Report of Toxic Spans Detection

Last Name : Giri

First Name : Surya

CWID : 10475010

Course : 2022F CS 584 A

Purpose : Final Project - Task 2: Toxic Spans Detection

1. Introduction

One of the most often discussed subjects in the history of computational linguistics has been sequence labeling. With the formal description of the grammar, problems like Dependency Parsing, Word Sense Disambiguation, and Sequence Labeling, among others, emerged. A task in natural language processing is sequence labeling. Each token (word) is intended to be categorized in class space C. This categorization strategy may be independent (where each word is handled separately) or dependent (each word is dependent on other words). In our case, the spans can be long, and a word may or may not be toxic depending on other words, and not necessarily just one word. Thus, we can predict which part of a sentence or an entire sequence is toxic, instead of identifying just the toxic words. This helps us detect negative sentences even if it contains not many negative words.

Transfer learning is a machine learning method where a model developed for a task is reused as the starting point for a model on a second task. Pre-trained models are frequently utilized as the foundation for deep learning tasks in computer vision and natural language processing because they save both time and money compared to developing neural network models from scratch and because they perform vastly better on related tasks. Therefore, I am using Electra model to generate pre-trained word embeddings which not only saves time in pre-processing, but also generates pre-trained embeddings which results in better Name Entity Recognition and thus better predictions. Electra has been chosen for its efficiency with memory management and speed.

Named entity recognition (NER), also known as entity chunking, extraction, or identification, is the process of locating and classifying significant pieces of data (entities) in text. Any word or group of words that constantly refers to the same item is considered to be an entity. Each recognized object is put into a specific category. NER machine learning (ML) models, for instance, may identify the word "super.AI" in a text and categorize it as a "Company."

Natural language processing (NLP), a branch of artificial intelligence, includes NER. NLP is concerned with the processing and analysis of natural language by computers, which refers to any language that has emerged naturally as opposed to artificially, like coding languages.

2. Problem Formulation

Problem Statement:

Given a sentence, you need to extract a list of indexes that point to toxic phrases in this sentence.

Word Embeddings:

To make this prediction, we train the model on the word embeddings generated from the training data. ML algorithms do not work with words but rather, they understand the embeddings generated from the words. Word embeddings are a sort of word representation that allows for the depiction of words with comparable meanings.

Tokenization:

Tokenization is the process of exchanging sensitive data for nonsensitive data called "tokens" that can be used in a database or internal system without bringing it into scope. Although the tokens are unrelated values, they retain certain elements of the original data commonly length or format so they can be used for uninterrupted business operations. The original sensitive data is then safely stored outside of the organization's internal systems. We do tokenization here using Electra transformer from Tensorflow. "The ELECTRA model was proposed in the paper ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators. ELECTRA is a new pretraining approach which trains two transformer models: the generator and the discriminator. The generator's role is to replace tokens in a sequence, and is therefore trained as a masked language model. The discriminator, which is the model we're interested in, tries to identify which tokens were replaced by the generator in the sequence." (ELECTRA (huggingface.co)

Conditional Random Fields:

There are two main classifications for machine learning models: generative and discriminative. As a sort of discriminative classifier, Conditional Random Fields model the distinction between the various classes. On the other hand, generative models simulate the creation of the data, which, after learning, can be utilized to construct classifications. Naive Bayes, a straightforward and well-liked probabilistic classifier, is an example of a generative algorithm, and Logistic Regression, a classifier based on maximum likelihood estimate, is an example of a discriminative model. In our case since the task is sequence labelling, which includes multiple words instead of just one, we have to use discriminative models; and as such we use CRF. We model the Conditional Distribution as follows:

$$\hat{y} = argmax_y P(y|x)$$

Conditional Distribution

Probability distribution:

$$P(y, X, \lambda) = \frac{1}{Z(X)} exp\{\sum_{i=1}^{n} \sum_{j} \lambda_j f_i(X, i, y_{i-1}, y_i)\}$$

$$\text{Where: } Z(x) = \sum_{y' \in y} \sum_{i=1}^{n} \sum_{j} \lambda_j f_i(X, i, y'_{i-1}, y'_i)$$

Probability Distribution for Conditional Random Fields

Negative Log-Likelyhood:

$$L(y, X, \lambda) = -log\{\prod_{k=1}^{m} P(y^k | x^k, \lambda)\}$$

$$= -\sum_{k=1}^{m} log[\frac{1}{Z(x_m)} exp\{\sum_{i=1}^{n} \sum_{j} \lambda_j f_j(X^m, i, y^k_{i-1}, y^k_i)]$$

Negative Log Liklihood of the CRF Probability Distribution

Partial Derivative with respect to Lambda:

$$\frac{\partial L(X, y, \lambda)}{\partial \lambda} = \frac{-1}{m} \sum_{k=1}^{m} F_j(y^k, x^k) + \sum_{k=1}^{m} p(y | x^k, \lambda) F_j(y, x^k)$$

$$\text{Where: } F_j(y, x) = \sum_{i=1}^{n} f_i(X, i, y_{i-1}, y_i)$$

Partial Derivative w.r.t. lambda

Finally, Gradient Descent update equation for CRF:

$$\lambda = \lambda + \alpha[\sum_{k=1}^{m} F_j(y^k, x^k) + \sum_{k=1}^{m} p(y | x^k, \lambda) F_j(y, x^k)]$$

Gradient Descent Update Equation for CRF

Evaluation metrics:

- F1 score for set.

Given a sentence, let denote the ground truth of spans and denote the prediction set of spans. The precision is calculated as below:

$$P = \frac{|A \cap G|}{|A|}.$$

The recall is calculated as below:

$$R = \frac{|A \cap G|}{|G|}.$$

The F1 score is calculated as below:

$$F_1 = \frac{2 \times P \times R}{P + R} = \frac{2 \times |A \cap G|}{|A| + |G|}.$$

Here, $|\cdot|$ denotes the cardinality of a set. Specially, if $|G| = 0$, we let $F_1 = 1$ if $|A|$ is also 0, otherwise $F_1 = 0$.

Finally, the F_1 score of the entire dataset is an average of F_1 score of all sentences.

3. Method

a. Load Dataset:
Our dataset is divided into two sets: training data (https://github.com/ipavlopoulos/toxic_spans/blob/master/SemEval2021/data/tsd_train.csv) and testing data (https://github.com/ipavlopoulos/toxic_spans/blob/master/SemEval2021/data/tsd_test.csv)
We generate validation dataset on our own by splitting the training data. I am using K-fold validation for my main training as it gets better results than train_test_split() from sklearn library. However, a version of that is included in my program and commented out at the end(80% training, 20% validation), where splitting of train and validation is done using K fold.

b. Manipulate Data:
We do some manipulations on the data to make sure it runs on our models. That includes converting JSON objects to python dictionaries, converting columns to numpy where and when needed since models work best with numpy arrays for data. We also change all the texts to lower case, to NOT have “Yes” and “yes” as two different words.

c. Tokenize Data
We use pre-trained tokenizer available from tensorflow for Electra model. This will tokenize our texts and help the model train and make predictions on it.

d. Split Data
We split the training data into 2 categories: training, validation. This has been done in 2 ways for this project, either using train_test_split() or usong K-Fold validation. According to my experiments, both seems to give almost same results, with K-fold being slightly better for validation.

e. Create Inputs and Outputs
Inputs and outputs to model have to be in a certain format and structure for it to work with the models. Models usually work on arrays of numbers, which is why we need to convert the tokenized text and the spans into a format that can be fed to the models. Inputting texts

directly as strings does not work, and thus needs conversions. We feed the train, validation and testing through a function, which is then divided into x and y.

f. Build Model

Here we take the base model (Electra) and add our model on top of it. I have used LSTM and Bidirectional LSTM on top of Electra to compare if much bi-directionality matters when we already have strong transformers models as base. I have done so as Electra is usually efficient, and can hence be combined with the heavier Bidirectional LSTM model. The model needs an input list of varying length with one or several input Tensors IN THE ORDER given in the docstring:

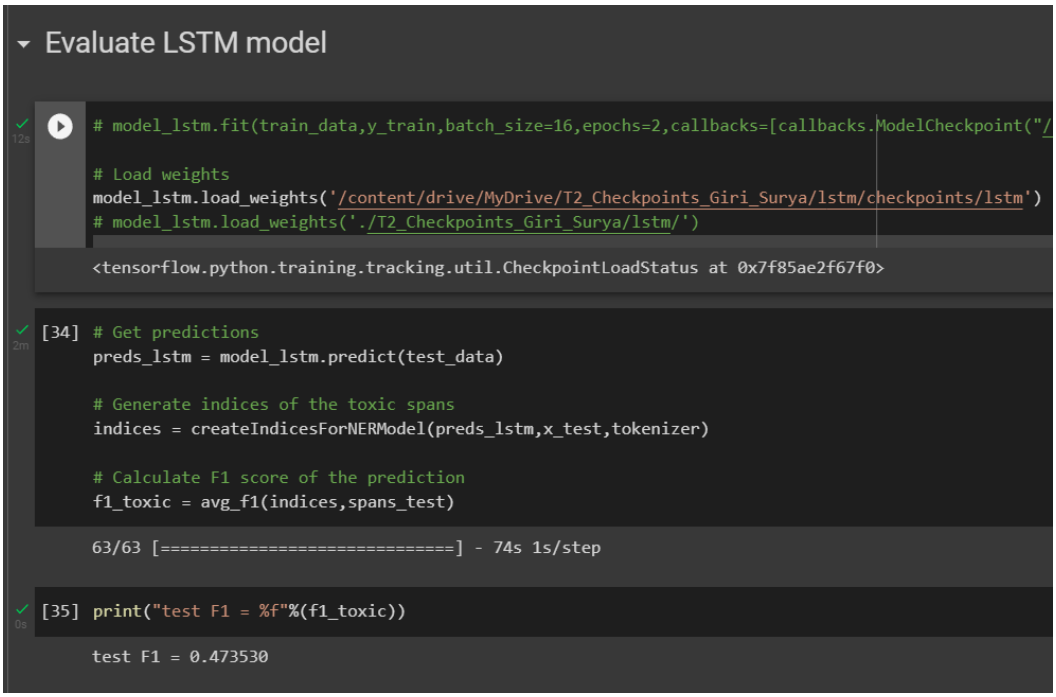
model([input_ids, attention_mask]) or model([input_ids, attention_mask, token_type_ids])

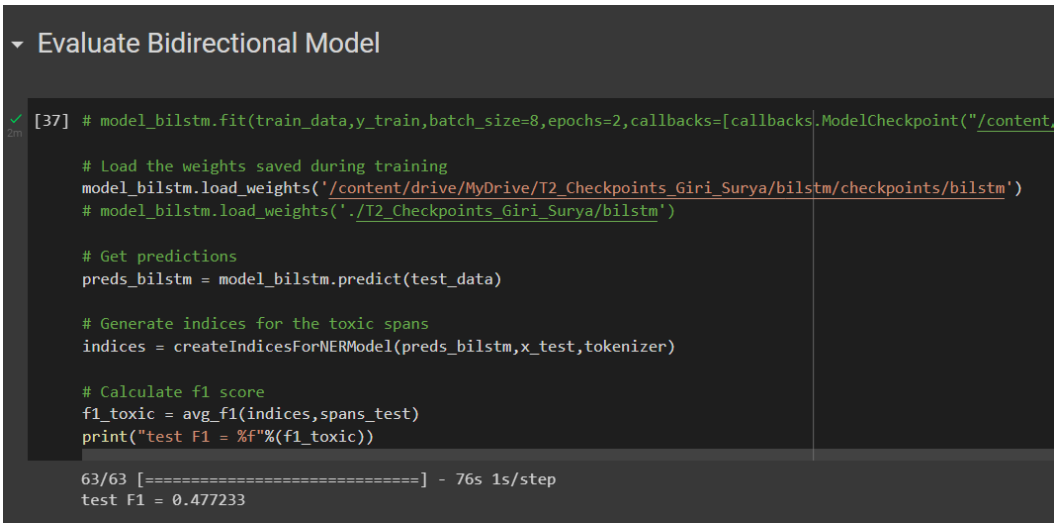
g. Fit and Train

For this we use run_train.py. The built model is now fit with training data and trained to generate weights. The training is checked using validation data taken from dataset. The trained weights are saved to Google Drive or can be stored locally(code included).

h. Make predictions

We use run_prediction.pynb for this. In this file, the model is built and compiled but not trained. We instead use weights saved during training, load it to the model and make predictions on the unseen test dataset. We will use these predictions to calculate out evaluation metrics, ie. F1 score.





4. Experiments

The experiments and trials that were conducted during this project included a variety of changes, from the dataset structuring to model comparison. The experiments and observations are as follows:

a. Data Manipulation: Not changing data columns to their proper type will result in the model not working or outputting empty or incorrect values.
b. Splitting Data: Data splitting can be done in various ways. I have experimented with K-fold and test_train_split(). K-fold usually gets slightly better validations results. Texts and spans from data can either pre-processed before or after split, but care should be taken when transforming the testing and validation set during pre-processing, as it can completely change results if testing data is seen before predicting, leading to incorrectly high accuracy.
c. Models: Compared base model(Electra), LSTM and Bi-LSTM. While Electra can get decent predictions, it can be improved and fine tuned using models on top of it. I experimented with both LSTM and Bi-LSTM. Bi-LSTM needs smaller batches to run and takes slightly longer as compared to LSTM but gives the same or slightly better results
d. Create Directories: Felt the need to create directories to ease the model checkpoint uploading process, so I wrote a function that creates directories as for storing model checkpoints for LSTM and BiLSTM models.
e. Model Checkpoints: While evaluating, I realized how tedious it is to link model checkpoints everytime and thus made functions to download and copy the checkpoint files directly from the drive. However, due to the size of the file, it could not be downloaded; and hence the function is commented out.

5. Conclusion

We got an f1 testing score of 47.353% for the LSTM model with Electra as base model, and an f1 testing score of 47.723 for our Bi-directional LSTM model. The Bi-LSTM model showed more promise during validation, with an increase of 2 to 3% in f1 validation score, but during testing, it practically did not make much difference. Hence we can conclude that Bi-directional LSTM is not much better than LSTM when it comes to time and f1 score. Thus, LSTM should be either fine tuned by performing hyperparameter tuning, or the base model needs to be changed for any significant improvements. However, if we are to implement heavier models on top of base models or if time efficiency is our concern, Electra would be the right choice as the base model.