documents to a matrix of TF-IDF features like below:

```
[[0.         0.         0.49035258 0.         0.         0.41250005
 0.         0.         0.59072194 0.         0.         0.3465725]
[0.56106597 0.         0.         0.         0.         0.39179133
 0.56106597 0.         0.         0.         0.         0.3465725]
[0.         0.60128545 0.40268787 0.         0.         0.33875373
 0.         0.         0.         0.60128545 0.         0.        ]
[0.42224214 0.         0.         0.52335825 0.52335825 0.
 0.         0.         0.         0.         0.52335825 0.        ]
[0.         0.         0.35714732 0.         0.         0.30044359
 0.43025115 0.53328521 0.43025115 0.         0.         0.35714732]]
```

3.3. Data Preprocessing for Deep Learning Models :

a. Cleaning and Forming Sentences:

The data is given in the form of words and some words have <EOS> tags in them for marking sentence endings. Hence we use those tags to know sentence endings and concatenate only the words by removing <EOS> tags to form sentences.

First 5 sentences in the data :

['Carlucci was deputy defense secretary from 1981 until 1986, national security advisor from 1986 until 1987, and defense secretary in 1987, following the resignation of Weinberger, his no mination by President Ronald Reagan and his confirmation in the Senate by a vote of 91 to 1. ', 'All m embers of the Bouvier family (except, of course, Marge) smoke heavily and have gruff voices and sar castic, snarky demeanours. ', 'Genuinely touching because it's realistic about all kinds of love. ' , 'Shire became famous for her roles as the wife of boxer Rocky Balboa in the Rocky movies, and for her role in The Godfather and its sequels. ', 'As a child, his hero was Batman, and as a teenager his interests shi fted towards music.']

b. Make a list of List Features:

We make a list of list features for each word and then append that to a list so as to have a list of sentence features.

List of Features for sentence 1:

[[[28.39756592, 4.642973379, 6.190631172, 10.34359378, 94.11764706], [12.98174442, 3.534384646, 5.263977133, 4.5653 4791, 76.47058824], [25.15212982, 5.809781213, 9.926356879, 9.145250595, 100.0], [20.28397566, 5.261551337, 8.092455168, 8.359292 719, 88.23529412], [17.84989858, 4.135982033, 4.507128808, 5.686065622, 88.23529412], [12.98174442, 4.429503339, 5.169371092, 5.2 9551248, 82.35294118], [17.03853955, 5.652104479, 11.94461908, 7.105156231, 94.11764706], [11.35902637, 3.216605382, 3.96860211 5, 3.786667421, 64.70588235], [15.4158215, 3.738151503, 7.680069863, 7.228871823, 70.58823529], [19.47261663, 3.694487177, 8.6431 10841, 6.894111987, 82.35294118], [20.28397566, 4.252420237, 6.098450927, 7.31377468, 94.11764706], [13.79310345, 3.672655013, 4. 296084563, 4.689063501, 76.47058824], [8.92494929, 2.750852566, 2.750852566, 2.750852566, 64.70588235], [10.54766734, 5.3343252 14, 5.581756398, 5.994141703, 64.70588235], [6.490872211, 2.282673955, 3.19962481, 2.782387913, 41.17647059], [12.1703854, 4.3324 71502, 6.581184314, 5.394970112, 70.58823529], [8.92494929, 2.767833138, 2.93278726, 2.93278726, 58.82352941], [14.60446247, 4.41 2522767, 5.606014357, 5.606014357, 76.47058824], [12.1703854, 3.017690117, 4.135982033, 3.604732728, 70.58823529], [2.434077079, 0.645261713, 0.645261713, 0.645261713, 17.64705882], [13.79310345, 4.225736482, 5.44348603, 5.826761785, 70.58823529], [16.22718 053, 4.247568645, 6.003844887, 5.377989541, 88.23529412], [2.434077079, 0.528823509, 0.528823509, 0.528823509, 17.64705882], [21. 90669371, 4.504703012, 7.667940883, 7.532096312, 94.11764706], [3.245436105, 1.030963263, 1.030963263, 1.030963263, 23.5294117 6], [17.03853955, 4.834611256, 6.341030519, 6.341030519, 88.23529412], [5.679513185, 2.093461874, 2.093461874, 2.093461874, 41.17 647059], [18.66125761, 4.407671175, 6.404101212, 6.70247411, 82.35294118], [3.245436105, 0.693777631, 0.999427916, 0.999427916, 17.64705882], [12.98174442, 4.024395421, 5.137835745, 5.251848153, 70.58823529], [14.60446247, 3.582900565, 4.582328481, 4.37613 5828, 82.35294118], [11.35902637, 2.784813709, 2.995857953, 2.995857953, 76.47058824], [4.868154158, 1.67379918, 1.67379918, 1.67 379918, 35.29411765], [11.35902637, 3.214179586, 4.48772244, 3.47859134, 70.58823529], [21.90669371, 4.395542196, 6.365288478, 7. 15367215, 100.0], [0.811359026, 0.143121959, 0.143121959, 0.143121959, 5.882352941], [8.92494929, 2.971599994, 4.577476889, 3.250 566524, 58.82352941], [19.47261663, 3.951621543, 7.735863169, 6.037806029, 94.11764706], [4.868154158, 1.34146514, 3.628990687, 1.34146514, 35.29411765], [8.113590264, 3.269972892, 4.351877869, 3.49072032, 52.94117647], [12.98174442, 3.614435911, 7.5806122 3, 4.279103992, 76.47058824], [5.679513185, 1.892120813, 6.79708015, 1.892120813, 41.17647059], [10.54766734, 4.388264808, 7.1439 68966, 5.317344643, 64.70588235], [10.54766734, 2.889122933, 11.36727965, 3.842460727, 58.82352941], [3.245436105, 1.171659426, 8.485434106, 1.171659426, 23.52941176]]]

c. Text to Sequences:

We cannot feed raw text directly into deep learning models. Text data must be encoded as numbers to be used as input or output for machine learning and deep learning models. Hence we have converted text to sequences.

Sequence of Sentence 1 :

[1674, 7, 1675, 355, 161, 16, 927, 82, 356, 357, 928, 1676, 16, 356, 82, 603, 2, 355, 161, 5, 603, 97, 1, 929, 3, 446, 9, 604, 20, 32, 930, 189, 2, 9, 1677, 5, 1, 190, 20, 4, 447, 3, 1678, 6, 299]

d. Padding :

Padding is used to ensure that all sequences in a list have the same length. By default this is done by padding 0 in the end of each sequence until each sequence has the same length as the longest sequence. List of Features is also padded in the same way.

Padded Sequence for sentence 1 :

[1674, 7, 1675, 355, 161, 16, 927, 82, 356, 357, 928, 1676, 16, 356, 82, 603, 2, 355, 161, 5, 603, 97, 1, 929, 3, 446, 9, 604, 20, 32, 930, 189, 2, 9, 1677, 5, 1, 190, 20, 4, 447, 3, 1678, 6, 299, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 0, 0, 0, 0, 0]

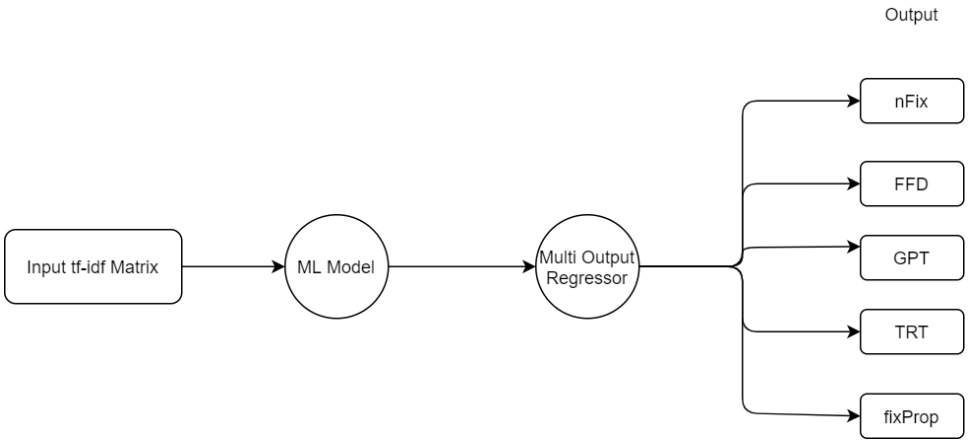
Padded Features for sentence 1 :

[[[28.39756592, 4.642973379, 6.190631172, 10.34359378, 94.11764706], [12.98174442, 3.534384646, 5.263977133, 4.5
6534791, 76.47058824], [25.15212982, 5.809781213, 9.926356879, 9.145250595, 100.0], [20.28397566, 5.261551337, 8.092455168, 8.359
292719, 88.23529412], [17.84989858, 4.135982033, 4.507128808, 5.686065622, 88.23529412], [12.98174442, 4.429503339, 5.169371092,
5.29551248, 82.35294118], [17.03853955, 5.652104479, 11.94461908, 7.105156231, 94.11764706], [11.35902637, 3.216605382, 3.968602
115, 3.786667421, 64.70588235], [15.4158215, 3.738151503, 7.680069863, 7.228871823, 70.58823529], [19.47261663, 3.694487177, 8.64
3110841, 6.894111987, 82.35294118], [20.28397566, 4.252420237, 6.098450927, 7.31377468, 94.11764706], [13.79310345, 3.672655013,
4.296084563, 4.689063501, 76.47058824], [8.92494929, 2.750852566, 2.750852566, 2.750852566, 64.70588235], [10.54766734, 5.334325
214, 5.581756398, 5.994141703, 64.70588235], [6.490872211, 2.282673955, 3.19962481, 2.782387913, 41.17647059], [12.1703854, 4.332
471502, 6.581184314, 5.394970112, 70.58823529], [8.92494929, 2.767833138, 2.93278726, 2.93278726, 58.82352941], [14.60446247, 4.4
12522767, 5.606014357, 5.606014357, 76.47058824], [12.1703854, 3.017690117, 4.135982033, 3.604732728, 70.58823529], [2.43407707
9, 0.645261713, 0.645261713, 0.645261713, 17.64705882], [13.79310345, 4.225736482, 5.44348603, 5.826761785, 70.58823529], [16.227
18053, 4.247568645, 6.003844887, 5.377989541, 88.23529412], [2.434077079, 0.528823509, 0.528823509, 0.528823509, 17.64705882], [
21.90669371, 4.504703012, 7.667940883, 7.532096312, 94.11764706], [3.245436105, 1.030963263, 1.030963263, 1.030963263, 23.52941
176], [17.03853955, 4.834611256, 6.341030519, 6.341030519, 88.23529412], [5.679513185, 2.093461874, 2.093461874, 2.093461874, 41.
17647059], [18.66125761, 4.407671175, 6.404101212, 6.70247411, 82.35294118], [3.245436105, 0.693777631, 0.999427916, 0.99942791
6, 17.64705882], [12.98174442, 4.024395421, 5.137835745, 5.251848153, 70.58823529], [14.60446247, 3.582900565, 4.582328481, 4.376
135828, 82.35294118], [11.35902637, 2.784813709, 2.995857953, 2.995857953, 76.47058824], [4.868154158, 1.67379918, 1.67379918, 1.
67379918, 35.29411765], [11.35902637, 3.214179586, 4.48772244, 3.47859134, 70.58823529], [21.90669371, 4.395542196, 6.365288478,
7.15367215, 100.0], [0.811359026, 0.143121959, 0.143121959, 0.143121959, 5.882352941], [8.92494929, 2.971599994, 4.577476889, 3.2
50566524, 58.82352941], [19.47261663, 3.951621543, 7.735863169, 6.037806029, 94.11764706], [4.868154158, 1.34146514, 3.62899068
7, 1.34146514, 35.29411765], [8.113590264, 3.269972892, 4.351877869, 3.49072032, 52.94117647], [12.98174442, 3.614435911, 7.58061
223, 4.279103992, 76.47058824], [5.679513185, 1.892120813, 6.79708015, 1.892120813, 41.17647059], [10.54766734, 4.388264808, 7.14
3968966, 5.317344643, 64.70588235], [10.54766734, 2.889122933, 11.36727965, 3.842460727, 58.82352941], [3.245436105, 1.17165942
6, 8.485434106, 1.171659426, 23.52941176], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0
, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0,
0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0]]]

3.4. Model Description

3.4.1. Conventional Machine Learning Models :

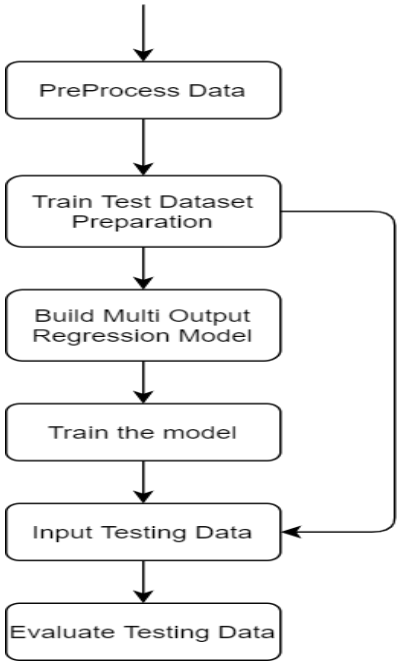
Input – Output pairs :



Model(tf-idf matrix , word features) = [[nFix[i],FFD[i],GPT[i],TRT[i],fixProp[i]]]

We feed the Tfidf matrix as the input to our model and pass the parameters to train our model on the data along with word features. We pass our Machine learning model to a multi output regressor model and predict all the features at once.

Flow chart :



3.4.1.1. Linear Regression :

Linear regression attempts to model the relationship between two variables by fitting a linear equation to observed data. One variable is considered to be an explanatory variable, and the other is considered to be a dependent variable.

A linear regression line has an equation of the form $Y = a + bX$, where X is the explanatory variable and Y is the dependent variable. The slope of the line is b and a is the intercept.

3.4.1.2. Ridge Regression :

Ridge Regression is a technique for analyzing multiple regression data that suffer from multicollinearity. When multicollinearity occurs, least squares estimates are unbiased, but their variances are large so they may be far from the true value. By adding a degree of bias to the regression estimates, ridge regression reduces the standard errors. It is hoped that the net effect will be to give estimates that are more reliable.

The cost function for ridge regression:

$$\text{Min}(||Y - X(\text{theta})||^2 + \lambda||\text{theta}||^2)$$

Lambda is the penalty term. λ given here is denoted by an alpha parameter in the ridge function. So, by changing the values of alpha, we are controlling the penalty term.

Higher the values of alpha, bigger is the penalty and therefore the magnitude of coefficients is reduced. It reduces the model complexity by coefficient shrinkage.

1. Value of alpha, which is a hyperparameter of Ridge, which means that they are not automatically learned by the model instead they have to be set manually. We run a grid search for optimum alpha values
2. To find optimum alpha for Ridge Regularization we are applying GridSearchCV.

Hence, with certain level model tuning, we can find out the best variables that influence the problem.

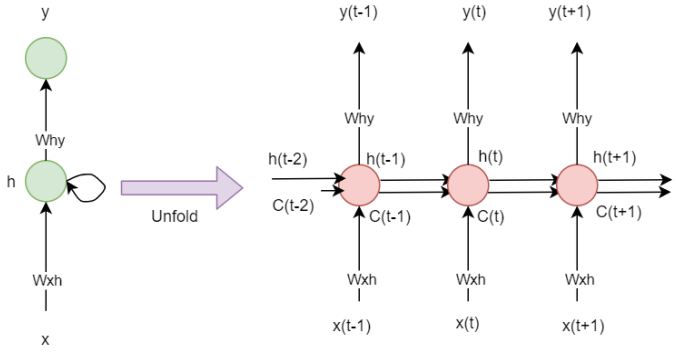
3.4.2. Deep Learning Models

3.4.2.1. LSTM Model Architecture :

Long Short-Term Memory is an advanced version of recurrent neural network (RNN) architecture that was designed to model chronological sequences and their long-range dependencies more precisely than conventional RNNs.

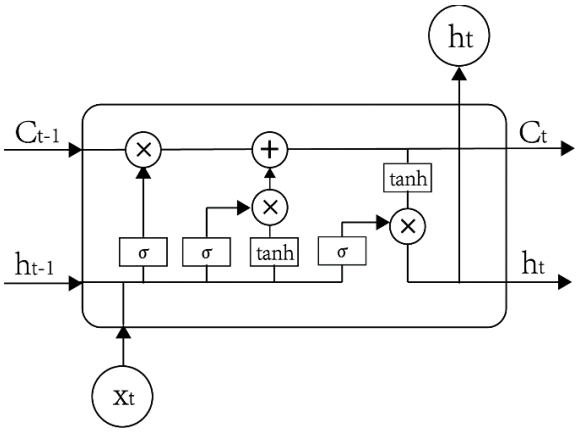
LSTMs provide us with a large range of parameters such as learning rates, and input and output biases.

LSTM Hidden Layers :



The hidden layer of LSTM is a gated unit or gated cell. It consists of four layers that interact with one another in a way to produce the output of that cell along with the cell state. These two things are then passed onto the next hidden layer. LSTMs comprises of three logistic sigmoid gates and one tanh layer. Gates have been introduced in order to limit the information that is passed through the cell. They determine which part of the information will be needed by the next cell and which part is to be discarded.

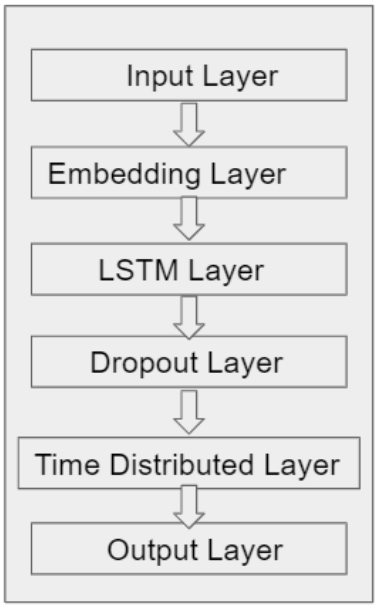
LSTM cell structure:



Each LSTM cell has three inputs h(t-1),C(t-1) and x(t) and two outputs h(t) and C(t) . For a given time t, h(t) is the hidden state, C(t) is the cell state or memory, x(t) is the current data point or input. The first sigmoid layer has two inputs– h_{t-1} and x_t where h_{t-1} is the hidden state of the previous cell. It is known as the forget gate as its output selects the amount of information of the previous cell to be included. The second sigmoid layer is the input gate that decides what new information is to be added to the cell. It takes two inputs h(t-1) and x(t). The tanh layer creates a vector C(t) of the new candidate values.

Together, these two layers determine the information to be stored in the cell state. The result is then added with the result of the forget gate multiplied with previous cell state f(t)*C(t-1) to produce the current cell state C(t). Next, the output of the cell is calculated using a sigmoid and a tanh layer. The sigmoid layer decides which part of the cell state will be present in the output whereas tanh layer shifts the output in the range of [-1,1]. The results of the two layers undergo point-wise multiplication to produce the output h(t) of the cell.

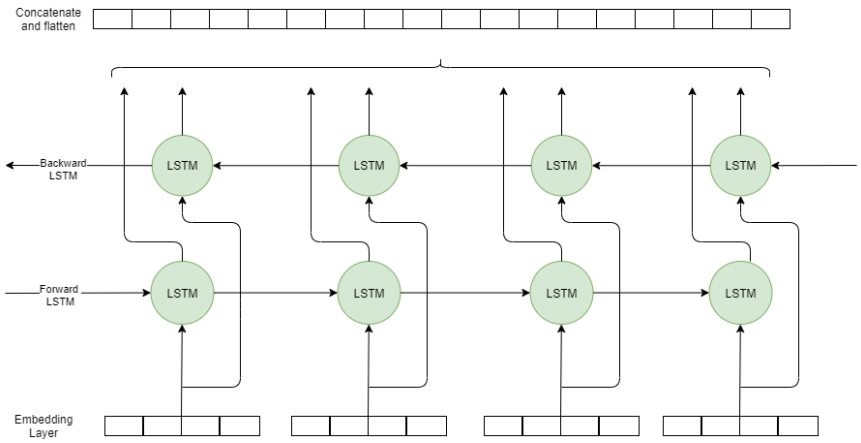
Our LSTM Model :



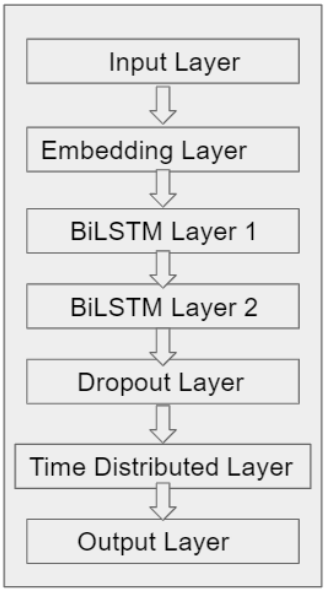
3.4.2.2. BiLSTM Model Architecture :

A Bidirectional LSTM**,** or BiLSTM**,** is a sequence processing model that consists of two LSTMs: one taking the input in a forward direction, and the other in a backwards direction. The first on the input sequence as-is and the second on a reversed copy of the input sequence. This can provide additional context to the network and result in faster and even fuller learning on the problem.

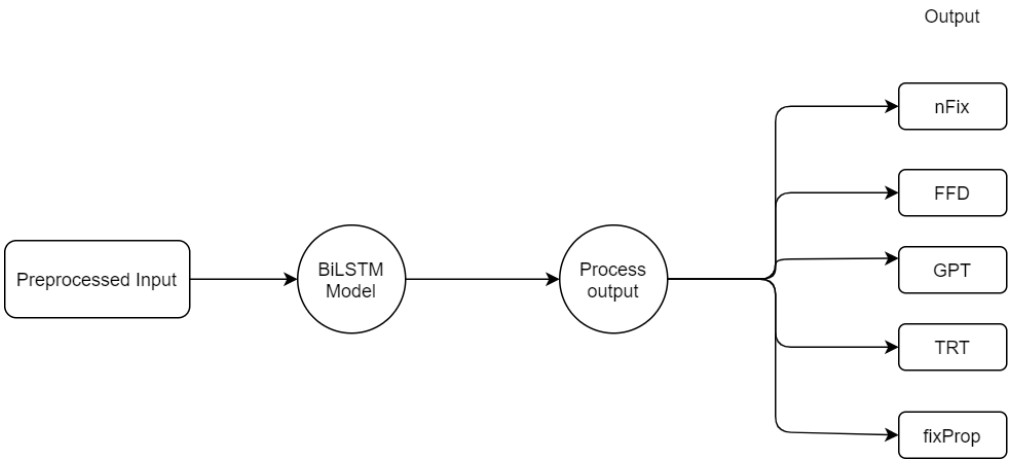
BiLSTM architecture :



Our BiLSTM Model :



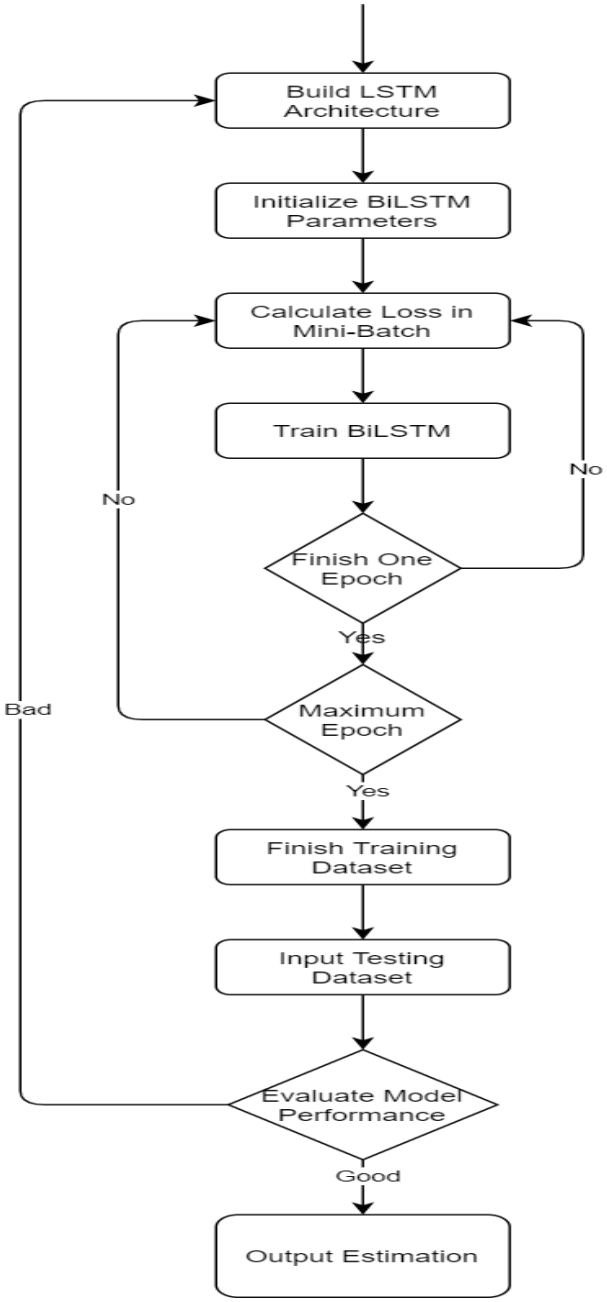
Input – Output pairs :



Model(preprocessed input , list features of sentence) =
[[nFix[i],FFD[i],GPT[i],TRT[i],fixProp[i]]]

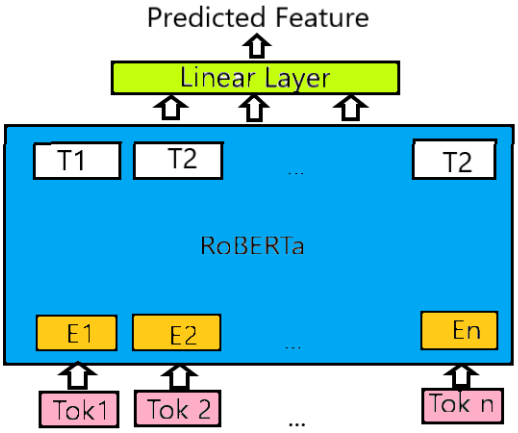
We pass the pre processed text input to our model and predict the features for test data. As the output data will also be padded to the length of longest sentence, we have to process the output and remove the padded length.

Flowchart :



3.4.2.3. RoBERTa Model Architecture :

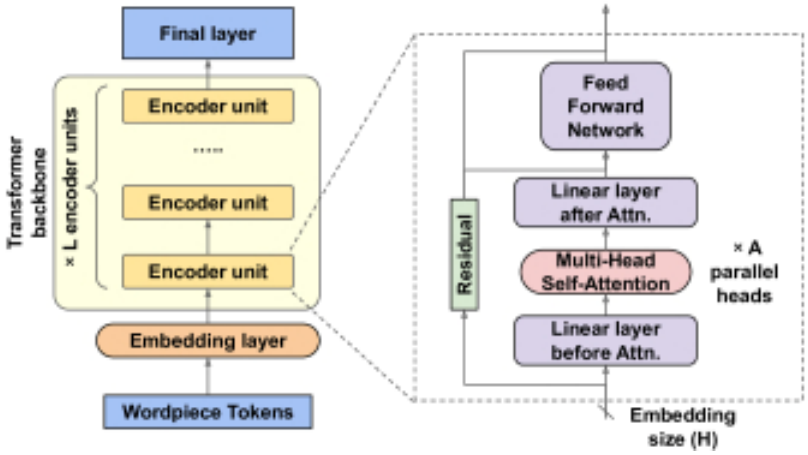
RoBERTa is a robustly optimized BERT pretraining approach. Our model consists of RoBERTa with a regression head on each token, which is a linear layer that predicts each output feature from the last layer's embeddings one at a time. The model is initialized from pretrained weights and fine-tuned on the task data.



RoBERTa model builds on BERT and modifies key hyperparameters, removing the next-sentence pretraining objective and training with much larger mini-batches and learning rates. RoBERTs has the same architecture as BERT , but uses a byte-level BPE(Byte Pair Encoding) as a tokenizer and uses a different pretraining scheme.

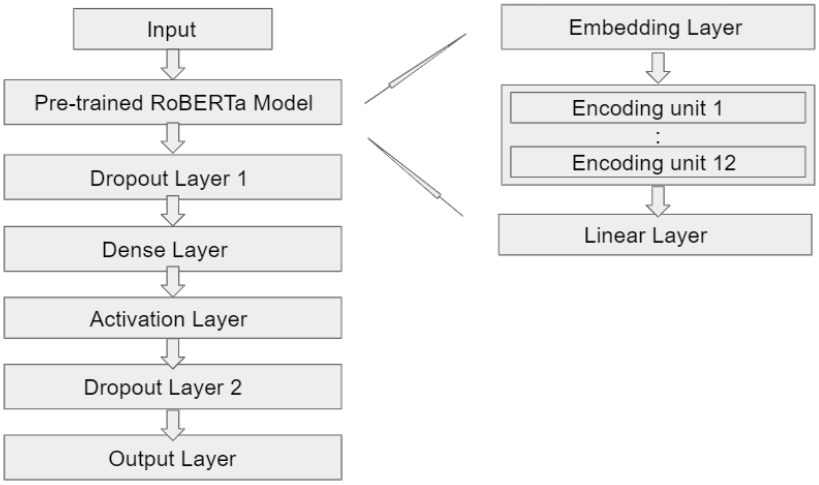
In cases where the original token is split into multiple RoBERTa tokens, we use the first RoBERTa token to make the prediction. The model is initialized with pretrained weights and fine-tuned on the task data to minimize the mean squared error across the predicted and true values.

Roberta Model Architecture :

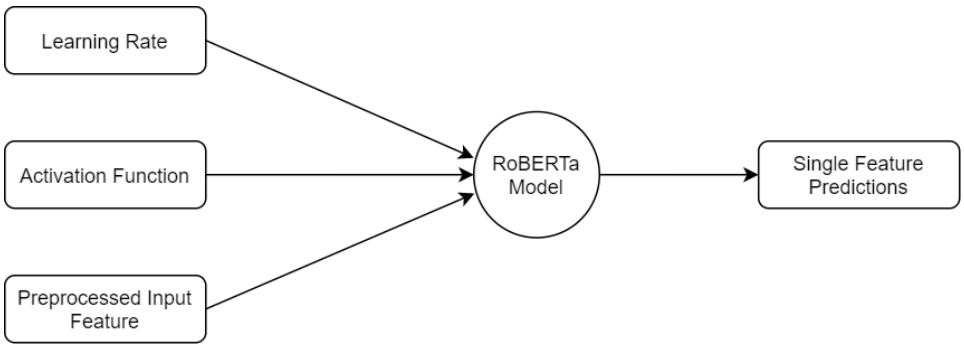


A word starts with its embedding representation from the embedding layer. Every layer does a multi-headed attention computation on the word representation of the previous layer to create a new intermediate representation. All these intermediate representations are of the same size. In the figure above, **E1** is the embedding representation, **T1** is the final output and **Trm** are the intermediate representations of the same token. In a 12-layers RoBERTa model a token will have 12 intermediate representations. In a 12 layers RoBERTa base model a token will have 12 intermediate representations as there will be 12 encoding layers.

Our Model:



Input – Output Pairs :



We pass the preprocessed text input and the feature to be predicted to our model. We train the model on that feature and predict the features for test data. The model outputs one feature at a time.

3.5. Loss Functions and Performance Metrics :

The **mean absolute error** (MAE) is the simplest regression error metric to understand. We'll calculate the residual for every data point, taking only the absolute value of each so that negative and positive residuals do not cancel out. We then take the average of all these residuals. Effectively, MAE describes the *typical* magnitude of the residuals.

$$\text{MAE} = \frac{\sum_{i=1}^{n} |y_i - x_i|}{n}$$

where y_i is the predicted value, x_i is the true value and n is the total number of samples.

The MAE is also the most intuitive of the metrics since we're just looking at the absolute difference between the data and the model's predictions. Because we use the absolute value of the residual, the MAE does not indicate underperformance or overperformance of the model . Each residual contributes proportionally to the total amount of error, meaning that larger errors will contribute linearly to the overall error.

4. Experimentation setting and Results

4.1. Conventional Machine Learning Models

4.1.1. Linear Regression Model

Pass the TF-IDF Vectorized input to the model and predict the features all at a time using multioutputregressor model.

Feature	MAE
nFix	6.89
FFD	1.14
GPT	3.80
TRT	2.62
fixProp	21.02
Mean MAE :	7.093

4.1.2. Ridge Regression Model

Pass the TF-IDF Vectorized input to the model and predict the features all at a time using multioutputregressor model.

Parameters :

alpha = 2.54 (best alpha value using GridsearchCV)

Feature	MAE
nFix	5.36
FFD	0.84
GPT	3.31
TRT	2.07
fixProp	15.11
Mean MAE :	5.340

4.2. Deep Learning Models :

4.2.1. LSTM model

Pass the preprocessed Input to the LSTM model and train it.

Layers :

- Embedding Layer
- LSTM Layer 1 : 512 units
- Dropout Layer
- Time Distributed Dense Layer : with activation = ‘relu’

Loss = mae and Optimizer = adam.

Feature	MAE (Actual)
nFix	8.5960
FFD	1.3761
GPT	4.0570
TRT	3.3038
fixProp	23.6436
Mean MAE :	8.1953

We have written a function called act_error() to unpad the output and then calculate mae using the true values to avoid a biased error.

4.2.2. BiLSTM model

Pass the preprocessed Input to the BiLSTM model and train it.

Layers :

- Embedding Layer
- BiLSTM Layer 1 : 512 units
- BiLSTM Layer 2 : 256 units
- Dropout Layer
- Time Distributed Dense Layer : with activation = ‘relu’

Loss = mae and Optimizer = adam.

Feature	MAE(actual error)
nFix	8.5544
FFD	1.4037
GPT	4.1890
TRT	3.3727
fixProp	22.1030
Mean MAE :	7.92456

We have written a function called act_error() to unpad the output and then calculate mae using the true values to avoid a biased error.

4.2.3. RoBERTa Model :

Pass the preprocessed Input to the RoBERTa model along with word lengths , word ids etc .

Number of epochs : 32

Learning rate : 5e-05

Feature to train the model on : choose one from [‘nfix’,’ffd’,’gpt’,’trt’,’fprop’]

Activation : relu

Feature	MAE
nFix	4.0181
FFD	0.6820
GPT	2.3064
TRT	1.5299
fixProp	11.2214
Mean MAE :	3.95157

5. Comparison and Discussion

The Five models presented in this project are used to predict the eye tracking features.

S.No				Model		
		Linear Regr	Ridge Regr	LSTM	BiLSTM	RoBERTa
1	MAE nFix	6.89	5.36	8.5960	8.5544	4.0181
2	MAE FFD	1.14	0.84	1.3761	1.4037	0.6820
3	MAE GPT	3.80	3.31	4.0570	4.1890	2.3064
4	MAE TRT	2.62	2.07	3.3038	3.3727	1.5299
5	MAE fixProp	21.02	15.11	23.6436	22.1030	11.2214
6	Average MAE	7.093	5.340	8.1953	7.92456	3.95157
7	Training Time	5 mins	5 mins	45 mins	1 hour	2 hours (for each feature)

Among the 5 models, Ridge and Linear Regression are the fastest but gives average results. Ridge model outperforms Linear model because of the bias/penalty term that it adds to the regression estimates. LSTM and BiLSTM models give almost similar results and they take good amount of time to train but they give high actual mae among all the models.

RoBERTa model is the slowest as it takes 2 hours to train the model for 32 epochs on a single feature but it gives the best predictions and results. The RoBERTa model gives best results i.e, the least mae because the input passes through 12 encoding layers and as the model was pre trained on a relatively large standard text dataset, the number of steps for the output to converge reduces and also efficiency increases.

6. Conclusion and Future work

In this project we propose a simple approach to predict eye tracking features using three different types of models and compared their results . In comparison with conventional eye movement models, the new approach was shown to achieve good accuracy in predicting a user's eye tracking features during reading.

From all the models we explored, RoBERTa gives the least Mean Absolute Error. As our task is a regression problem, loss matters the most but the model takes a lot of time in training. If the model is to be used for real world applications like Glaucoma detection, Partial Vision etc then the model with least amount of loss would be most preferable even though it takes much time to train because errors should be the least in medical applications.. But if you have a task with a time constraint then you can go with Ridge Regression model as it is the second best in terms of MAE and also the fastest.

In future work, several avenues may be explored to further improve performance. Predicting all the eye tracking features at once ie, Multi Output RoBERTa model is also another explorable task.

7. References

- Nora Hollenstein, Emmanuele Chersoni, Cassandra Jacobs, Yohei Oseki, Laurent Prévot, and Enrico Santus. 2021. CMCL 2021 shared task on eye-tracking prediction. In Proceedings of the Workshop on Cognitive Modeling and Computational Linguistics.

- Danny Merkx, Stefan L. Frank. 2020. Comparing Transformers and RNNs on predicting human sentence processing data.

- Xiaoming Wang, Xinbo Zhao, Jinchang Ren. 2018. A New Type of Eye Movement Model Based on Recurrent Neural Networks for Simulating the Gaze Behavior of Human Reading

- Keith Rayner. 1998. Eye movements in reading and information processing: 20 years of research. Psychological bulletin, 124(3):372.

- Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A robustly optimized BERT pretraining approach. arXiv preprint arXiv:1907.11692.