**Toxic Comment Classification**

Pranav N Venkit & Zeba Karishma  
Pennsylvania State University  
University Park, PA

**Abstract**

The aim of this project is to classify any given comment to be toxic or not. There are numerous methods to implement the same. Hence our intention is to create a classification model that can efficiently segregate the models to obtain a qualified higher accuracy, in the Kaggle leaderboard. Through this project, we are also introduced to various deep-learning models as well as the current implications of the same, in a technical ground of research. Using this solution, we intend to create a safe space for open communication, in a socio-technological platform.

**1. INTRODUCTION**

The intention of this problem statement is to fundamentally understand the various real-time scenarios of solution with respect to Data Mining. Machine Learning and Artificial Intelligence are the critical fields of research in the current technical ground of development. Many solutions have been established, ranging from technology to economics, through the application of Machine Learning. Hence, we intend to explore the various means of implementation of such methods, using a real-time scenario provided by the Kaggle contest – “Toxic Comment Classification Challenge”.

The problem statement is presented by the Conversation AI team, a research initiative founded by Jigsaw and Google, who are working on tools to help improve online conversations. Like their fundamental motivation, the problem statement expects the teams to construct a kernel that helps improve online conversations by identifying the toxic ones. The challenge requires us to build a multi-headed model that’s capable of detecting different types of toxicity like treats, obscenity, insults, and identity-based hate. The dataset offered consists of comments from Wikipedia’s talk page edits. Through this model improvement, the challenge solution hopes to help improve the online discussion platform to become more productive and respectful.

The data provided is separated into train and test files, each consisting of around 150,000 comments. The train dataset has been labeled by human raters for toxic behaviors. The labels are as follows – toxic, severe\_toxic, obscene, threat, insult and identity\_hate. The test dataset, consisting of only comments, is to be used for identifying the labeled values provide in the train set. The competition also provides a sample submission file, to illustrate the format required to provide the output.

For the output, each id in the test set must be used to predict the probability for each of the six possible types of comment toxicity. The columns are expected to be in the same order as provided in the submission format. The accuracy of the model is calculated on the mean column-wise ROC AUC. In other words, the score is the average of the individual AUCs of each predicted column.

**2. METHODS USED**

From the evaluation of the problem statement, we are familiarized with the fact that the this falls under the subset of “Natural Language Processing”. The sentiment of the texts needs to be identified using few critical words, to determine the toxicity of the sentences.

This project was segregated into three submissions, or checkpoints. Our primary approach was to handle the problem statement using a progressive methodology. We, as a team, were well-versed with the basic concepts of Machine Learning using fundamental techniques such as construction of a Logistic Regression, XGBoost and Discussion Tree. Therefore, for the first checkpoint, we decided to analyze the texts using these methods.

**I. Checkpoint 1**

In the initial approach, the kernel was created using the TFIDF approach for preprocessing and tokenizing the data using TfidfVectorizer function. This was followed by using the basic XGBoost algorithm which provided an accuracy of 96% with a runtime of almost five hours. Due to the large expense of time, a better comparative model was created using Decision Tree and Support Vector Machine approach. Both models provided a small change in accuracy with a small variation in the duration of runtime.

To increase the accuracy of the same, 5-fold cross validation was executed, but this change exponentially increased the run-time, leading to the exclusion of the cross-validation step. This was followed with the implementation of Logistic Regression algorithm as the model is expected to work significantly well when the dependent target is categorial. This model provided a greater local output of 98.02% with significantly lower runtime (40 minutes).

Ensuing this, a kernel was created to understand a novel approach using Keras, as the package is widely known to be efficient in handling text-based data with greater and easier data preprocessing functions. Hence, with the aid of online resources, LSTM Neural Network algorithm was developed using the aforementioned package. This approach provided more significant change in accuracy, as compared to the initial methods tried, but was meaningfully close in performance when compared with the Logistic Regression kernel developed by us. A conclusive understanding was reached in creating an efficient combination of both the better algorithm, to create an ensemble, that can boost the overall performance by weighing previously mislabeled examples with higher weight provided.  We reached a public score of 0.97778 in this checkpoint.

**II. Checkpoint 2**

From the previous checkpoint, we were now elucidated of the fact that ‘normal’ feature classification models will not be able to provide a high accuracy for this problem. For applications related to sentiment analysis, ‘deep learning’ proved to be a better option of implementation, as it is known for success in the same.

We hence focused much of our attention to the various possible iterations of deep learning models that we could implement. But we also realized that we hadn’t performed much text analysis and preprocessing on the comments. There was a huge amount of junk values that were present in the dataset. A lot of foreign languages were encountered, including a large number of special characters and symbols. For the preprocessing step, we used the Google Cloud Translate API [1] to translate the words of each comment to English. This significantly improved the accuracy of each of our model. Following this, we used two significant models, from our previous tests, to create an ensemble.

We were also elucidated on the concept of embedding layers, by going through a lot of blogs [3] and papers. The functionality of including embedding layers such as FastText [2] seemed to be the next progressive step to take. Having a predefined dictionary mentioning the possible iteration and sentiment of each word with respect to others, was indeed required to reduce the giant computational time that was taken by our models.

Hence, as a solution to the first checkpoint, we created two primary kernels. The first kernel was created using keras layer Bidirectional LSTM [4]. It replicates the first recurrent layer so that there are two layers side by side. Input to the 1st layer remains same whereas a reverse copy of the input sequence is provided to the other. It has been used to provide additional context to the network and result in faster and even fuller learning on the problem. This was enhanced by generating a Glove embedding layer [4] to create an embedding matrix that helps in handling and identifying words and their distance with other similar words. Keras embedding layer has been used as the 1st layer in the model which is fed with integer encoded input data, so that each word is represented by unique integer. This data preprocessing is achieved using the Tokenizer API of the keras preprocessing layer. Transformed inputs are passed into Bidirectional LSTM model and Conv1D layer to learn the local pattern among the neighboring words i.e. among multiple word embeddings. In addition, MaxPooling and Average pooling is done to create a summarized version of features detected in the input and finally sigmoid activation is used to create the model of the learned inputs. This model has a running time of 90 minutes with a score of 0.98434.

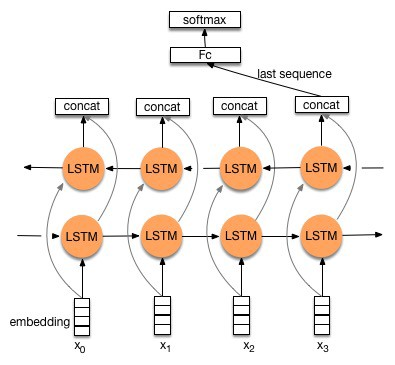


Fig. 1: Bi-LSTM Architecture

The second kernel is created by the combination of two different RNN layers: Bidirectional Gated Recurrent Unit and Bidirectional Long Short Term Memory layer. This model utilizes Fastext, by Facebook, as a part of its embedding layer. Each of the two layers are convolved individually and the results are concatenated after applying the pooling layers. The concatenated results are passed to the dense layer, which in turn utilizes sigmoid activation function to create an output model. Finally, this model is utilized to predict the toxicity of the comments. This model was executed for 3 epochs, each of which executed for 40minutes with a score of 0.98414.

The final model was submitted by creating an ensemble of both the kernels, giving a final accuracy of 0.98537. The weights of the model were heuristically added as there wasn’t a know tuning method to provide a more relevant weight to each of the model.

**III. Checkpoint 3**

The best performance that was concluded by us is an ensemble of eight different kernel. All the procedures tried for implementation were, as a combination, used to achieve a greater score through the ensemble method.

In this iteration of the kernel, more effort was provided to the preprocessing of data.

A few characters were present in shorthand representation, which had to be recovered. Hence this was executed with normal data parsing, to convert the words to its original representation. (E.g.: converting “haven’t” to “have not”)

The whole dataset was also converted to lowercase representation for cleaner implementation, followed by tokenizing them into vectors.

The first two models were created by using the same models from the previous checkpoints. As the deep learning model of LSTM was being explored, we tried simplifying the same by executing a GRU. GRU stands for a Gated Recurrent Unit, which is a more simplified form of a Bi-LSTM model as it contains lesser gates of implementation [5]. The GRU model provided an accuracy of 0.98302. The GRU model usually provides greater accuracy as compared to the Bi-LSTM model but as our dataset was very centric to toxic words, the LSTM model proved to be a better option to execute.

Consequently, we executed the next model by combining both the LSTM and GRU model by implementing the FastText embedding layer. This increased the score to 0.98444. But the execution of this model took almost 6 hours. Hence to reduce the runtime of the same, we tried implementing the GPU intensive LSTM and GRU library. This proved to be a useful endeavor as the runtime significantly reduced, as compared to the previous iteration. Due to the computational advantage that was obtained, we added both the embedding layers to this model, i.e. FastText and Glove. The execution of this model proved to be the highest score, as a single kernel, as it gave us a score of 0.98549.

To increase the ensemble score, having more compatible models always prove to be useful. Hence, our team explored more optimized deep learning models. This led us to the discovery of the BERT model [7]. BERT, or Bidirectional Embedding Representations, is a new method of pre-training language representations which obtains state-of-the-art results on a wide array of Natural Language Processing (NLP) tasks. Standard conditional language models can only be trained left-to-right or right-to-left, since bidirectional conditioning would allow each word to indirectly “see itself” in a multi-layered context. To solve this problem, BERT uses “MASKING” [7] technique to mask out some of the words in the input and then condition each word bidirectionally to predict the masked words.

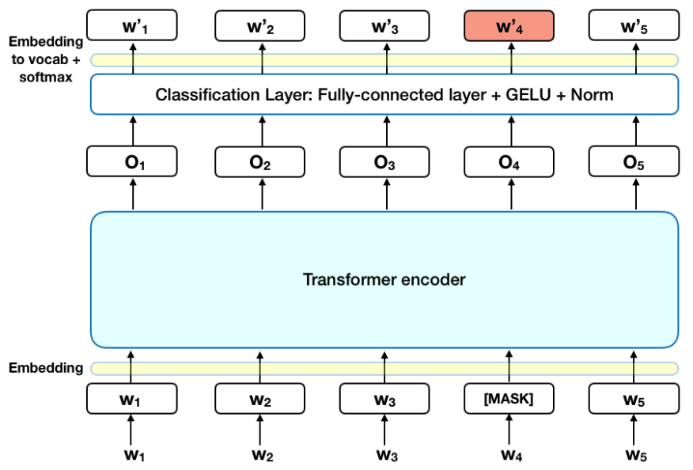


Fig 3. BERT implementation layer

Among the multiple BERT pre-trained models that are available, we have used BERT-Large uncased dataset, having 24-layer, 1024-hidden, 16-heads, 340M parameters [8].

We have done the tokenization using manually downloaded BERT\_INIT\_CHKPNT and BERT\_VOCAB files [8]. To convert data into a format in which understands, we create example objects and then we convert them into features understandable by BERT. By just running, 2 epochs of this model, we were able to achieve a score of 0.98532.

We also explored more into the CNN and LightGBM model, to provide a more diverse set of scores, to obtain a final ensemble. The seventh kernel was created using the LightGBM algorithm as the gradient boosting method, similar to the XGboost method, uses tree-based learning. This implementation was known to be faster and more accurate than its predecessor. To effectively implement the algorithm, we preprocessed the data, once again, to remove punctuation and empty spaces. Shorthand words were converted to its original form, for easier implementation.

Following the preprocessing step, the words are split into its primitive form by using the split function. This reduces the various forms of words to a singular representation. The sentence was then segregated into bags of characters using the n-gram function. The TF-IDF value of each bag and word is calculated and placed with the dataset. Each dataset is converted to its vector representation with its TF-IDF value and passed to the LightGBM model.

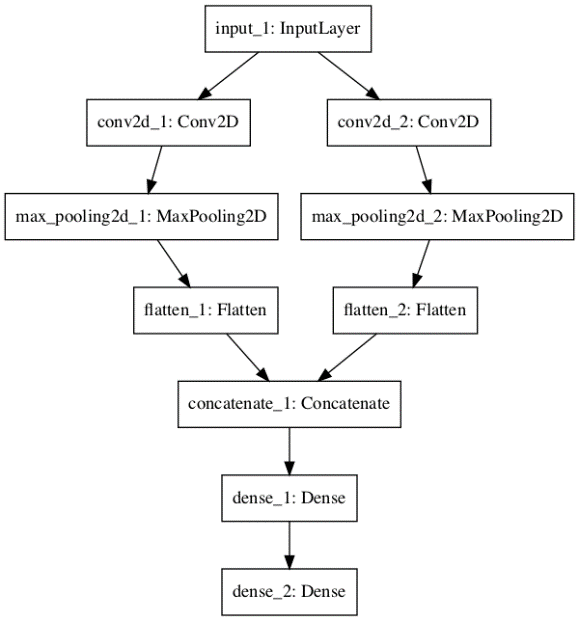


Fig. 2: CNN Model Architecture

The Convoluted Neural Network model provided us a score of 0.98334 while the LightGBM model provided a score of 0.97926. The latter score is significantly lower as compared to the previous scores mentioned as the dataset doesn’t respond well with the basic machine learning classifier models. The final ensemble model achieved through the combination of all the eight models came up to 0.98810, which was our final score for this project.

**3. RESULTS**

The final result obtained was a heuristically created ensemble, where the maximum weight was given to the Bi-LSTM-GRU and the BERT deep learning model. All of the above algorithms were decided to be used as a combination, creating a final kernel that provides an approach that allows the production of better predictive performance compared to a single model. This model was created in order to decrease variance and bias by invariably improving the prediction. Multiple weighted average combinations of all the algorithms were tried and a combined weightage of all the mentioned algorithms produced the best results. This was hence the final method that is used by our team to provide the final submission. The score provided by each model is mentioned in the below table.

|  |  |  |
| --- | --- | --- |
| **Algorithm Implemented** | **Public Score** | **Private Score** |
| Project Submission- 1 | 0.97870 | 0.97778 |
| Project Submission- 2 | 0.98537 | 0.98371 |
| Bidirectional- Glove embed | 0.98434 | 0.98234 |
| GRU- Fastext embed | 0.98302 | 0.98169 |
| Bidirectional+GRU- Fastext embed | ​0.98414 | 0.98314 |
| CNN + Preprocessed Data+ Glove | 0.98334 | 0.98264 |
| Logistic Regression + Preprocessed Data | 0.97565 | 0.97323 |
| LightGBM + Preprocessed Data | 0.97926 | 0.97833 |
| OOF Stacking | 0.97920 | 0.97833 |
| BERT | 0.98532 | 0.98519 |
| LSTM+GRU (GPU intensive) | 0.98627 | 0.98549 |
| **Final Ensemble** | **0.98810** | **0.98759** |

Table 1: Output table per model

The final Kaggle project submission is as shown below. The ensemble public score comes to 0.98810 while the private score is of 0.98759. This model was achieved due to the combined effective combination of all the models explained above.

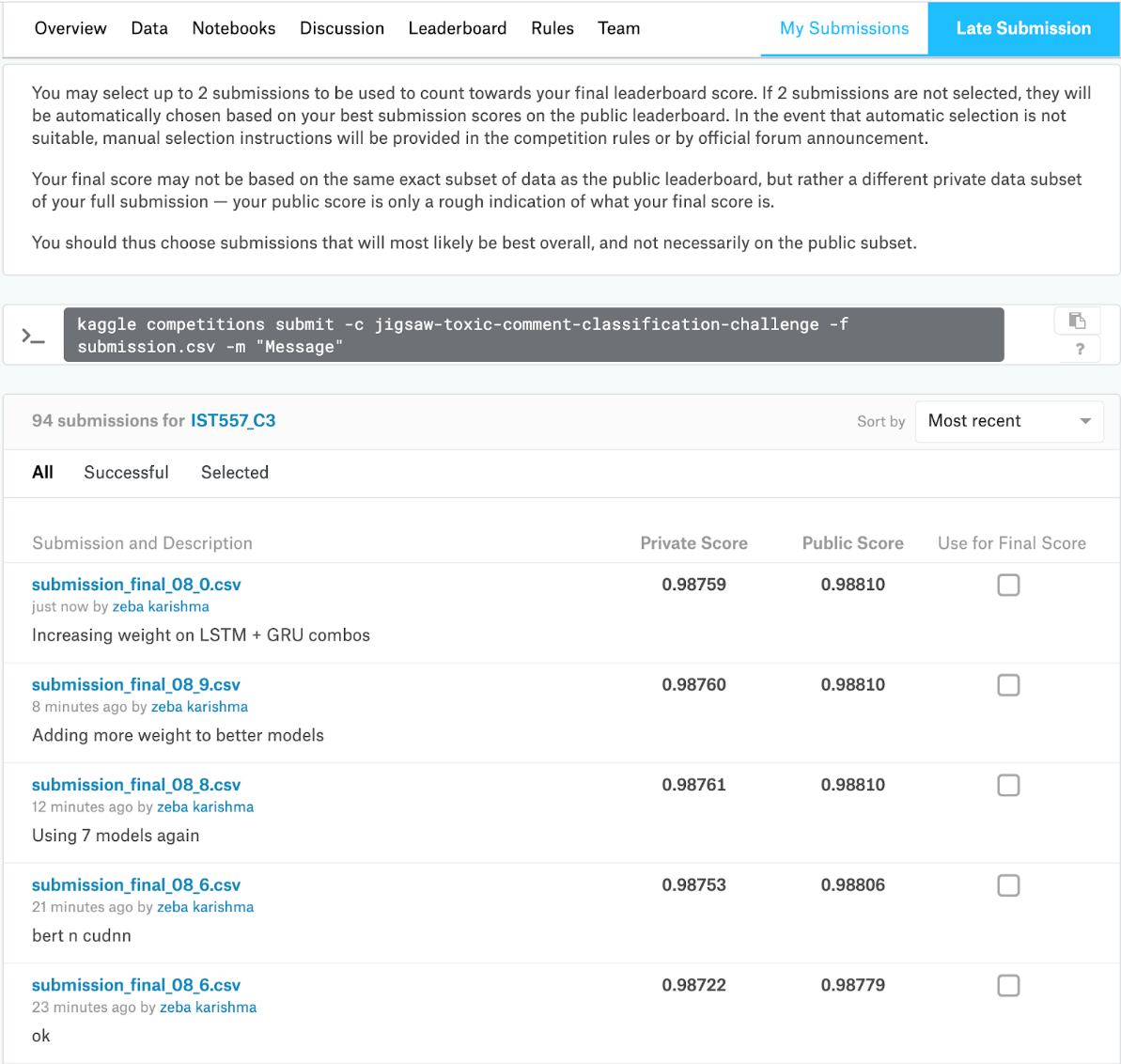
**

Fig 3: Kaggle Submission Page

From the learnings of the 1st and the 2nd iteration, it was concluded that using deep learning models enable the result to be more accurate. Hence most of the models executed by our team was an effective deep learning implementation. From the previous iteration of the project, we also concluded the effectiveness of using an embedded layer in the kernel. In our dataset, Bi-LSTM model fared better as compared to the other classifier models as the data consisted of a series of words. To find the probabilistic value of the sentence to be toxic or not requires the assessment of each word with respect to the other word in the statement. A sentence may not necessarily be toxic even if there are ‘curse’ words present in the statement. Therefore, the model is required to have a memory unit to judge the presence of other words present in the statement. This was well implemented by the Bi-LSTM model.

Our superlative model from all the collection of implemented models is credited to the ‘Bi-LSTM-GRU-Double Embedded’ model. Due to the execution of reinforcement learning and the execution of a double embedded layer, the model was pretrained with a significant collection of possible words in the English language. The model was also aware of the other similar occurrence of words due to the embeds. With the presence of reinforcement learning, the model was able to understand the connotative meaning of each word with respect to the whole sentence. Due to these characteristics, this individual model catalyzed the ensemble to provide a greater result.

We also reproached the fundamental data-mining algorithms, using the newly achieved preprocessed data. The XGBoost and Logistic regression model did not prove to be very useful in achieving a greater score. This led to the discovery of the LightGBM kernel, which was a lighter version of the XGBoost algorithm. This method achieved an overall public score of 97.9. This was invariably added to the overall kernel as well. We tried to implement OOF, Out Of Fold, stacking method through the already implemented methods (LSTM, Logistic Regression and CNN). This method utilizes cross validation to get predictions on the training dataset and uses it to learn the model. But the accuracy achieved was too less (97.4) to be used in the ensemble, hence the implementation was discarded.

**4. SUMMARY**

Through the Kaggle challenge- “Jigsaw Toxic Comment Classification”, we got to truly understand the impact of sentiment analysis on real-time data. It was indeed intriguing to understand and contemplate the effect such methods can have in perspective of a global solution. The dataset used consisted of comments from Wikipedia. As the solution correlated to identifying if a comment was toxic or not, the application used was a subset to Natural Language Processing.

This challenge invariably provided a platform to Deep Learning methodologies as fundamental Machine Learning algorithms were not as efficient as compared to the latter. Hence, we were able to learn and implement various novel kernels such as LSTM and CNN. The most accurate kernel for this implementation, by us, is credited to the recently developed BERT implementation. Bidirectional LSTM and Convoluted Neural Network also led to a good throughput, and understanding the neural network implications of such algorithms were enlightening as well. The combination of both Bi-LSTM and GRU, using both the embedding layers and adaptive learning rate, provided a greater score of accuracy as compared to the BERT model as well.

The biggest challenge, for us, was processing the dataset. A lot of comments required detailed processing before it could be used in the program. The true impact of preprocessing was understood by comparing the results acquired before and after the preprocessing layer. The margin of improvement was indeed drastic. Translation and conversion of junk characters were also implemented to acquire a better understanding of the data.

But the principal aspect in the implementing was credited to ensemble learning. The Ensemble method is a machine learning technique that combines several base models in order to produce one optimal predictive model. The systematic execution of every single method led to a greater solution due to the ensemble. This was the solution to the final kernel submitted by our project.

As a team who was novel to the concepts of Natural Language Processing, it was indeed helpful in finding the various sources available online. The Kaggle discussion page had very enlightening outlook to the possible approaches to the solutions as well. Through this competition, we also understood the importance of resource consumption. A few models, such as BERT, took 2 days to execute. Having numerous model that needed to be implemented, within the limited time, it was essential to have a thorough plan. Situations such as this taught the importance of coordination and teamwork.

We found the usage of Google Colaboratory [9] to be very valuable throughout this venture. Having an online platform, that provided additional CPU and RAM, was indeed convenient to run a few more iterations of the kernel.

The future plan of this project is to understand how one can implement hyperparameter tuning using ensemble learning. We had to create our ensemble using the conventional ‘trial and test’ model, which wasn’t exactly effective. In the final phase of the project, we discovered the stochastic algorithm of ‘Hill climb ensemble’ [10]. We hence want to explore the possible execution of the same, to understand if a better result can be achieved. We were not able to implement effective parameter tuning to all the individual models. This will be marked as the ensuing future plan for this computational effort.

**4. TEAM CONTRIBUTION**

It is tough to precisely segregate work contributed by the individual members of the team as this project was an effective output of both the members alike. But we shall be providing a very brief segregation, for better organization of this project.

Pranav worked on the implementation of the Bidirectional LSTM model with glove, GRU\_FastText Model, LightGBM, CNN model, XGBoost and Logistic Regression Kernel. He also worked on the creation and implementation of the Preprocessing phase of the dataset.

Zeba worked on the Keras preprocessing, BERT model, fastText and Bidirectional LSTM-  GRU (CuDNN) model and OOF Stacking Kernel. The Ensemble Learning algorithm was implemented by her as well.

Algorithm optimizations were explored by both of the team members and hence ensemble was a combined decision to give a better accuracy. The other various model creation and testing were done by the team as a whole.

**5. REFERENCES**

1. Aiken, Milam, and Shilpa Balan. "An analysis of Google Translate accuracy." *Translation journal* 16.2 (2011): 1-3.
2. Joulin, Armand, et al. "Fasttext. zip: Compressing text classification models." *arXiv preprint arXiv:1612.03651* (2016).
3. Medium, Deep Learning For Embedding Layers, blog

< <https://towardsdatascience.com/deep-learning-4-embedding-layers-f9a02d55ac12> >

1. Pennington, Jeffrey, Richard Socher, and Christopher Manning. "Glove: Global vectors for word representation." *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 2014
2. Kaggle, Improved LSTM Baseline Glove Dropout, Kaggle Notebook

< <https://www.kaggle.com/jhoward/improved-lstm-baseline-glove-dropout> >

1. Chung, Junyoung, et al. "Empirical evaluation of gated recurrent neural networks on sequence modeling." *arXiv preprint arXiv:1412.3555* (2014).
2. Devlin, Jacob, et al. "Bert: Pre-training of deep bidirectional transformers for language understanding." *arXiv preprint arXiv:1810.04805* (2018).
3. Github, BERT, computer software

< <https://github.com/google-research/bert> >

1. Bisong, Ekaba. "Google Colaboratory." *Building Machine Learning and Deep Learning Models on Google Cloud Platform*. Apress, Berkeley, CA, 2019. 59-64.
2. Caruana, Rich, Art Munson, and Alexandru Niculescu-Mizil. "Getting the most out of ensemble selection." *Sixth International Conference on Data Mining (ICDM'06)*. IEEE, 2006.