# Machine Learning Engineer Nanodegree

## Capstone Project Report

by Wasifa Chowdhury

# Definition

## Project Overview

Natural language processing (NLP) has grown by leaps and bounds over the past few years with the emergence of deep learning technologies. Machine learning models can now tackle complex tasks like question answering, text extraction, and sentence generation. A characteristic of natural language is that there are many different ways to convey the same information - several meanings can be contained in a single text and that the same meaning can be expressed by different texts. Take, for example, the following text,

*"How often have I said to you that when you have eliminated the impossible, whatever remains, however improbable, must be the truth?"* -Sir Arthur Conan Doyle

While humans can process the meaning of this sentence rather quickly, can we measure how well machines navigate the ambiguity of multiple senses of words and draw on context to determine the relationship between sentences? If NLP can be applied between sentences, this could have profound implications for real-world applications like fact-checking, identifying fake news, and much more.

Moreover, most of the work to date on NLP has been focused mainly on English, leaving low- and medium-resource languages behind. As more and more of the everyday discourse moves online, developing NLP systems in other languages is a growing need since information is shared not only in English but in other languages as well. To address this problem, Kaggle has initiated a competition, Contradictory, My Dear Watson, to challenge machine learning practitioners to build a system that automatically classifies how pairs of sentences are related from texts in fifteen different languages. The aim of this capstone project is to create a multi-class classification model using transfer learning.

My personal motivation for this project comes from the fact that I'm interested in how natural language understanding can be used to break down language barriers. Advancement in language understanding and representation tasks can have real impact in the daily lives of

many people by making digital information accessible to everyone, regardless of language or geographic location. Also as someone whose native language is a scarce-resource language, it motivates me more to investigate better machine learning methods for multilingual corpus.

## Problem Statement

Textual entailment recognition, also known as natural language inferencing, is the task of deciding, given two text fragments, whether the meaning of one text can be inferred from another text. Textual entailment measures natural language understanding as it requires computing semantic interpretation of the text.

For premise-hypothesis sentence pairs, the goal is to predict whether a hypothesis statement is true (entailment), false (contradiction), or undetermined (neutral) given a premise sentence. Here entailment is a directional relation which means that the hypothesis must be entailed from the given premise, but the premise need not be entailed from the hypothesis.

The table below shows an example of each of these cases for the following premise:

*He came, he opened the door and I remember looking back and seeing the expression on his face, and I could tell that he was disappointed.*

| | |
|---|---|
| *Just by the look on his face when he came through the door I just knew that he was let down.* | *entailment* |
| *He was trying not to make us feel guilty but we knew we had caused him trouble.* | *neutral* |
| *He was so excited and bursting with joy that he practically knocked the door off it's frame.* | *contradiction* |

The first hypothesis statement in the table is true based on the information in the premise. So, this pair is related by entailment. The information contained in the second hypothesis might be true, but we can't conclude this based on the information in the premise. So, this relationship is neutral. Since the last hypothesis is completely opposite of what the premise says, this pair is related by contradiction.

Since the pair of input sentences are in fifteen different languages, the goal is to train a multilingual model that can handle multiple languages simultaneously and classify input pairs into the three classes.

In recent years, deep learning based techniques have pushed the state-of-the-art limits in varying NLP tasks (e.g. language modeling[1], machine translation[2], and question answering[3]). For the capstone project, transfer learning with data augmentation will be used to train neural networks with Transformer[4] architecture and classify the textual entailment relationship between premise-hypothesis pairs. Instead of training a separate model for every language, a single model will be trained on the multilingual corpus in order to learn similar underlying language patterns across languages. Since the training dataset is small, consisting of less than 100k

examples, we will fine-tune publicly available pre-trained language models that are significantly cheaper than training a model from scratch. We will use the XLM-RoBERTa[5] model from Facebook that is pre-trained on 2.5TB of filtered CommonCrawl data in 100 different languages. Data augmentation in the form of back-translation will also be used to boost model's performance.

## Metrics

Accuracy is used as the evaluation metric to quantify the performance of both the benchmark model and the solution model presented. For each input sample in the test set, the goal is to predict whether a given hypothesis is related to its premise by contradiction, entailment, or whether neither of those is true (neutral). The output classes map to the logical condition as:

0 == entailment

1 == neutral

2 == contradiction

Hence accuracy returns the ratio between the number of correct predictions to the total number of predictions on the test dataset, as shown in the following formula:

$$Accuracy = \frac{Number\ of\ correct\ predictions}{Total\ number\ of\ predictions}$$

Since all the output classes are of equal importance in our task, accuracy is a suitable metric to measure how a model performs across all the classes.

# Analysis

## Data Exploration

The dataset consists of train and test files with the following format:

- train.csv: This file contains the ID, premise, hypothesis, and label, as well as the language of the text and its two-letter abbreviation
- test.csv: This file contains the ID, premise, hypothesis, language, and language abbreviation, without labels.

The table below shows sample rows from the training set. Here the ID column holds a unique identifier for each sample while the label column stores the classification of the relationship

between the premise and hypothesis (e.g. 0 for entailment, 1 for neutral, 2 for contradiction). The pair of input sentences are in the following languages - Arabic, Bulgarian, Chinese, German, Greek, English, Spanish, French, Hindi, Russian, Swahili, Thai, Turkish, Urdu, and Vietnamese. The train set contains 12,120 data examples while the test set contains 5,195 examples in the aforementioned languages.

| | id | premise | hypothesis | lang_abv | language | label |
|---|---|---|---|---|---|---|
| **0** | 5130fd2cb5 | and these comments were considered in formulat... | The rules developed in the interim were put to... | en | English | 0 |
| **1** | 5b72532a0b | These are issues that we wrestle with in pract... | Practice groups are not permitted to work on t... | en | English | 2 |
| **2** | 3931fbe82a | Des petites choses comme celles-là font une di... | J'essayais d'accomplir quelque chose. | fr | French | 0 |
| **3** | 5622f0c60b | you know they can't really defend themselves I... | They can't defend themselves because of their ... | en | English | 0 |
| **4** | 86aaa48b45 | ในการเล่นบทบาทสมมุติก็เช่นกัน โอกาสที่จะได้แสด... | เด็กสามารถเห็นได้ว่าชาติพันธุ์แตกต่างกันอย่างไร | th | Thai | 1 |

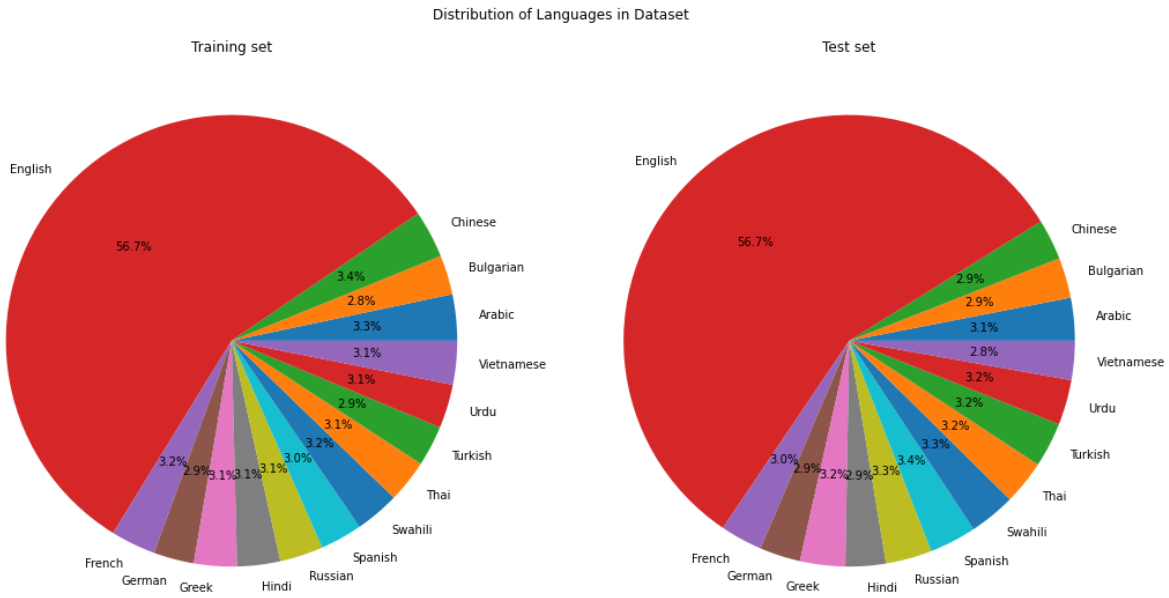
We firstly get the number of missing data points per column and drop any incomplete rows of data; the stats are shown below.

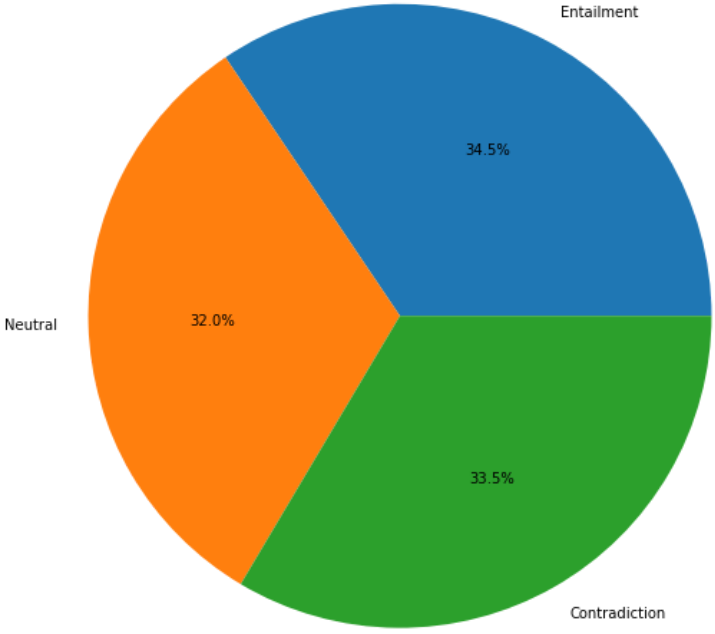
```
Number of missing data points per column:

id           0
premise      0
hypothesis   0
lang_abv     0
language     0
label        0
dtype: int64
```

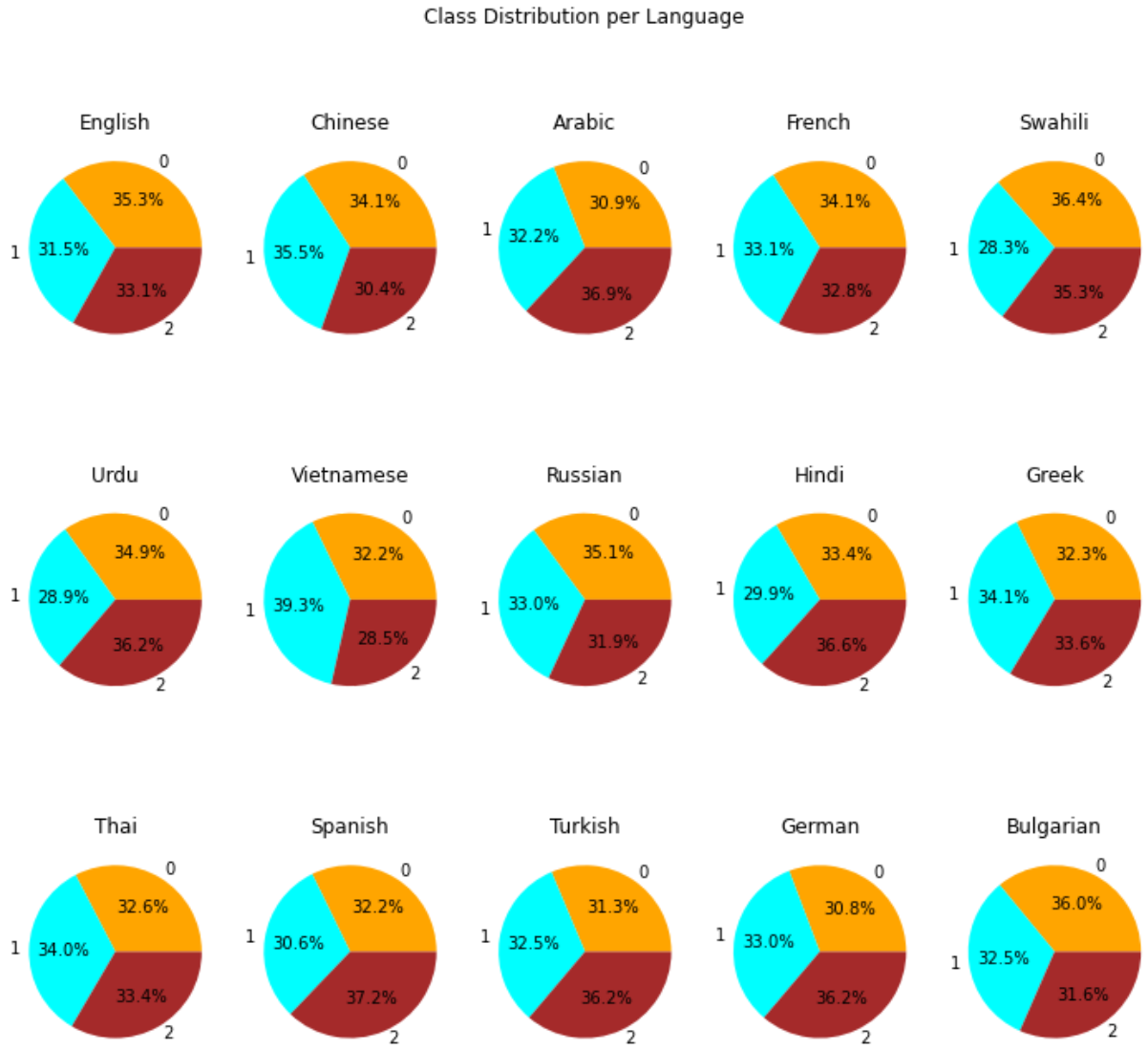
The following figures plot the data statistics on the training and test sets. The first two pie charts compare the distribution of examples across all the languages for the two sets. As expected, more than half of the training and test examples are in English as data resources are abundant in this language. Rest of the data is fairly shared between other fourteen languages.

We also check the number of training observations per class. We can see that the overall training set contains roughly equal numbers of examples per class, while the distribution of the training labels is also quite balanced in all the languages.



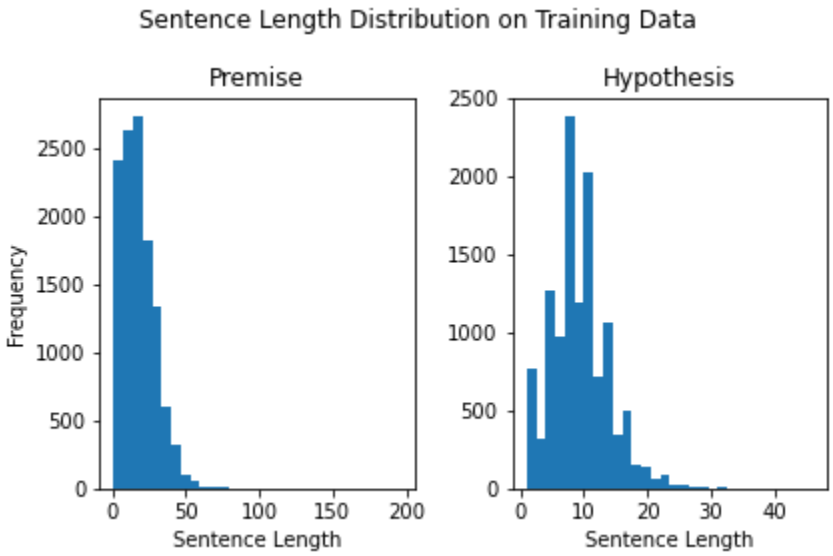

Class Distribution over Training Data



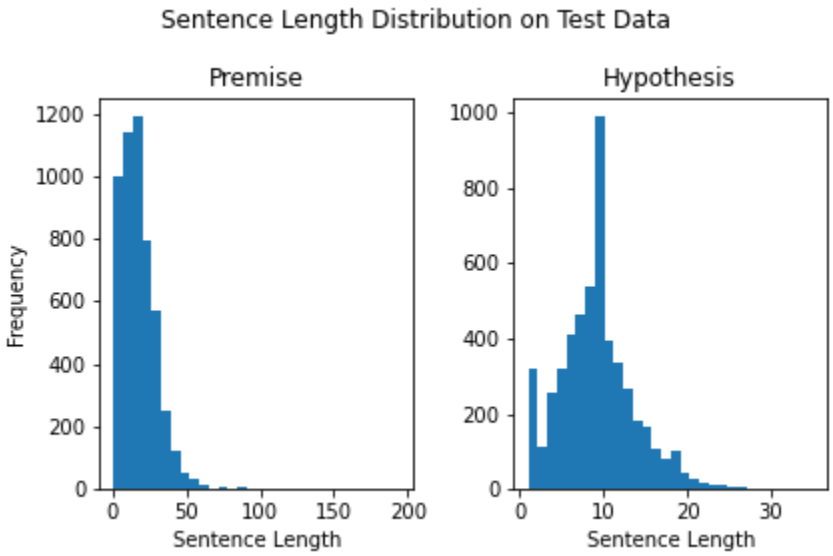

## Exploratory Visualization

We visualize the sentence length distribution, defined as the number of words in a sentence, for input sentences in training as well as test data in all the languages.

The following figure shows the sentence length count in premise and hypothesis sentences of the training data. We can see that the lengths of premise sentences are greater than those of the hypothesis sentences in the training set - majority of the premise sentences have lengths with no more than 50 tokens while nearly all of the hypothesis input sequences have less than 20 tokens. This is justifiable as the premise is the statement on which the argument is based containing necessary information to verify the truth of the hypothesis statement.




We also plot the sentence length distribution on the test data, as provided in the figure below. Similar to training data, premise sentences in the test set are longer than the hypothesis sentences.



# Algorithms and Techniques

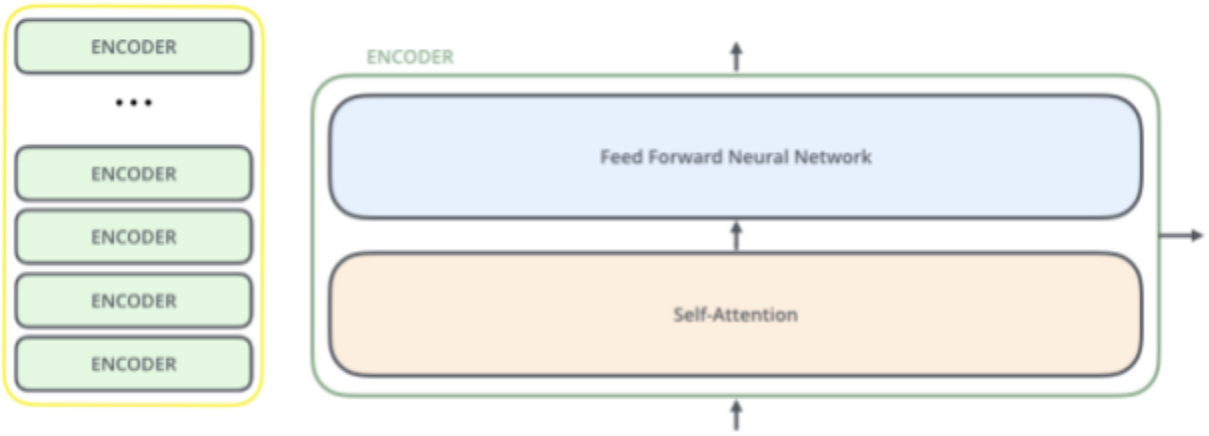
## Transfer Learning:

To classify the textual entailment relation between the premise and hypothesis sentences, we use transfer learning from a pre-trained network, which is a saved network that was previously trained on an enormous amount of unannotated text on the web. The intuition behind transfer learning is that if a model is trained on a large and general enough dataset, this model can effectively serve as a generic model able to capture the language patterns. We can then take advantage of these learned feature maps and customize this model for our specific task.

The base model is created and initialized using the configuration settings and weights from the pre-trained network. Classification layers are stacked on top of this base model to classify input instances into the 3 classes. The entire model is then fine-tuned by jointly training both the newly-added dense layers and the layers of the base model using our small labeled dataset. This allows the model weights to be adjusted to get feature representations relevant to the task at hand.

## Multilingual BERT:

The Multilingual BERT[6] or M-BERT was released by Google along with BERT[7], which stands for Bidirectional Encoder Representations for Transformers. BERT is a pre-trained model consisting of a stack of Transformer-based encoders. All the encoders are identical and each one contains two sub-layers, the self-attention and the feed-forward layers. BERT takes an input sequence which keeps flowing up the stack. Each encoder firstly applies self-attention to look at other relevant words in the input sentence as it encodes a specific word. The results from the self-attention layer are then passed through the feed-forward network, whose output is fed as input in the next encoder.



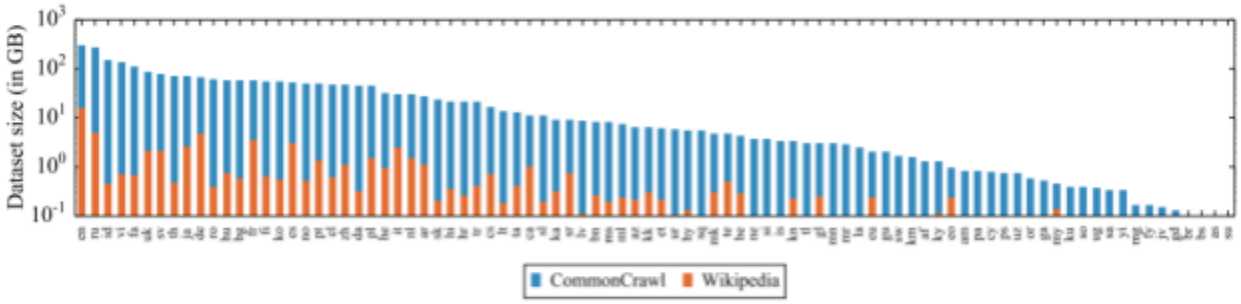

BERT architecture. Source: [8, 9]

To better handle language, BERT is jointly pre-trained using English Wikipedia and BookCorpus datasets on two different tasks - Masked Language Model (MLM) and Next Sentence Prediction (NSP) tasks. In MLM, some of the input words are randomly masked out and then the masked words are predicted by conditioning on both the left and right input context. In NSP, BERT learns to model relationships between sentences through the binary classification task of predicting, given two input sentences, whether one sentence follows another one.

As our dataset is multilingual, we use the M-BERT, which is the same as BERT but trained on text from 104 different languages. In particular, it was trained on Wikipedia content with a shared vocabulary across all the languages. Since this model is pre-trained on 2 language-based tasks using corpus in diverse languages, which includes the 15 languages from our dataset as well, it is a suitable baseline for our problem.

## XLM-RoBERTa:

The XLM-RoBERTa or XLM-R model relies solely on the MLM objective and is pre-trained with multilingual data in an enormous scale. Unlabeled text in 100 languages is extracted from CommonCrawl[10] corpus, totaling 2.5TB of text to make up the training data. This increases the amount of data by several orders of magnitude, in particular for low-resource languages, as shown in the figure below. We can also see that this dataset is several orders of magnitude larger than the Wikipedia corpus that was used to train the BERT model. Hence we use this model to improve on the BERT baseline.




Amount of data in GiB (log-scale) for the 88 languages that appear in both Wikipedia and CommonCrawl corpus. Source: [5]

Here is a summary of the different layers in the XLM-R model architecture:

**Embedding:** In the embedding layer, words in an input sequence are mapped to vector representations by looking up in a matrix of size $V \times D_w$, where $V$ is the vocabulary and $D_w$ is the size of the word embeddings. The embedding vectors are then fed as input to the first encoder layer. Positional vectors of dimension $D_p$ are also added to each word embedding to keep track of the order of the words in the input sequences.

**Encoder:** Each encoder layer processes the self-attention and feed-forward vectors and passes outputs to the next encoder layer.

**Layer Normalization:** Layer normalization step is applied within the encoder layers to normalize the distributions of the intermediate layers and speed up learning.

**Dropout:** Large neural networks, like M-BERT and XLM-RoBERTa, when trained on relatively small datasets can overfit the training data. Regularization in the form of dropout is applied to the neural networks by randomly ignoring 10% of the neurons in the intermediate layers of the encoders.

## Data Augmentation:

The amount of labeled data available to train a machine learning model might impact the model's performance. This is especially true in case of deep learning-based NLP models that generally benefit from larger amounts of annotated training examples to be able to distinguish between the different output classes. However it can be an expensive and time-consuming process to manually annotate additional data. As we saw in the Data Exploration section, most of the training examples in our dataset are in English. To increase the number of training examples in other low-resource languages, we use data augmentation and generate additional, synthetic data using the original data we have.

While generating augmented images can be easily performed using flipping, rotating and other transformations to the input image, it is not easy to augment textual data due to the ambiguities present in a language. For example, changing a word in a sentence with its synonym might alter the overall context of the sentence. Since our data is in different languages where external knowledge-base resources are not readily available for the low-resource languages, we use back-translation to improve our model's performance.

In back-translation, the input text data is translated to some language and then translated back to the original language. This can help to generate textual data with different words while preserving the context of the input text data. The table shows a back-translation of a hypothesis input in the train set.

| | |
|---|---|
| **Input** | *The rules developed in the interim were put together with these comments in mind.* |
| **Back-translated** | *The rules developed in the meantime have been drawn up taking these comments into account.* |

## Benchmark

**Random choice:** This is a naive approach where there is an equal probability for an input sample to belong to any one of the 3 output classes. This submission yields ~33% accuracy in the Kaggle leaderboard.

**Multilingual BERT:** We fine-tune the M-BERT model on our training dataset to get the baseline predictions for textual entailment recognition. We extract the pretrained BERT vector representations from the *[CLS]* or classification token in the last layer and pass that as input to a linear layer with *Softmax* activation function for further training. This yields an accuracy of ~65% in the submission leaderboard.

# Methodology

## Data Preprocessing

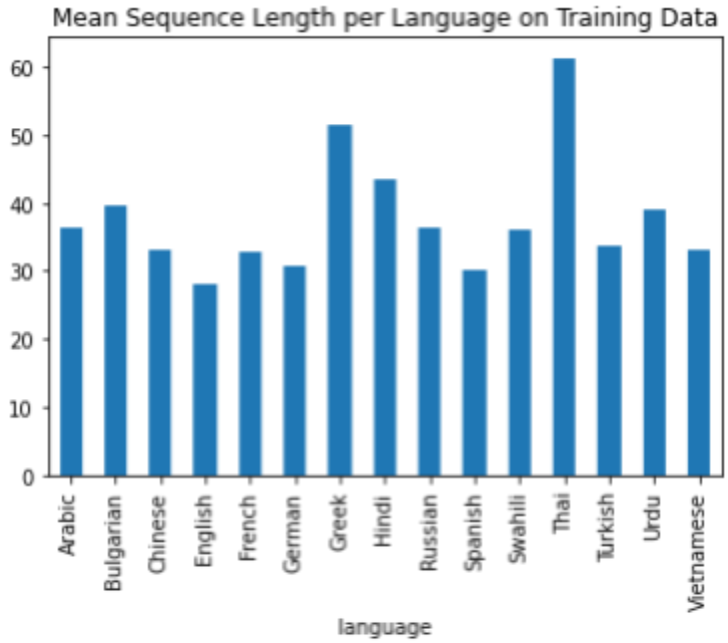
To prepare the inputs for fine-tuning the models, we use the same preprocessing steps used to pre-train the M-BERT and XLM-RoBERTa models. There are mainly three steps to preprocessing - normalization, tokenization (e.g. splitting strings in sub-word token strings) and converting token strings to ids.

Each model applies different rules for processing a text. The M-BERT model does not perform any normalization on the input (e.g. no lower casing, accent stripping, or Unicode normalization). Hence we also follow the same rules when preprocessing input data for our task. For tokenization, the model uses a WordPiece[11] tokenizer with a shared vocabulary of size 119,547. Thus our input sentences are tokenized into subwords so that each word piece is an element of the dictionary. A benefit of using a subword tokenization algorithm is that while frequently used words do not need be split into smaller subwords, unseen and rare words can be decomposed into meaningful subwords. For example, the English word, *annoyingly*, might be considered a rare word and could be decomposed into *annoying* and *ly*.

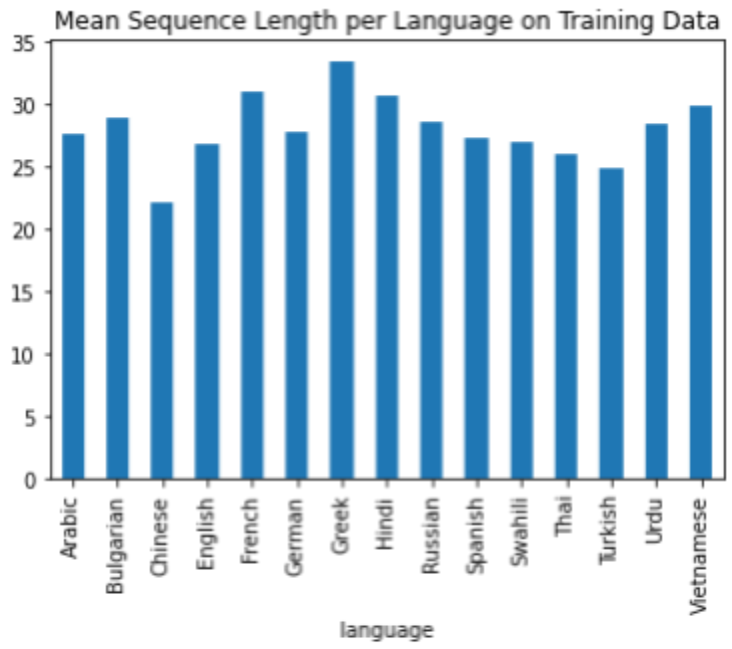
The first figure below shows the mean sequence length (e.g. average number of word/subword tokens in a sentence) per language for the premise inputs in our training set. We can see that Thai, Greek, and Hindi languages have the highest number of tokens on average while English and German have the least number of tokens consisting of around 30 tokens on average. Since many of the English words can be found as a whole word in the vocabulary, the tokenizer keeps sentences in English mostly intact. On the other hand, sentences in morphologically rich languages, like Greek and Hindi, contain many inflected words that need to be broken down further. Hence the tokenizer generates longer token distributions for those languages. Also, Thai language doesn’t use space between words, hence it is hard to identify sentence boundaries in Thai text. This might be the reason for the tokenizer to segment the Thai words into multiple subwords. While Chinese language also doesn’t have sentence boundaries, M-BERT adds

spaces around every Chinese character in the CJK Unicode range before applying WordPiece tokenization.

The XLM-RoBERTa model uses the SentencePiece[12] tokenizer with a vocabulary size of 250,002. Since not all natural languages are space segmented, the SentencePiece algorithm firstly converts the inputs into unicode characters and includes spaces in the set of characters. It then uses multiple subword algorithms, including byte-pair encoding[13] (BPE) and unigram language model[14], to construct the appropriate vocabulary. From the second figure below, we can see that the tokenizer shrinks the average sequence length for all languages on the training data. Specifically, the average number of subword tokens for input sentences in Thai is significantly reduced from ~61 tokens using WordPiece tokenizer to ~26 tokens using SentencePiece tokenizer.




Using WordPiece tokenizer




Using SentencePiece tokenizer

To handle the inputs for different types of NLP-based tasks, Transformer-based models follow a number of input transformations. For textual entailment classification, since every input example contains pair of sentences, the M-BERT model expects the input to be in the following format:

*[CLS] ...Premise tokens... [SEP]..Hypothesis tokens... [SEP]*

Here, *[CLS]* and *[SEP]* are special tokens included in the vocabulary, where *[CLS]* is used in the beginning of an input sequence for sentence-level classification while *[SEP]* token separates the two sentences. The input sequences also need to be truncated or padded with the *[PAD]* token because each vector in an input training batch needs to have the same size. In case of M-BERT, the maximum allowed length is 512. The maximum length of the input sequences should be large enough such that we don't lose much data when fed in the machine learning model. Additionally, a very big number would make the model complex. Since the majority of the premise and hypothesis sentences are shorter than 60 tokens, we set the maximum length of input sequences to 120.

The input token sequences are then converted into numerical forms where three types of input are fed into the M-BERT model - *input_ids*, *token_type_ids*, and *attention_mask*. *Input_ids* map the input sequence tokens to their respective indices in the vocabulary. The *token_type_ids* segment token indices to indicate first and second portions of the inputs. Indices are selected in [0, 1], where,

- 0 corresponds to a premise sentence token
- 1 corresponds to a hypothesis sentence token

*Attention_mask* tells the model which tokens in the input sequences are words and which are padding. Mask values are selected in [0, 1], where,

- 1 used for tokens that are not masked
- 0 used for tokens that are masked

Likewise, for the XLM-R model, the input token sequences are represented in the following format:

*<s> ...Premise tokens... </s></s>..Hypothesis tokens... </s>*

The *<s>* and *</s>* are special tokens in the vocabulary to specify beginning and end of input sequences and *</s>* also indicates which tokens are part of the premise sentence and which are part of the hypothesis sentence.

In the case of the XLM-R model, we pass only the *input_ids* and *attention_mask* vectors during training and inference.

# Implementation

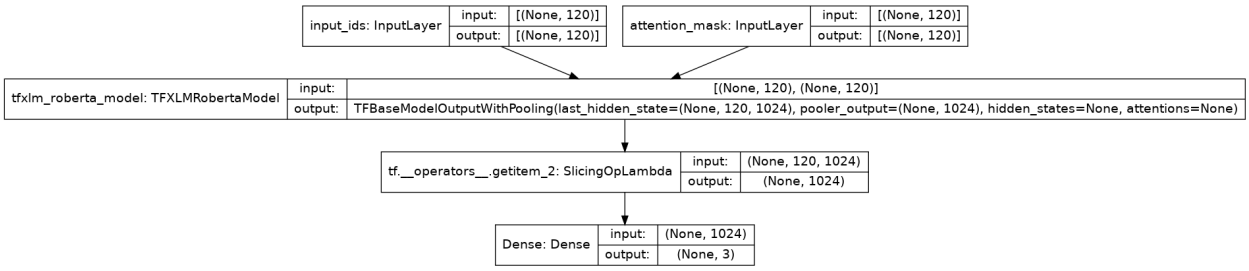
The dataset provided by Kaggle does not have any validation set to monitor the performance of a model during training, so the train set was split into a train subset and a validation subset for evaluation. From the shuffled train set containing 12,120 examples, 9,696 examples are randomly sampled to make up the train subset while the remaining 2,424 examples (e.g. 20% of original train set) are used as the validation subset. We stratify data during train-validation split to preserve the original distribution of the target classes.

Firstly, the baseline models were implemented to compare the initial results with our approach. Supervised learning is used to re-train the M-BERT model on our labeled train subset with a cross-entropy loss function using the following objective,

$$Loss = \sum_{i=1}^{n} t_i log(p_i)$$

Here, $t_i$ is the truth label, $p_i$ is the softmax probability for the $i^{th}$ class, and $n$ is the total number of output classes. During training, the model predictions are calculated from the input data through forward propagation, and the model's parameters are adjusted to minimize the difference between the actual and predicted outputs through backpropagation.

After setting the baselines, the pretrained XLM-R model is fine-tuned on the training subset. The architecture of our classifier is shown in the figure below.




The input matrices (with shape [*batch_size*, *max_sequence_length*]) are passed through the XLM-R model to get output vectors of size 1,024 for each input token. Since this is a sentence classification task, we ignore all except the first vector that is associated with the *[CLS]* token. That vector is then passed as the input to a final dense softmax layer with 3 neurons to get the probability distributions on the output classes. The classifier is trained for a number of epochs, where the trainable parameters of both the XLM-R and classification layers are optimized using cross-entropy objective function.

The validation subset is used on the baseline as well as our model to monitor the validation loss and apply early-stopping during training so that the models don't overfit on the training subset. The following table summarizes the hyper-parameter values of the pretrained models used for the experiments. As we use the larger version of the XLM-R model, it has more trainable parameters compared to the M-BERT model. The model is much larger in scale, consisting of 24 encoder layers, feed-forward networks outputting vectors of size 1,024 hidden units, and 16 multi-attention heads that expand the model's ability to focus on different input positions that can help lead to better encoding of the tokens.

| Model | Parameters | Layers | Hidden Units | Heads | Languages | Vocab Size |
|---|---|---|---|---|---|---|
| M-BERT | 177,853,440 | 12 | 768 | 12 | 104 | 119,547 |
| XLM-R-Large | 559,890,432 | 24 | 1,024 | 16 | 100 | 250,002 |

The additional classification layers increase the total number of trainable parameters by only a few thousand values, summing to 177,855,747 and 559,893,507 parameters to be tuned when training the classifiers based on M-BERT and XLM-R architectures respectively. The classifiers were trained with the following hyperparameter values.

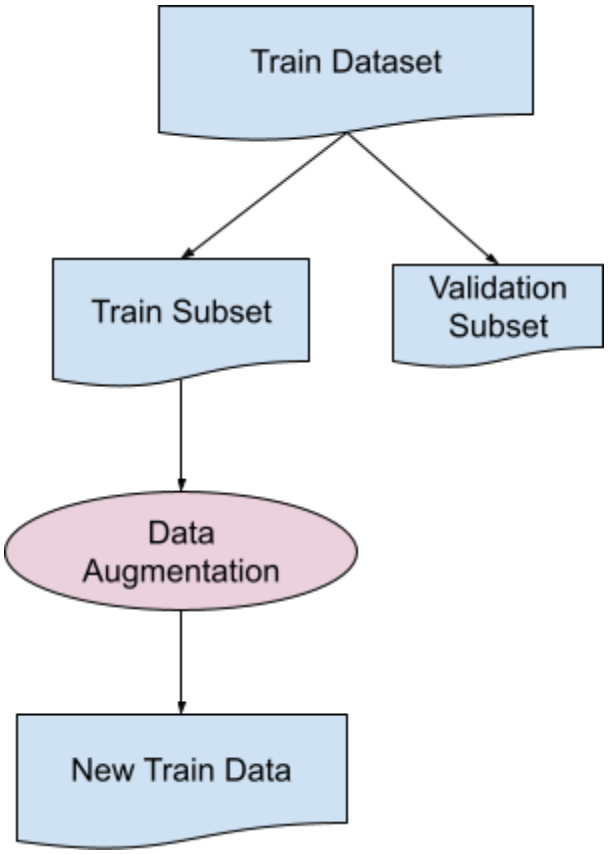
| Epoch | Batch Size | Max Sequence Length | Learning Rate |
|---|---|---|---|
| 4 | 64 | 120 | 1e-5 |

To train these huge models, Tensor Processing Units or TPUs with 8 cores were used. TPUs are hardware accelerators specialized in deep learning tasks and are available to use for free in Kaggle. All the implementations were performed in both Tensorflow and Pytorch frameworks with Python programming language.

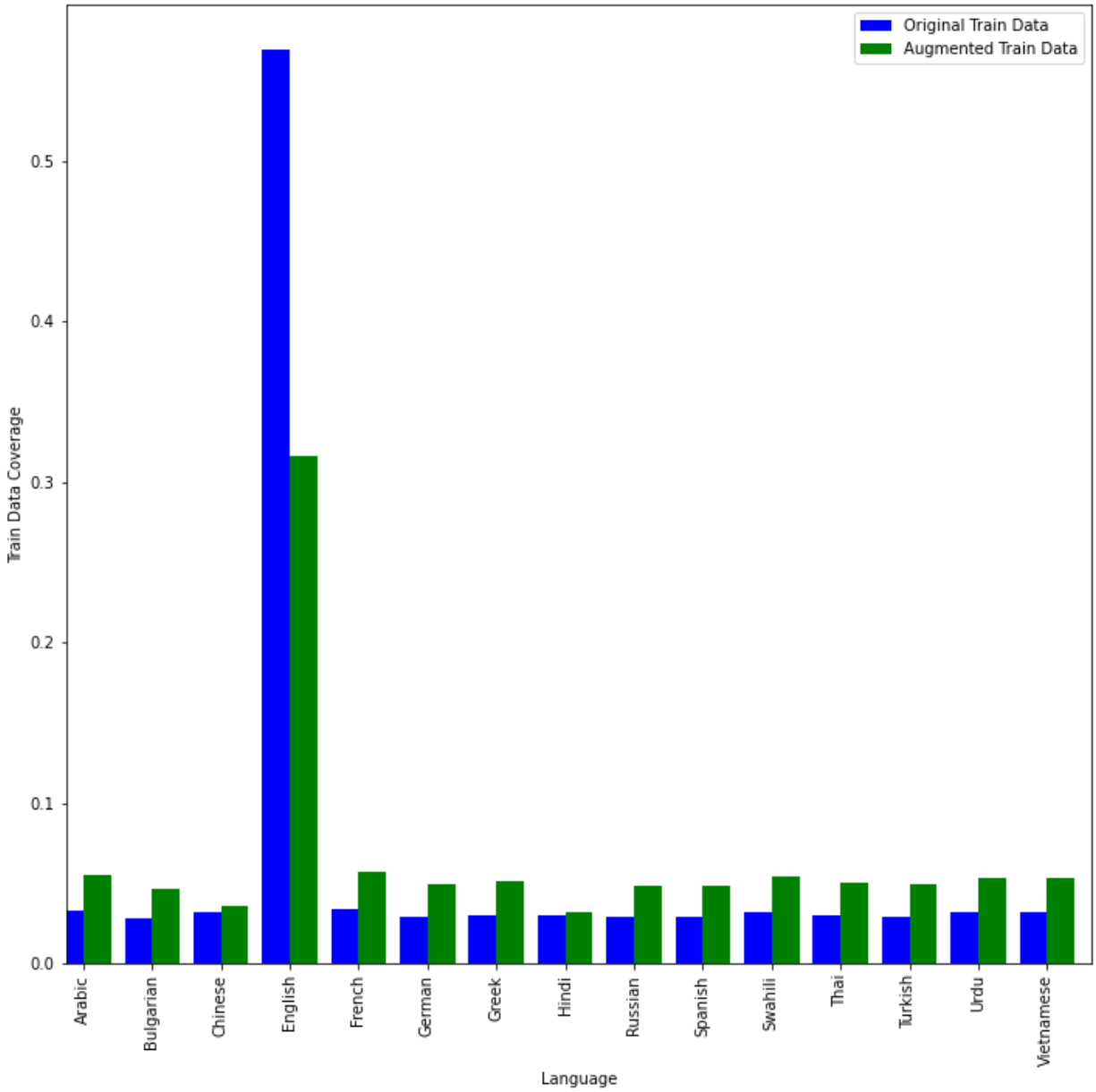
## Refinement

We experiment with other pooling strategies to extract the semantic features of the input pairs from the transformer. We use average pooling where the final transformer layer hidden states of all the input sequence tokens are averaged to get the input vector representation. Another pooling strategy we use is instead of directly passing the *[cls]* token hidden state vector to the classification layer, the vector is further processed by passing it through a fully connected layer with *tanh* activation function.

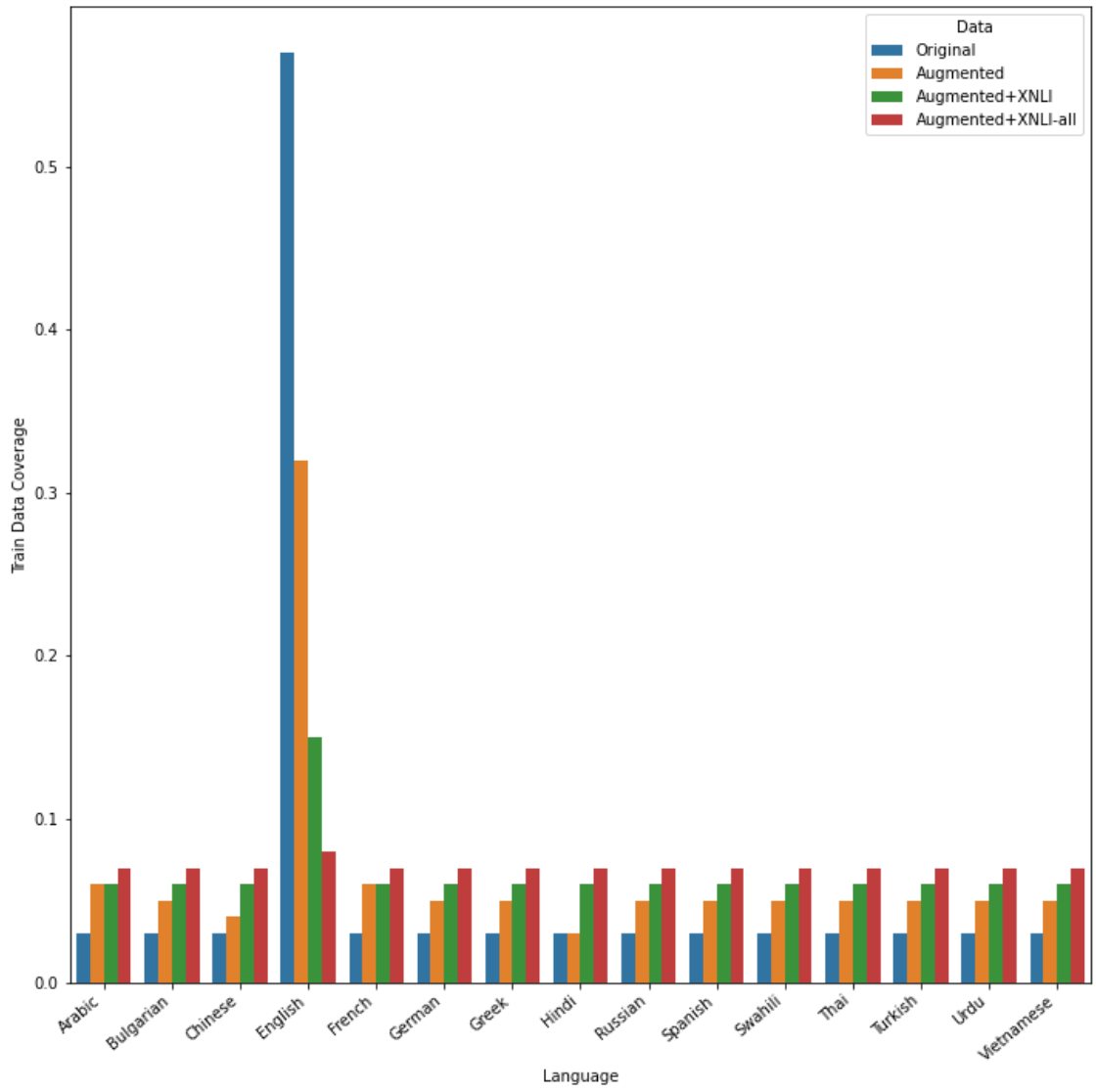
To handle data scarcity in under-represented languages, an augmented dataset is generated before training the models, concatenated with the original train subset, and later fed into data loaders to train the model. The data augmentation process is depicted in the flowchart below.

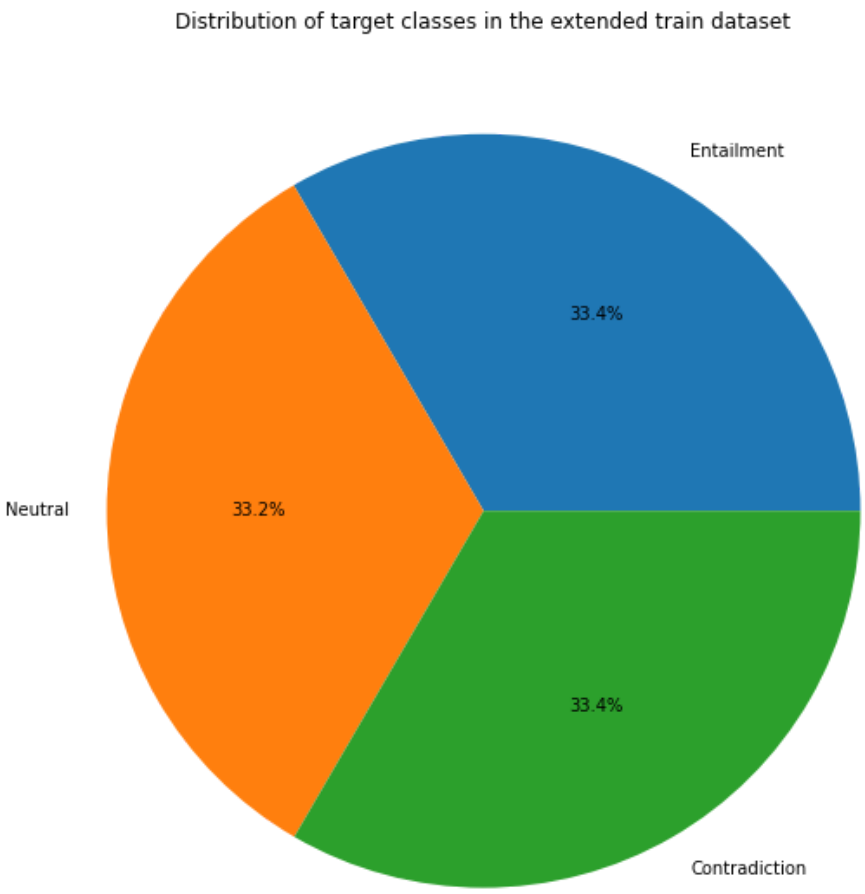


Language translations API like Google Translate is used to generate the back-translations from a mixture of languages. To augment the training data, we use input in the 14 languages with low data coverage and make sure that the new training dataset maintains output class distribution of the original data. From the 4,180 training examples (leaving out English examples), we get a total of 11,946 unique back-translations. While we expected more augmented data, most of the back-translations from the Google Translate API returned the exact same translations of the original examples in the source languages. The chart below compares the distribution of language in train data before and after data augmentation. We can see from the chart that data augmentation greatly reduces the overall train data coverage in English language from ~56% to ~31%, while increasing the number of input samples in all other languages.



We additionally use an auxiliary dataset to check whether increasing the number of samples in the low-resource languages can achieve larger gains overall as well as across all the languages. The Cross-lingual Natural Language Inference[15] (XNLI) corpus is a crowd-sourced collection of 5,000 test and 2,500 validation pairs for the English MultiNLI[16] corpus. The pairs are annotated with textual entailment and translated into 14 languages: French, Spanish, German, Greek, Bulgarian, Russian, Turkish, Arabic, Vietnamese, Thai, Chinese, Hindi, Swahili and Urdu. This results in 112.5k annotated pairs. We conduct a separate experiment where the XLM-R model was fine-tuned on the concatenated train subset, back-translated data and the XNLI validation and test sets. The figure below compares the distribution of languages in the training datasets. Here, XNLI refers to the validation dataset only while XNLI-all contains both the validation and test sets. Extending the original train subset with the augmented and auxiliary data helps to keep a balance in the number of samples per language. The extended train dataset also maintains a fair distribution in the number of samples per class, as shown in the pie chart below.






# Results

## Model Evaluation, Validation, and Justification

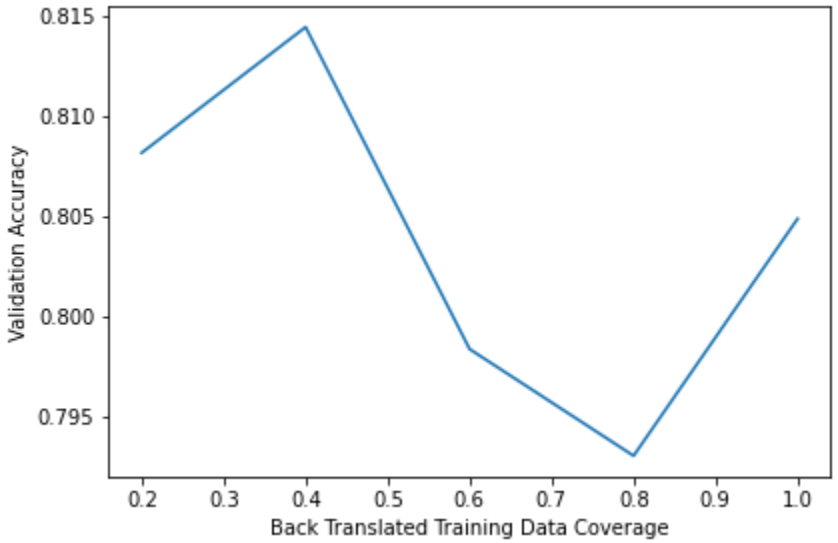
For all the experiments (except random choice baseline), models are trained and validated simultaneously using the train and validation data. The training is stopped when no further decrease in the validation loss is observed and only the best model is saved along with their weights, where a model only gets saved per epoch if it shows higher validation accuracy than the previous epoch.

To get the prediction labels from the softmax layer probabilities, the label of the output class with the maximum probability score from the 3 classes is assigned as the predicted label for each sample. Hence a model prediction is correct when the predicted label value is equal to the ground-truth label. The table below displays all the experiments performed along with their kaggle leaderboard test set accuracy results. The XLM-R model, when trained using only the original train subset, beats the random choice baseline by a large margin and the M-BERT model by a 20% increase in accuracy. Back-translating the training data provides further improvements in accuracy by 3.59%. Fine-tuning the XLM-R model with the auxiliary XNLI data gives the best performance, with a 12.05% boost in the classification accuracy. Overall, passing the features extracted from the *[cls]* token to a fully connected layer before the final layer helps in small gains in the scores.

| **Model** | **Pooling** | **Accuracy** |
|---|---|---|
| Random Choice | - | 0.333 |
| M-BERT | *[CLS]* | 0.650 |
| XLM-R | *[CLS]* | 0.780 |
| XLM-R + Back-Translation | *[CLS]* | 0.803 |
| XLM-R + Back-Translation | *[CLS]* + Fully Connected Layer | 0.808 |
| XLM-R + Back-Translation + XNLI-val | *[CLS]* | 0.821 |
| XLM-R + Back-Translation + XNLI-val | *[CLS]* + Fully Connected Layer | 0.830 |
| XLM-R + Back-Translation + XNLI-all | Average | 0.867 |
| XLM-R + Back-Translation + XNLI-all | *[CLS]* | 0.874 |
| XLM-R + Back-Translation + XNLI-all | *[CLS]* + Fully Connected Layer | 0.874 |

The following figure shows the classification performance on the validation data with access to different amounts of back-translated training data. We observe that the validation accuracy increases from 0.808 to 0.814 when using only 40% of the 11,946 back-translations. The accuracy starts declining with further increase in augmented data coverage and using all the back-translations reduces the accuracy to 0.805. One reason for this decline might be because synthetic examples tend to introduce label noise in data as the performance depends on the quality of the machine translation system. The results also suggest that while synthetic examples help increase the training data, they are not of the same value as human-catered examples. Moreover, the model might learn better from diverse data than from data that are

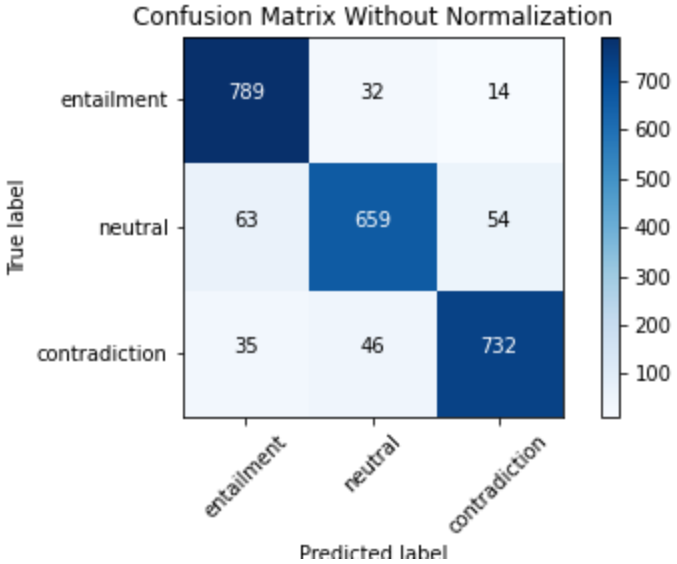
slight variations or paraphrases of the original data. Hence once the model is trained on 40% of the back-translated data, adding more augmented data doesn't help in the classification performance.

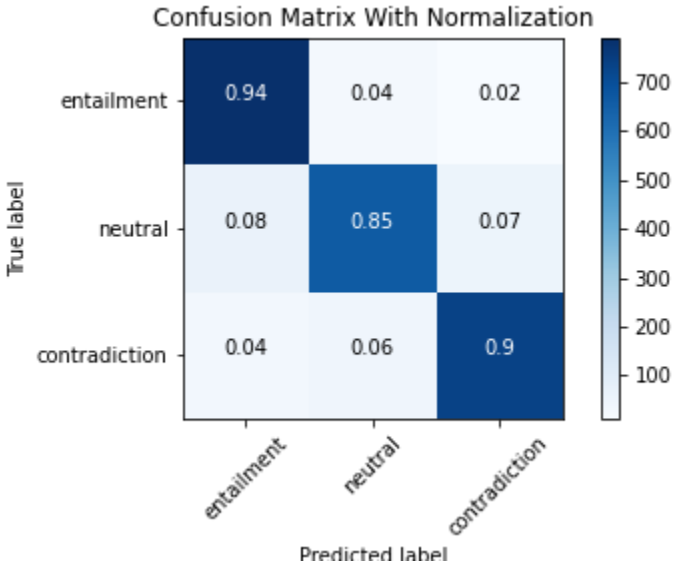



To extract more information about how our best model performs, the table below reports the classification results for our best model using the validation data. The accuracy is 0.90, which means the model is 90% accurate in making a correct prediction from a total of 2,424 predictions. The precision, recall, and f1-score metric values per output class are also presented. Precision refers to the percentage of the results which are relevant, recall refers to the percentage of total relevant results correctly classified by the model, and f1-score is a harmonic mean of precision and recall. From the table, we can see that the model performs decently across all the classes. In particular, the model is really good at predicting *entailment* and *contradiction* classes, presumably because the training data provided by Kaggle itself has more *entailment* and *contradiction* samples than *neutral* samples. For the *entailment* class, the precision value is comparatively lower than the recall value, which suggests that the model is able to classify most of the *entailment* samples correctly but might contain some false positives. Also, the model has relatively low recall value for the *neutral* class, which might indicate the presence of false negatives.

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| **contradiction** | 0.92 | 0.90 | 0.91 | 813 |
| **entailment** | 0.89 | 0.94 | 0.92 | 835 |
| **neutral** | 0.89 | 0.85 | 0.87 | 776 |
| **accuracy** | | | 0.90 | 2424 |
| **macro avg** | 0.90 | 0.90 | 0.90 | 2424 |
| **weighted avg** | 0.90 | 0.90 | 0.90 | 2424 |

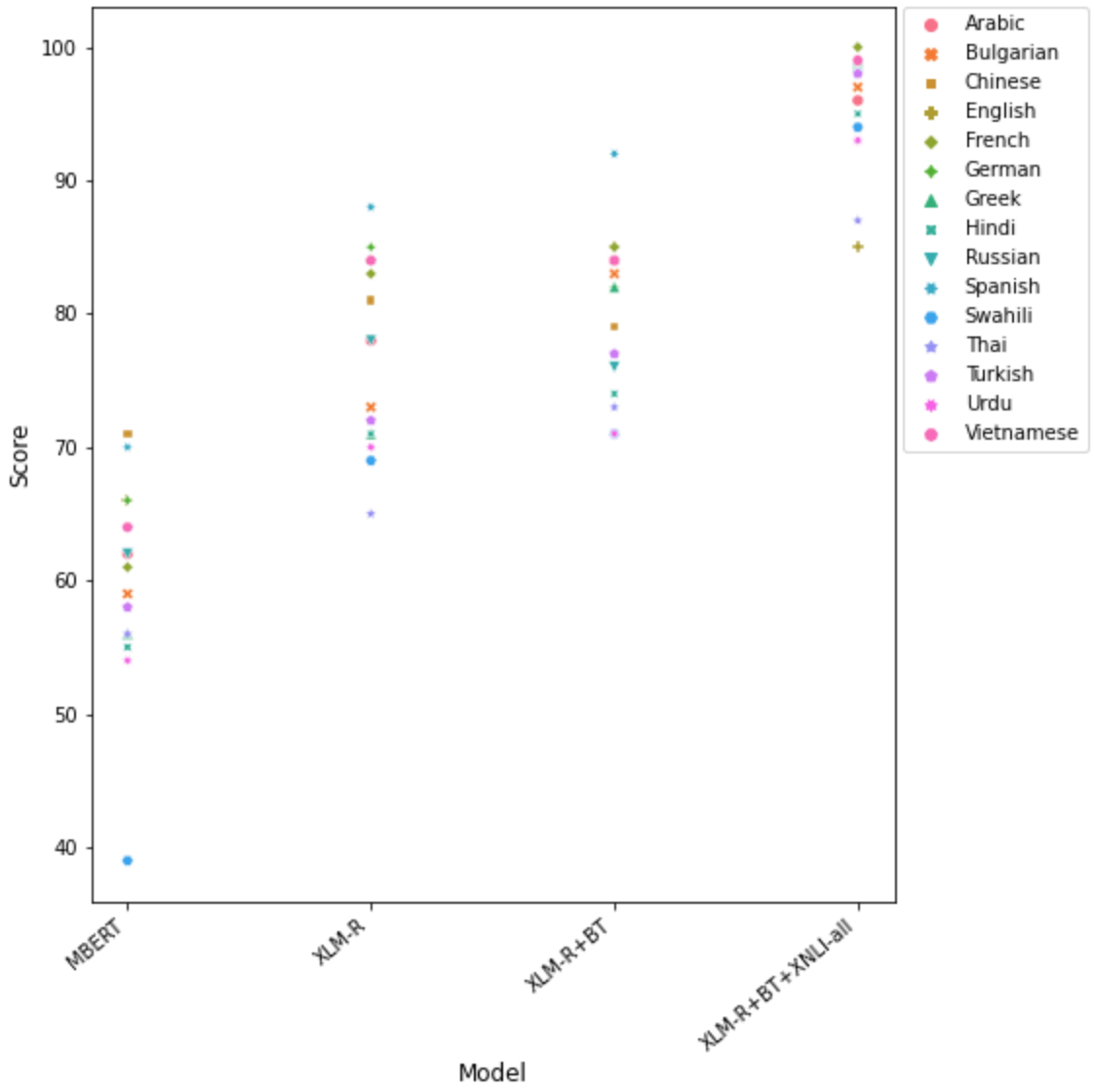
To take a look closer at the false positives and negatives, the normalized and unnormalized confusion matrix plots of the predictions on the validation data is provided below. The confusion matrix helps us visualize whether the model is confused in discriminating between the 3 classes. The model correctly classifies 789 of the *entailment* samples and misclassifies 63 of the *neutral* samples as belonging to the *entailment* class. Out of the 776 *neutral* samples, the model correctly predicts 659 of them, and misclassified 117 of the samples. While the model is good at correctly predicting the *entailment* and *contradiction* samples, it seems to struggle in differentiating between the *neutral* class and rest of the classes. The table below shows some premise-hypothesis examples in the validation data where our model prediction is *entailment* while the ground-truth label is *neutral*. Most of these examples are hard cases in which inference is very probable but not completely certain and would require common knowledge and human-level understanding of language to decipher the vagueness.


Confusion Matrix Without Normalization
entailment
neutral
contradiction
True label
789
32
14
63
659
54
35
46
732
700
600
500
400
300
200
100
entailment
neutral
contradiction
Predicted label


Confusion Matrix With Normalization
entailment
neutral
contradiction
True label
0.94
0.04
0.02
0.08
0.85
0.07
0.04
0.06
0.9
700
600
500
400
300
200
100
entailment
neutral
contradiction
Predicted label

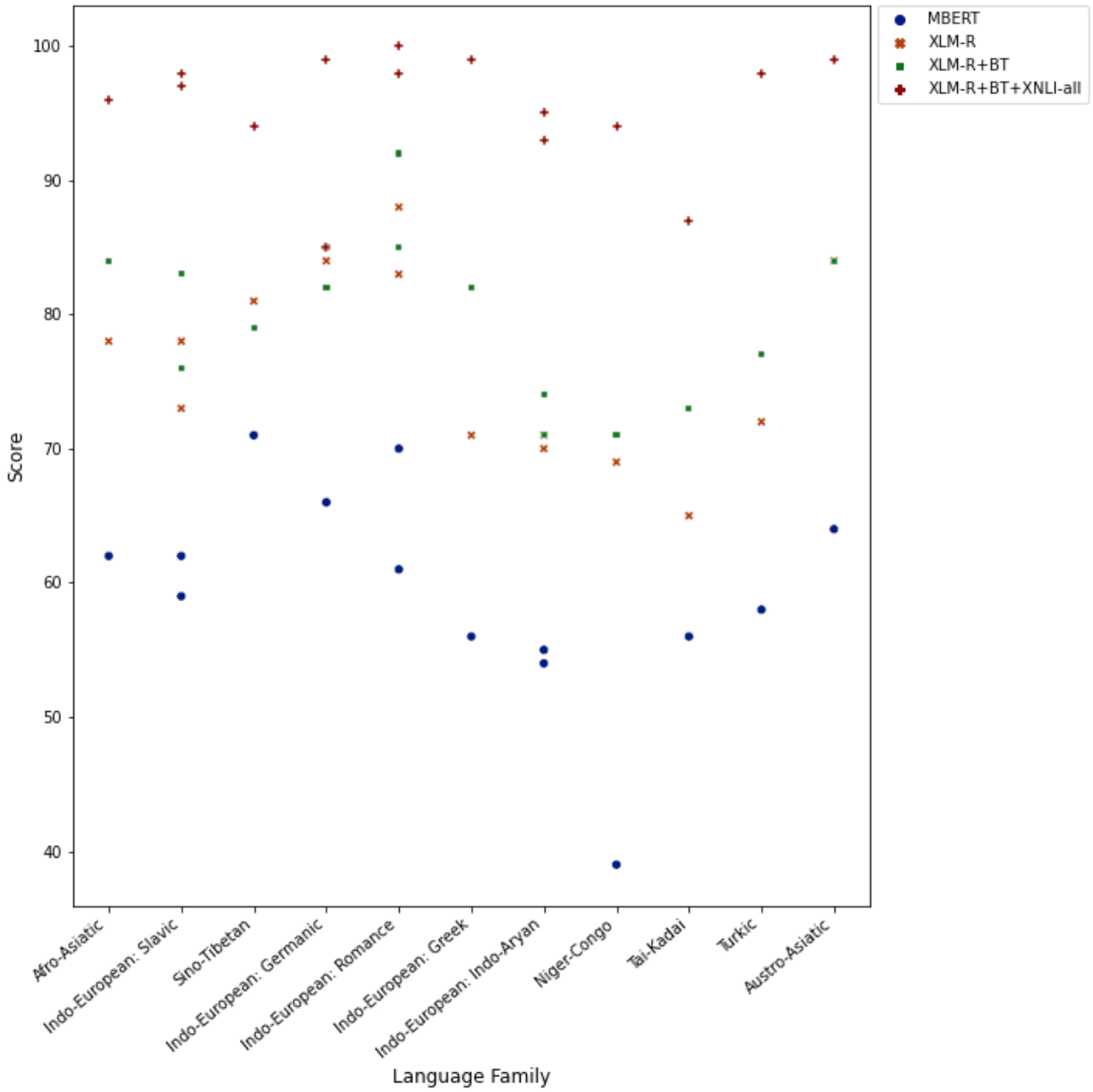
| Premise | Hypothesis |
|---|---|
| *Yet, in the mouths of the white townsfolk of Salisbury, N.C., it sounds convincing.* | *White townsfolk in Salisbury, N.C. are easily convinced of things.* |
| *The final rule contains a Federalism Assessment under Executive Order* | *The final rule had a federalism assessment that was added through executive order by the President.* |
| *Flying at a discount should be more dangerous.* | *Discounted flight deals offered by some travel agents come with an element of risk.* |
| *So uh i hope you like your office* | *I hope you like your new office.* |

The figure below gives an overview of the validation accuracy in percentage across all the languages in each model. On M-BERT, we observe low performance across all the languages and, in particular, low-resource languages have the lowest scores. For example, the model could correctly predict only 39% of the samples in Swahili language. The XLM-R significantly improves upon the baseline across all the languages and we can see that the accuracy of samples in Swahili language went up from 39% to 69%. Augmenting the original train data with back-translations increases the classification performance for most of the languages, however, scores between the diverse languages still have a wider spread. For example, languages like Spanish, English, French, German, and Greek all have accuracy scores greater than 80% while the accuracies for languages like Swahili, Urdu, and Hindi fall in a relatively lower range of 70% to 74%. Using an auxiliary XNLI corpus helps to notably reduce this gap in accuracy between the languages, and we observe that the scores of the majority of the languages now cluster in a relatively small range. The results suggest that the model might benefit more from varied examples present in the XNLI corpus than from back-translations which might contain overlapping features with the original data.



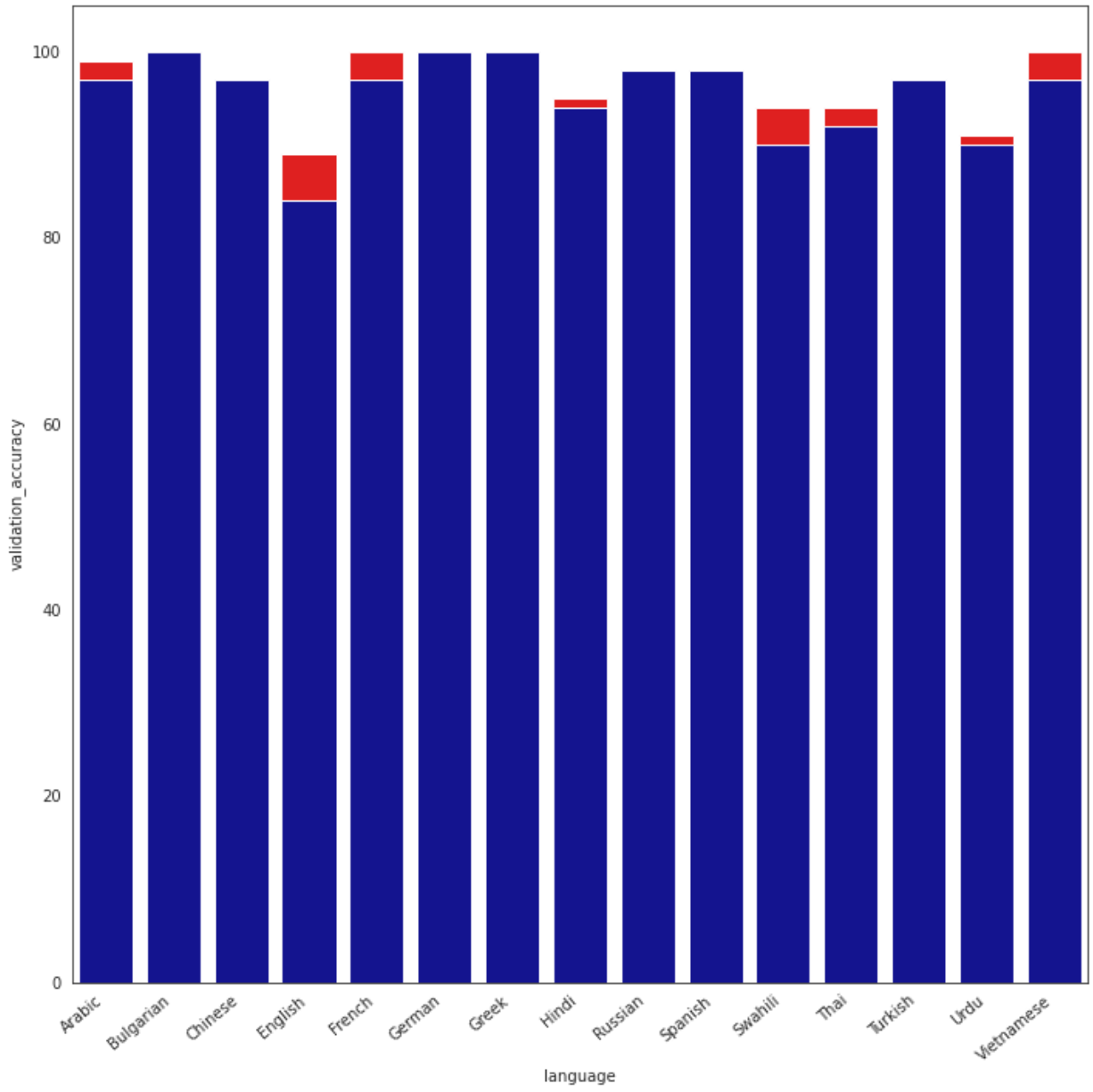

We also analyze the performance of the models based on different language families. A language family is a group of languages related through descent from a common ancestral language or parental language, called the proto-language of that family. The table below shows the 15 languages grouped by language families and the next figure plots the results. We observe that some branches of the Indo-European language family like Romance, Germanic, and Slavic, perform fairly well across all the models. Overall, the difference in performance is the highest between the Indo-European language families and low-resource language families like Niger-Congo and Tai-Kadai. The performance of these low-resource language families improves by a large margin once the original train data is extended by the XNLI corpus and our best model yields at least 95% accuracy for the majority of the under-represented languages.

| Language Family | Language |
|---|---|
| Afro-Asiatic | Arabic |
| Indo-European: Slavic | Bulgarian, Russian |
| Indo-European: Germanic | German, English |
| Indo-European: Greek | Greek |
| Indo-European: Romance | Spanish, French |
| Indo-European: Indo-Aryan | Hindi, Urdu |
| Sino-Tibetan | Chinese |
| Niger-Congo | Swahili |
| Tai-Kadai | Thai |
| Turkic | Turkish |
| Austro-Asiatic | Vietnamese |




In order to check whether low-resource languages can benefit from labeled resources available in other high-resource languages like English, we conduct another experiment by re-training our best model on the MNLI corpus consisting of several thousands of English premise-hypothesis sentence pairs. As expected, the new model improves the validation accuracy for samples in the English language. But interestingly, we also observe an increase in the performance for other languages, as shown in the figure below. Here, the blue bar plots represent the scores per language on our best model while the red bar plots represent the increase in accuracy per language after retraining our best model with the MNLI data. Since many languages have similarities in syntax or vocabulary, the multilingual model might be able to learn the underlying language patterns across languages and transfer the linguistic knowledge to improve performance in languages that do not have copious amounts of training data. This final model

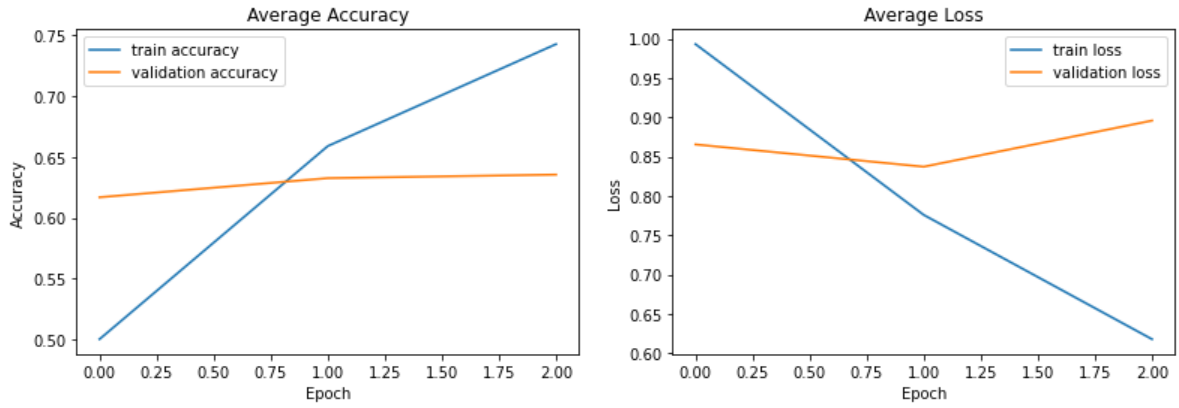
gave an accuracy of 94% on the test data, sending me to the top 3% of the participants in the Kaggle leaderboard, which is my best one so far.



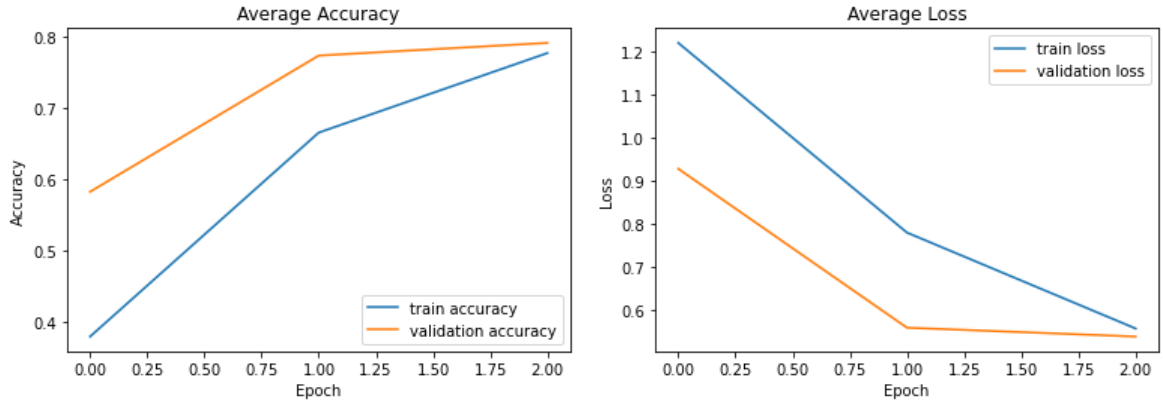
# Conclusion

## Free-Form Visualization

The plots below show the accuracy and loss history of the M-BERT model using the original train data. We can see that there is a huge gap between the training and validation performance - the validation loss keeps decreasing initially but after around 2 epochs, the training loss keeps decreasing while the validation loss keeps increasing instead of decreasing. Clearly this model is overfitting on the training data.

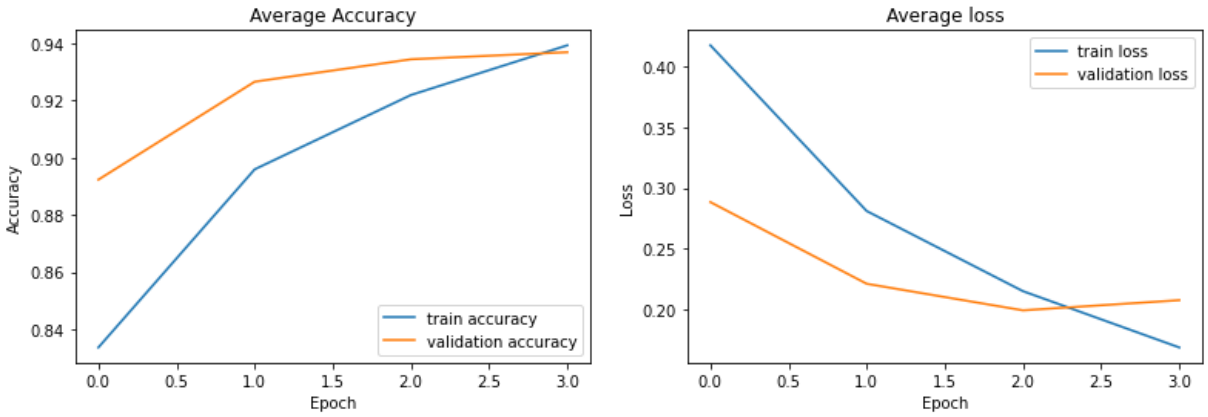



The XLM-R model, fine-tuned on the original train data, is much better at generalizing to unseen data, as depicted in the following plots. The validation loss starts at 0.9272 at epoch 1 and decreases to 0.5382 on the 3rd epoch, resulting in an average validation accuracy of around 0.78.




We also record the accuracy and loss values over 4 epochs on the final model using the augmented and auxiliary datasets. More training examples on the different languages help the XLM-R model reach an average validation accuracy of 0.9369 after the 3rd epoch. As early stopping is applied, once the validation loss starts increasing after the 3rd epoch, the model

saves the best checkpoint at epoch 3. This leads to the best test accuracy score of 94% on the leaderboard.




# Reflection

To find an end-to-end solution for the problem of classifying textual entailment relationship between premise-hypothesis pairs in 15 different languages, I used the following steps:

1. Split and pre-process the original train data into separate train and validation subsets.
2. Create a benchmark for the classifier using M-BERT.
3. Improve upon the baseline performance by using a different model architecture like XLM-R.
4. Experiment with different feature extraction methods and fit the classifier to the training data multiple times until a good set of parameters were found. The validation dataset was used to tune the model hyperparameters.
5. Use data augmentation in the form of back-translation to address the problem of data scarcity in the low-resource languages.
6. Fine-tune the classifier on diverse training examples using publicly available auxiliary datasets like XNLI and MNLI to increase classification performance across all the languages.

One of the difficult aspects of the project for me was to familiarize myself with the different pre-trained language models offered in the Huggingface Transformers library. I used M-BERT and XLM-R for the project, but there were other model architectures and all these models had different configuration settings. I had to spend some time in literature review and experimentation to find a suitable setting and model for the task at hand.

# Improvement

Due to time and resource constraint, it was not possible to conduct more experiments with other pretrained language models such as T5 and different model architectures which might be more

effective. Given enough time and computational power, I'd definitely like to explore the different approaches.

While our best model is good at outputting correct predictions for the *entailment* and *contradiction* classes, it returns many false negatives in case of the *neutral* class. Adding more train data samples in the *neutral* class or using an external knowledge base source might improve model's learning to distinguish between the 3 classes.

Our findings from the results suggest that multilingual models are able to grasp similar language patterns and generalize across languages. Hence, even when the model is fine-tuned on plenty of English data only, the classification performance increases in other languages as well. As a follow up research work, it would be interesting to use various probing tasks as introspection techniques to visualize the various layers in the model and identify what kind of linguistic information is being learned in each layer.

# References


1. https://arxiv.org/pdf/1909.08053v4.pdf
2. https://arxiv.org/pdf/1808.09381v2.pdf
3. https://arxiv.org/pdf/1910.10683v3.pdf
4. https://papers.nips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf
5. https://ai.facebook.com/blog/-xlm-r-state-of-the-art-cross-lingual-understanding-through-self-supervision/
6. https://github.com/google-research/bert/blob/master/multilingual.md
7. https://github.com/google-research/bert
8. http://jalammar.github.io/illustrated-transformer/
9. http://jalammar.github.io/illustrated-bert/
10. https://commoncrawl.org/
11. https://static.googleusercontent.com/media/research.google.com/ja//pubs/archive/37842.pdf
12. https://arxiv.org/pdf/1808.06226.pdf
13. https://www.aclweb.org/anthology/P16-1162.pdf
14. https://arxiv.org/pdf/1804.10959.pdf
15. https://arxiv.org/abs/1809.05053
16. https://arxiv.org/pdf/1704.05426.pdf