StumbleUpon Challenge

Thanks for giving me the opportunity to participate in the job hackathon. Here is my approach to the problem:

Quick Highlights

1. The model is made in PyTorch.
2. Due to time constraints, only text is used for classification. Other numerical features are discarded.
3. BERT[1] is used to encode text into embeddings.
4. A library, huggingface[2], is used to quickly import model architecture for agile prototyping.
5. Latest weights and biases[3] run: https://wandb.ai/rahools/stumbleupon/runs/3gzyqkk2

   WandB helps us store all the artifacts generated by the notebook. I have stored:
   a. Each epochs model checkpoint
   b. Model Hyperparameters
   c. Validation report
   d. Final submission CSV

6. Kaggle Notebook: https://www.kaggle.com/rahools/huggingface-bert-model

Report Highlights

All of the metrics calculated here are based on a validation set, which is 15% of the whole training dataset provided by the challenge. The validation set doesn't appear in my model's training dataset.

1. Class Ephemeral
   a. Precision: .74
   b. Recall: .83
2. Class Evergreen
   a. Precision: .82
   b. Recal: .72

F1 Score: .778
Kaggle Leaderboard Score: 0.78864

Data Preprocessing

1. Dataset is divided into two parts: train(85%) and validation(15%)
2. JSON in the column named 'boilerplate' is parsed and made into a python dictionary.
3. The parsed dictionary helps to assign a new column 'body', storing text contained by the 'body' key in the new column. If the 'body' key is empty, the 'title' key is used.

4. BERT has a limit of 512 tokens/words it can generate embeddings for. If a longer sentence is passed BERT automatically truncates the sentence and processes the first half. In our case, this might be detrimental because our model classifier articles and starting of the article is generally a hook for readers and might be a tangent to the article's content. Therefore, I planned to manually truncate the articles to include the last 512 words. Why from the last? Mostly end section of the article is a conclusion of the article by the author, hence indicative of the article's content.
5. Passed above generated truncated text to BERT's pretrained tokenizer and attention mask generator. Tokenizer basically formats input text into a format that the BERT model likes to work with.
6. Generated a PyTorch compliant Dataset and Dataloader. Dataloaders are super fancy python generators that provide us with data, be it at the time of training or validation.

Model Architecture

For saving time and focus more on hyperparameter tuning, I decided to go with a pre-defined model architecture based on a pretrained BERT encoder model from the huggingface transformer library called BertForSequenceClassification[4]. The model pipeline looks like:

1. Input generated from the tokenizer and attention mask generator is feed into a pretrained BERT encoder model which returns the input's text embeddings.
2. Text embeddings from the BERT model are then passed from a dropout layer onto a final prediction linear layer.

BERT model -> Dropout Layer -> Linear Layer

Passing text input to our model results in it generating output logits. Now the logits could be used to generate the output label probabilities by applying softmax.

References

1. https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270
2. https://huggingface.co/transformers/
3. https://wandb.ai/
4. https://huggingface.co/transformers/model_doc/bert.html#bertforsequenceclassification
5.