PROJECT PROGRESS

<u>ABSTRACT:</u>

We came across a number of English authors with different native languages. We highly believe that there will be a strong influence of their native language in their English works. In this project, we aimed to Fine-tune a pre-trained BERT model to find out the native language of the authors who has a native language other than English. To measure and compare the results of the fine-tuned model, we also built a logistic regression model using BERT embedding.

Data Statistics:

Total number of languages in each dataset =10

Languages: Japanese, Korean, Vietnamese, Mandarin, Russian, Thai, Spanish, Cantonese, Polish and, Arabic.

There are two datasets:

Training dataset

No. of samples = 6000 data samples

No. of sample per language: 600.

Columns: There are 3 columns namely, native language, language ID, and the appropriate text written by the author.

Test dataset

No. of samples = 2000 data samples

No. of sample per language: 200.

Columns: There are 2 columns namely, native language and the appropriate text written by the author.

Data Size (in megabytes): 4.2 for the training set and 1.4 for the test set.

Type of the data file : .csv

Sources for the datasets:

i)**Training_set**--https://docs.google.com/spreadsheets/d/14e0LqYA6YtxcOlQFg2jB7TqeI1z8qf_U1_6VdaEIA/edit?usp=sharing

ii)**Test_set**--https://docs.google.com/spreadsheets/d/1W0p1gJq1_zo21VjmvmusmfbkGnKAWXRRbTrT5Ka3b4g/edit?usp=sharing

Environment used:

- Platform: Google Colab
- Programming Language: python 3.7
- Packages: SKlearn, NumPy, torch, tqdm, Keras and pandas

2.1 Approach

1. Preprocessing: We wrote a python script to clean up and preprocess the dataset. In this script, We used pandas.read_csv() to load the datasets. We used the drop() to remove the native language columns. Converted all the text to lower case and removed unwanted punctuations using tqdm.pandas().

2. Obtaining a pre-trained model: We have obtained the Bert-base uncased model(pre-trained model) from the google AI developer's repo. In total it consists of 780 layers out of which 768 are hidden layers. These layers perform a major role in obtaining pre-trained contextual embeddings where the layers generate the fixed contextual representations of each input tokens. It also consists of 12 heads and 110 million parameters. We have chosen the uncased model because it strips out the accent markers which will lead to better classification. The reason for Bert-base is that it work on Low-end machines with limited performance and capacity.

Link: https://storage.googleapis.com/bert_models/2018_10_18/uncased_L-12_H-768_A-12.zip

3. Finetuning: We have used PyTorch-transformers to finetune the pre-trained Bert language model. We will use the Finetune Language Model repository for this purpose.

Link: https://github.com/huggingface/transformers/archive/v1.0.0.zip

Reason for fine-tuning: Since the pre-trained models have already been trained in the bottom layers of the network extensively, we only need to tune them for obtaining their features as output. This takes very small amounts of time and smaller datasets to achieve good performance compared to training an original BERT model from scratch.

4. Tokenization: After creating a fine-tuned model, we used the BERT tokenizer to tokenize the data in BERT format. When we printed the tokenized texts, we observed that for the words in BERT vocabulary, tokenizer keeps them as it is. If the word is not found in BERT vocabulary, then the tokenizer will find a subword in BERT's vocabulary and when printed we will find a "##" before the sub-word indicating such. Furthermore, we added special tokens at the beginning('CLS') and end('SEP') of each sentence for BERT to work properly.

> **Reason for BERT tokenization**: This kind of tokenization is beneficial when dealing with out of vocabulary words, and it may help better represent complicated words. We could use any other tokenization technique of course, but we'll get the best results if we tokenize with the same tokenizer the BERT model was trained on.

After obtaining tokens, we convert them to word ID as per BERT vocabulary.

5.Padding: We have chosen the Maximum sequence length as 256. For texts that have a length of less than 256, we have added padding data to fill empty spaces to bring uniformity in the input lengths. For those above which has a length above 256, the excess text has been truncated. Overall about 20% of the data has been truncated.

6.Attention Masks: We have created an attention mask to determine whether a given tokenized text is a part of token or padding.

7.Torch tensors and data loader: We have converted all our input ids, input labels, and input masks (obtained from tokenization) into torch tensors because we have fine-tuned our model using py-torch and its required data type has to be torch tensors. Since we don't need our entire dataset to be loaded into the memory , we created an iterator for our train data using torch DataLoader which helps in saving the memory during the training.

8.Performing predictions: We defined the model to the GPU(google colab) and trained it using the data loader that we created for the training dataset. Once training is done, we created a data loader for the test dataset and tested the model with it for predictions. As a result, we obtained the classification report(F1, precision, recall, and accuracy) and the confusion matrix.

2.2 Parameter Tuning

Learning rate and epochs: Setting the learning rate as 2e-5 and epochs as 4, we were able to obtain the best results out of our experiments. **Anything over these parameter values** leads to a very significant training loss and overfitting of the model where the model learns too much with a lesser iteration causing a drop in accuracy. **Anything lower than this parameter values** does not provide an efficient output. The possible reason could be related to the size of the dataset that we have.

Max_seq_length: After analyzing the datasets manually, we found that few of the author's text has a length of more than 512(Bert Threshold).Hence we tried to set the value as 512 but this ended up producing very misclassification and computational overload. This could be because of the major text's length being way too shorter than 512. Moreover we can't go very low as 128 because we might end up losing a lot of the author's content that will affect the model's learning(insufficiency of data). Hence we ended up choosing an optimal value 256, which provided better results

Batch size: Batch size more than 16 causes high memory consumption resulting in a crash during the model training (due to insufficient memory).Hence we choose 16 as batch size. I have learned that if the batch size is high, model trains faster, this was the main reason for trying out a higher batch size

Padding: Since not all the text had a sequence length higher than 256 , there were some discrepancies. Hence to form a uniformity and to fill the black spaces in the lower length texts, we have added padding to the texts.

Correct_bias: To reproduce BertAdam specific behavior, we set correct_bias=False

Experiments and results

To obtain the best results out the model , we choose two parameters that records a major effects in the results . Those are the number of epochs and learning rate of the model . These two parameters decides how the model is going to learn the training data , thats the reason these clock a major impact in the results when we stumbled between a range of values. As previously mentioned in the presentation , we experimented more than 30 possible variations of learning rate and epochs combinations. As a result , we obtained lot of interesting results which will be further discussed during analysis

Out of the experiments performed , we obtained the best optimal results of our model with,

Learning rate = 2e-5

Epochs = 4

	Precision	**Recall**	**F1 score**
Japanese	0.16	0.19	0.18
Korean	0.22	0.13	0.17
Vietnamese	0.11	0.03	0.05
Mandarin	0.13	0.21	0.16
Russian	0.09	0.07	0.08
Thai	0.18	0.26	0.21
Spanish	0.05	0.09	0.06
Cantonese	0.13	0.12	0.16
Polish	0.10	0.14	0.12
Arabic	0.40	0.41	0.41
Macro average	0.16	0.17	0.16
Weighted average	0.17	0.16	0.16

Overall accuracy: 0.17

Best classified language: Arabic

Worst classified language: Vietnamese

Anything over these parameter values leads to a very significant training loss and overfitting of the model.

Anything lower than this parameter values does not provide an efficient output. The possible reason could be related to the size of dataset that we have.

As expected, we see a lot of misclassification in the confusion matrix.

Analysis:

· The F1 scores of each language and the overall accuracy we obtained are very less. The highest F1 score is 0.41, which is not at all encouraging.

· However, we did some interesting patterns in the results. If we take the decreasing order of F1 scores, the top six F1 scores are those of Asian languages. There are exactly six Asian languages and all of them are being better classified than non-Asian languages by the fine-tuned model.

· Coming to the non-Asian languages, we see that only polish has an F1 score above 0.10. Russia, Spanish and Vietnamese are at the bottom of the table which clearly tells us that the model is doing comparatively better while working with Asian languages.

· While choosing the project topic, we were confident about the datasets and assumed that having 600 data samples per language would be enough to properly train the model. But, after obtaining the optimal results, I feel that a larger dataset might have produced better results. Now, I cannot guarantee that with larger datasets we would get best results, but I strongly believe that we can get better results as the model could be trained better.

Logistic Regression Model using Bert Vectors

In the early stages of the project, our plan was to create a Logistic Regression(LR) model using BERT embeddings. After professor's guidance, we dropped the plan and came up with the fine-tuning approach. Now after obtaining the poor results from the fine-tuned, we decided to compare the results of both approaches to have a better understanding of the working of the BERT.

1. Preprocessing: As far as training a model, the preprocessing the dataset is mandatory. As like the fine-tuned approach, here also we removed the native language columns, covert all the text to lower case, removed all the accent markers and punctuations, changed the format where each line carries on author's text and converted the .csv to .txt (helps in better training)

2. Obtaining Bert vectors: With the help of extract_feature.py script from google repo, we generate the feature vectors from the pre-trained model (BERT-Base) with the preprocessed data as input. Here we choose to set the maximum sequence length to 128, batch size to 8, and layers to -1. As we obtained 2 jsonlines files which were about 5.67 GB of data.(one embedding per training example)

3. Extracting the vectors: we convert jsonlines file to BERT vectors so that we get the vectors for train data with which we develop the model. A total of 6000 vectors have been generated. Each for each line of the input csv file(train dataset)**.**

4. Training the model: Once after obtaining the vectors, we trained a logistic regression model to predict the native_language attribute of the data provided.

5. Performing predictions: Using the test dataset's Bert vectors(performed step 3 for test jsonlines file), we have tested the model. We obtained the classification report(f1 score, precision, recall, accuracy), confusion matrix, and misclassifications with the help of the skit learn metrics

Source: The approach is inspired by an article written by **Amretha Selvraj** published in medium.com.
(https://medium.com/analytics-vidhya/step-by-step-implementation-of-bert-for-text-categorization-task-aba80417bd84)

Optimal results obtained by author’s model				Optimal results obtained by our model			
Native Language	Precision	Recall	F1 Score	Native Language	Precision	Recall	F1 score
Japanese	0.46	0.85	0.60	Japanese	0.44	0.50	0.47
Korean	0.29	0.38	0.32	Korean	0.45	0.39	0.42
Vietnamese	0.14	0.23	0.17	Vietnamese	0.47	0.40	0.43
Mandarin	0.09	0.10	0.10	Mandarin	0.34	0.32	0.33
Russian	0.59	0.56	0.57	Russian	0.50	0.56	0.53
Thai	0.71	0.56	0.63	Thai	0.50	0.51	0.50
Spanish	0.53	0.40	0.46	Spanish	0.62	0.57	0.60
Cantonese	0.39	0.23	0.29	Cantonese	0.31	0.32	0.32
Polish	0.53	0.33	0.41	Polish	0.49	0.49	0.49
Arabic	0.38	0.15	0.21	Arabic	0.47	0.51	0.49

Author’s model

Accuracy: 38%

Best classified language: Thai

Worst classified language: Mandarin

Our model

Accuracy: 46%

Best classified language: Spanish

Worst classified language: Cantonese

Analysis:

As we can see, our model has achieved better accuracy than the author's model. We have used 4 epochs and learning rate of 2e-5. Whereas they have used learning rate of 1e-5. I think setting a bigger value of learning rate helped the model train in a more efficient manner.

Both the models have given results which do not have particular pattern in terms of F1 scores.

The interesting part is that the difference between F1 scores of Mandarin, Arabic and Cantonese in each model is huge. Our model worked well with these languages leading to a large difference in overall accuracy.

Comparing our LR model with BERT fine-tuned model

In comparison, our logistic regression model has given significantly better accuracy than our fine-tuned model.

The results of logistic regression model do not have a pattern like those of fine-tuned model.

If we closely notice, F1 scores of Spanish and Russian in LR model are higher than the rest in the table, whereas in the finetuned model those are on the lower side.

Eight languages in the our LR model have F1 scores that are equal to or more than 0.41, which is the highest F1 score obtained by fine-tuned model.

Furthermore, the F1 scores of all languages in the LR model are significantly higher than those in finetuned model except for Arabic where it is only marginally higher with a mere difference of 0.06.

Additionally, the lowest F1 score in LR model results is 0.32, which is larger than all the F1 scores obtained by the finetuned model except for Arabic.

The results obtained speak volumes about LR model which is performing lot better than finetuned model for our project.

Comparing author's LR model with BERT fine-tuned model

The least classified language in the author's LR model has F1 score of 0.1 whereas there are three languages which have less than 0.1 F1 score in the fine-tuned model.

Similarly, five languages in the author's model have F1 scores that are equal to or more than 0.41, which is the highest F1 score obtained by fine-tuned model.

Just like our LR model, the author's LR model has produced better F1 scores for all languages when compared to fine-tuned model.

Link:
https://medium.com/analytics-vidhya/step-by-step-implementation-of-bert-for-text-categorizatio n-task-aba80417bd84

For learning rate=2e-10 and epoch=4

	Precision	Recall	Accuracy	Number of correct predictions(out of 600)

Japanese	0.15	0.16	0.16	96
Korean	0.17	0.22	0.19	114
Vietnamese	0.19	0.19	0.19	115
Mandarin	0.15	0.13	0.14	84
Russian	0.06	0.05	0.05	30
Thai	0.09	0.12	0.10	60
Spanish	0.05	0.02	0.05	30
Cantonese	0.02	0.02	0.03	18
Polish	0.08	0.06	0.07	42
Arabic	0.07	0.09	0.08	48
Macro average	0.10	0.09	0.08	
Weighted average	0.10	0.11	0.10	

Overall accuracy: 0.12

For learning rate= 1e-3, number of epochs=4

	Precision	Recall	Accuracy	Number of correct predictions(out of 600)
Japanese	0.19	0.18	0.19	113
Korean	0.19	0.26	0.22	132
Vietnamese	0.23	0.23	0.23	138
Mandarin	0.13	0.12	0.12	72
Russian	0.06	0.08	0.07	42
Thai	0.14	0.12	0.13	78

Spanish	0.07	0.10	0.08	48
Cantonese	0.14	0.09	0.11	66
Polish	0.03	0.04	0.04	24
Arabic	0.09	0.04	0.05	30
Macro average	0.13	0.12	0.12	
Weighted average	0.14	0.14	0.14	

Overall accuracy: 0.14

For learning rate= 1e-3, number of epochs=3

	Precision	Recall	Accuracy	Number of correct predictions(out of 600)
Japanese	0.25	0.33	0.29	174
Korean	0.06	0.03	0.04	24
Vietnamese	0.07	0.12	0.09	54
Mandarin	0.17	0.25	0.21	126
Russian	0.05	0.04	0.05	29
Thai	0.03	0.05	0.04	24
Spanish	0.07	0.04	0.05	31
Cantonese	0.06	0.03	0.04	26
Polish	0.06	0.03	0.04	24
Arabic	0.12	0.07	0.09	53
Macro average	0.09	0.10	0.09	
Weighted average	0.11	0.12	0.11	

Overall accuracy: 0.12

For learning rate= 2e-5, number of epochs=3

	Precision	Recall	Accuracy	Number of correct predictions(out of 600)
Japanese	0.12	0.15	0.13	78
Korean	0.12	0.19	0.15	90
Vietnamese	0.15	0.12	0.13	78
Mandarin	0.18	0.18	0.18	108
Russian	0.03	0.02	0.02	12
Thai	0.11	0.12	0.11	65
Spanish	0.17	0.13	0.15	91
Cantonese	0.0	0.0	0.0	0
Polish	0.09	0.08	0.08	49
Arabic	0.07	0.08	0.07	44
Macro average	0.10	0.11	0.10	
Weighted average	0.11	0.12	0.11	66

Overall accuracy: 0.12

3.2

The latest approach mentioned above is working well. However, we did hit some roadblocks as well. Initially, the native languages were in string format which was not suitable while fine tuning the model. Thus, we added another column which contained the language ID. Each language had a different ID ranging from 1 to 10. After which, we faced an CUDA runtime errors. Hence, we changed the range from 0 to 9 and it worked.

We also faced issue with learning rate. We used learning rate of 2e-5 because for a lower learning rate we were experiencing over shooting and for the higher, we were experiencing convergence.

We also faced issues with sequence length. Initially, we did not implement padding or declare a maximum sequence length. This resulted in indexing errors which we rectified after implementing padding.

3.3

Coming to data, we faced the issue with string mentioned in the above section. Our initial idea was to exclude headers of the datasets which we did. We faced computational errors which is why we decided to keep the headers.

At a point, while fine tuning the model, our code crashed because of insufficient RAM. Google colab offers 12 GB of RAM. Fortunately, we got a notification from colab offering us an option to upgrade the RAM to 25 GB for free. After the upgrade, we were able to successfully implement fine tuning.

While implementing fine tuning, we unintentionally missed to use the GPU offered by colab. The fine tuning ran for 10 hours before aborting when we realized about the GPU. The fine tuning was completed, and we got the model in less than an hour after we used the colab GPU.

We ran the model with epoch values of 2,3 and 4 and tried to find out the accuracy of the model along with precision, recall and confusion matrix but encountered errors. We are working on it and it is expected to be complete in a couple of days.

Tasks Task description Abhishek Krishna Vandadi Barani Kumar Deepa Bada Akshit Reddy

Task1 Research about new approach. Researched about the relevant resources and material needed for new approach. Researched about the relevant resources and material needed for new approach. Researched about the relevant resources and material needed for new approach. Researched about the relevant resources and material needed for new approach.

Difficulties It took some time to find good resources explaining BERT fine tuning model. We also had to rework on the approach.

Task 2 Performed Pytorch integration and prerequisite process prior to fine tuning of the model. Worked on fine tuning the model. Worked on tokenization using the BERT tokenizer and designed attention masks. Defined the model.

Difficulties Initial idea was to work with tensorflow checkpoints but had to search for an alternative when it did not seem to work.

Task 3 Setting up google colab after facing trouble with cloud. Prepare code for BERT pre trained model.(aborted after you asked us to use an existing pre trained model) Worked on the errors in the preprocessing phase and pytorch integration phase like the errors caused by removal of headers. Worked on the errors during fine tuning phase caus. Worked on the errors encountered while creating attention masks. Worked on runtime errors.

Difficulties During preprocessing, there were some minor but not easily identifiable issues with excluding the header as mentioned in the explanation.

model for NLP that has demonstrated high accuracy across complex projects.

<u>Data Statistics:</u>

Total number of languages in each dataset=10

Languages: Japanese, Korean, Vietnamese, Mandarin, Russian, Thai, Spanish, Cantonese, Polish and, Arabic.

There are two datasets:

· **Training dataset** has 6000 data samples, 600 each for every language. There are 2 columns namely, a native language column and, a text column consists of the text written by the author. We have added an additional Language ID column to make it easier while training.

· **Test dataset** has 2000 data samples, 200 each for every language. There are 2 columns namely, a native language column and, a text column consists of the text written by the author.

Data Size (in megabytes) : 4.2 for training set and 1.4 for test set.

Type of the data file : .csv file

Sources for the datasets:

Training set-

https://docs.google.com/spreadsheets/d/14e0LqYA6YtxcOIQFg2jB7TqeI1z8qf_U1_6VdaEIA/edit?usp=sharing

Test set-

https://docs.google.com/spreadsheets/d/1W0p1gJq1_zo21VjmvmusmfbkGnKAWXRRbTrT5Ka3b4g/edit?usp=sharing

Environment used:

· Platform: Google Colab

· Programming Language: python

· Packages: SKlearn, numpy, torch, tqdm and pandas

2.1 Approach

Step 1:

Preprocessing: The native language columns are removed from training and test set.

Step 2:

Obtaining a pre-trained model: We have obtained the Bert-base uncased model(pre-trained model) from the internet. We have chosen the uncased model because it strips out the accent markers which will lead to better classification and Bert-base because it consumes less memory and cost efficient. As it is developed by google developers, we also did not have any doubts on its functioning.

Link: https://storage.googleapis.com/bert_models/2018_10_18/uncased_L-12_H-768_A-12.zip

Step 3:

Finetuning: We have used pytorch-transformers to finetune pretrained Bert language model. We will use Finetune Language Model repository for this purpose.

Step 4:

We have used training set to train the model. Finally, we use test set to test the model and obtain the results.

2.2

2.3

Section 3

3.1

We initially worked towards obtaining the BERT vectors as it was our original plan. But, after the meeting with the professor, we got a clearer idea on what we are expected to do. Hence, we had to re-design our approach. Since the last two weeks, we reworked on the approach.

Updated approach:

Step 1:

The 3 datasets: training, test and validation are in .csv format. The training dataset consists of 3 columns: native language of the author, language ID and text column that consists of the text of the author. The test and validation set consist of native language column and text column that consists of text of the author. The native language column is eliminated from the training set.

Step 2:

We downloaded the BERT pre trained model downloaded from the paper published by google AI developers.

Step 3:

Finally, we use the finetune language model for predicting the native language of the author. And finally, after predictions are done, we can easily find out the accuracy of the model.

We have completed the preprocessing part where we have removed the native language column for the training data set. We have also downloaded the pre-trained BERT model developed by Google. We chose the BERT base uncased model for our project as it is cost-efficient, uses less memory comparatively and removes accent markers.

We used the BERT tokenizer to tokenize the data in BERT format. When we printed the tokenized texts, we observed that for the words in BERT vocabulary, tokenizer keeps them as it is. If the word is not found in BERT vocabulary, then the tokenizer will find a sub word in BERT's vocabulary and when printed we will find a "##" before the sub word indicating such.

We converted the tokens to word ID's as per BERT vocabulary and also padded sequences. The reason for including padding is that the sequence lengths are not the same for all the records. To avoid over exploitation of memory, we have set the maximum sequence length to 256 even though the model can take up to 512. Because of this restriction, the records with sequence length less than 256 will undergo padding whereas the ones with length of more than 256 will be truncated. Since, we have added padding, for identification purpose, attention masks have been created to find out if a given tokenized text is a part of token or padding.

Optimal results

We have tried about **35 different combinations** for number of epochs and learning rate.

We got best results for **epochs=4** and **learning rate=2e-5**

	Precision	**Recall**	**F1 score**
Japanese	0.16	0.19	0.18

Korean	0.22	0.13	0.17
Vietnamese	0.11	0.03	0.05
Mandarin	0.13	0.21	0.16
Russian	0.09	0.07	0.08
Thai	0.18	0.26	0.21
Spanish	0.05	0.09	0.06
Cantonese	0.13	0.12	0.16
Polish	0.10	0.14	0.12
Arabic	0.40	0.41	0.41
Macro average	0.16	0.17	0.16
Weighted average	0.17	0.16	0.16

Overall accuracy: **0.17**

Best classified language: **Arabic**

Worst classified language: **Vietnamese**

Anything over these parameter values leads to a very significant training loss and overfitting of the model.

Anything lower than this parameter values does not provide an efficient output. The possible reason could be related to the size of dataset that we have.

As expected, we see a lot of misclassification in the confusion matrix.

Analysis:

• The F1 scores of each language and the overall accuracy we obtained are very less. The highest F1 score is 0.41, which is not at all encouraging.

· However, we did some interesting patterns in the results. If we take the decreasing order of F1 scores, the top six F1 scores are those of Asian languages. There are exactly six Asian languages and all of them are being better classified than non-Asian languages by the fine-tuned model.

· Coming to the non-Asian languages, we see that only polish has an F1 score above 0.10. Russia, Spanish and Vietnamese are at the bottom of the table which clearly tells us that the model is doing comparatively better while working with Asian languages.

· While choosing the project topic, we were confident about the datasets and assumed that having 600 data samples per language would be enough to properly train the model. But, after obtaining the optimal results, I feel that a larger dataset might have produced better results. Now, I cannot guarantee that with larger datasets we would get best results, but I strongly believe that we can get better results as the model could be trained better.

Logistic Regression Model

Our initial idea was to create a Logistic Regression(LR) model using BERT vectors derived from pre-trained model. We later aborted the idea. However, after working with BERT finetuned model and obtaining poor results, we wanted to implement the approach and compare the results with those of finetuned model to understand the concept of BERT better.

Approach:

Step 1:

Preprocessing: We use the same datasets as earlier. Here, we remove the native language columns from both the datasets as well as Language ID column from training set.

Step 2:

Furthermore, we use the same Bert-base uncased model(pre-trained model). We obtain bert vectors from the pre-trained model.

Step 3:

Using these vectors, we develop a logistic regression model which we used for prediction.

Source: The approach is inspired from an article written by Amretha Selvraj which was published by medium.com.

Optimal results obtained by author's model				**Optimal results obtained by our model**			
Native Language	**Precision**	**Recall**	**F1 Score**	**Native Language**	**Precision**	**Recall**	**F1 score**
Japanese	0.46	0.85	0.60	Japanese	0.44	0.50	0.47
Korean	0.29	0.38	0.32	Korean	0.45	0.39	0.42
Vietnamese	0.14	0.23	0.17	Vietnamese	0.47	0.40	0.43
Mandarin	0.09	0.10	0.10	Mandarin	0.34	0.32	0.33
Russian	0.59	0.56	0.57	Russian	0.50	0.56	0.53
Thai	0.71	0.56	0.63	Thai	0.50	0.51	0.50
Spanish	0.53	0.40	0.46	Spanish	0.62	0.57	0.60

Cantonese	0.39	0.23	0.29	Cantonese	0.31	0.32	0.32
Polish	0.53	0.33	0.41	Polish	0.49	0.49	0.49
Arabic	0.38	0.15	0.21	Arabic	0.47	0.51	0.49

Author's model

Accuracy: **38%**

Best classified language: **Thai**

Worst classified language: **Mandarin**

Our model

Accuracy: **46%**

Best classified language: **Spanish**

Worst classified language: **Cantonese**

Analysis:

As we can see, our model has achieved better accuracy than the author's model. We have used 4 epochs and learning rate of 2e-5. Whereas they have used learning rate of 1e-5. I think setting a bigger value of learning rate helped the model train in a more efficient manner.

Both the models have given results which do not have particular pattern in terms of F1 scores.

The interesting part is that the difference between F1 scores of Mandarin, Arabic and Cantonese in each model is huge. Our model worked well with these languages leading to a large difference in overall accuracy.

Comparing our LR model with BERT fine-tuned model

In comparison, our logistic regression model has given significantly better accuracy than our fine-tuned model.

The results of logistic regression model do not have a pattern like those of fine-tuned model.

If we closely notice, F1 scores of Spanish and Russian in LR model are higher than the rest in the table, whereas in the finetuned model those are on the lower side.

Eight languages in the our LR model have F1 scores that are equal to or more than 0.41, which is the highest F1 score obtained by fine-tuned model.

Furthermore, the F1 scores of all languages in the LR model are significantly higher than those in finetuned model except for Arabic where it is only marginally higher with a mere difference of 0.06.

Additionally, the lowest F1 score in LR model results is 0.32, which is larger than all the F1 scores obtained by the finetuned model except for Arabic.

The results obtained speak volumes about LR model which is performing lot better than finetuned model for our project.

Comparing author's LR model with BERT fine-tuned model

The least classified language in the author's LR model has F1 score of 0.1 whereas there are three languages which have less than 0.1 F1 score in the fine-tuned model.

Similarly, five languages in the author's model have F1 scores that are equal to or more than 0.41, which is the highest F1 score obtained by fine-tuned model.

Just like our LR model, the author's LR model has produced better F1 scores for all languages when compared to fine-tuned model.

Link:
https://medium.com/analytics-vidhya/step-by-step-implementation-of-bert-for-text-categorization-task-aba80417bd84

For learning rate=2e-10 and epoch=4

	Precision	Recall	Accuracy	Number of correct predictions(out of 600)
Japanese	0.15	0.16	0.16	96
Korean	0.17	0.22	0.19	114
Vietnamese	0.19	0.19	0.19	115
Mandarin	0.15	0.13	0.14	84
Russian	0.06	0.05	0.05	30
Thai	0.09	0.12	0.10	60
Spanish	0.05	0.02	0.05	30
Cantonese	0.02	0.02	0.03	18
Polish	0.08	0.06	0.07	42

Arabic	0.07	0.09	0.08	48
Macro average	0.10	0.09	0.08	
Weighted average	0.10	0.11	0.10	

Overall accuracy: 0.12

For learning rate= 1e-3, number of epochs=4

	Precision	Recall	Accuracy	Number of correct predictions(out of 600)
Japanese	0.19	0.18	0.19	113
Korean	0.19	0.26	0.22	132
Vietnamese	0.23	0.23	0.23	138
Mandarin	0.13	0.12	0.12	72
Russian	0.06	0.08	0.07	42
Thai	0.14	0.12	0.13	78
Spanish	0.07	0.10	0.08	48
Cantonese	0.14	0.09	0.11	66

Polish	0.03	0.04	0.04	24
Arabic	0.09	0.04	0.05	30
Macro average	0.13	0.12	0.12	
Weighted average	0.14	0.14	0.14	

Overall accuracy: 0.14

For learning rate= 1e-3, number of epochs=3

	Precision	Recall	Accuracy	Number of correct predictions(out of 600)
Japanese	0.25	0.33	0.29	174
Korean	0.06	0.03	0.04	24
Vietnamese	0.07	0.12	0.09	54
Mandarin	0.17	0.25	0.21	126
Russian	0.05	0.04	0.05	29
Thai	0.03	0.05	0.04	24
Spanish	0.07	0.04	0.05	31

Cantonese	0.06	0.03	0.04	26
Polish	0.06	0.03	0.04	24
Arabic	0.12	0.07	0.09	53
Macro average	0.09	0.10	0.09	
Weighted average	0.11	0.12	0.11	

Overall accuracy: 0.12

For learning rate= 2e-5, number of epochs=3

	Precision	Recall	Accuracy	Number of correct predictions(out of 600)
Japanese	0.12	0.15	0.13	78
Korean	0.12	0.19	0.15	90
Vietnamese	0.15	0.12	0.13	78
Mandarin	0.18	0.18	0.18	108
Russian	0.03	0.02	0.02	12
Thai	0.11	0.12	0.11	65

Spanish	0.17	0.13	0.15	91
Cantonese	0.0	0.0	0.0	0
Polish	0.09	0.08	0.08	49
Arabic	0.07	0.08	0.07	44
Macro average	0.10	0.11	0.10	
Weighted average	0.11	0.12	0.11	66

Overall accuracy: 0.12

3.2

The latest approach mentioned above is working well. However, we did hit some roadblocks as well. Initially, the native languages were in string format which was not suitable while fine tuning the model. Thus, we added another column which contained the language ID. Each language had a different ID ranging from 1 to 10. After which, we faced an CUDA runtime errors. Hence, we changed the range from 0 to 9 and it worked.

We also faced issue with learning rate. We used learning rate of $2e^{-5}$ because for a lower learning rate we were experiencing over shooting and for the higher, we were experiencing convergence.

We also faced issues with sequence length. Initially, we did not implement padding or declare a maximum sequence length. This resulted in indexing errors which we rectified after implementing padding.

3.3

Coming to data, we faced the issue with string mentioned in the above section. Our initial idea was to exclude headers of the datasets which we did. We faced computational errors which is why we decided to keep the headers.

At a point, while fine tuning the model, our code crashed because of insufficient RAM. Google colab offers 12 GB of RAM. Fortunately, we got a notification from colab offering us an option to upgrade the RAM to 25 GB for free. After the upgrade, we were able to successfully implement fine tuning.

While implementing fine tuning, we unintentionally missed to use the GPU offered by colab. The fine tuning ran for 10 hours before aborting when we realized about the GPU. The fine tuning was completed, and we got the model in less than an hour after we used the colab GPU.

We ran the model with epoch values of 2,3 and 4 and tried to find out the accuracy of the model along with precision, recall and confusion matrix but encountered errors. We are working on it and it is expected to be complete in a couple of days.

Tasks	**Task description**	**Abhishek Krishna Vandadi**	**Barani Kumar**	**Deepa Bada**	**Akshit Reddy**
Task1	Research about new approach.	Researched about the relevant resources and material needed for new approach.	Researched about the relevant resources and material needed for new approach.	Researched about the relevant resources and material needed for new approach.	Researched about the relevant resources and material needed for new approach.
Difficulties	It took some time to find good resources explaining BERT fine tuning model. We also had to rework on the approach.				

Task 2		Performed Pytorch integration and prerequisite process prior to fine tuning of the model.	Worked on fine tuning the model.	Worked on tokenization using the BERT tokenizer and designed attention masks.	Defined the model.
Difficulties		Initial idea was to work with tensorflow checkpoints but had to search for an alternative when it did not seem to work.			
Task 3	Setting up google colab after facing trouble with cloud. Prepare code for BERT pre trained model.(aborted after you asked us to use an existing pre trained model)	Worked on the errors in the preprocessing phase and pytorch integration phase like the errors caused by removal of headers.	Worked on the errors during fine tuning phase caus.	Worked on the errors encountered while creating attention masks.	Worked on runtime errors.

Difficulties		During preprocessing, there were some minor but not easily identifiable issues with excluding the header as mentioned in the explanation.			