NLP Deep
Project LAB 04

Alexandre Lemonnier - Sarah Gutierez
Victor Simonin - Eliott Bouhana

Promotion 2023

Contents

1 Introduction

We want to sell a moderation API tackling toxic content on Twitter. We find a collection of tweets labeled on HuggingFace. We want to train a model to predict the toxicity of a tweet. Two datasets seem close to our needs: ‘hate‘ and ‘offensive‘.

We will use the ‘hate‘ dataset due to its greatest toxicity. The moderation we need here is to detect some type of high toxicity firstly instead of offensive language. The fact that our model will be for a commercial application is also a reason to choose the ‘hate‘ dataset, hence the need to detect high toxicity.

2 Annotate data

Here are our first main guidelines to annotate the tweets :

- Read the tweet carefully.
- Identify the target of the tweet. Is the target an individual, a group, or an institution ?
- Identify the tone of the tweet. Is it angry, hateful, or otherwise inflammatory ?
- Identify any potentially offensive language in the tweet.
- Does it contain any personal information or insults ?
- The context in which the tweet was made. Was it in response to another user or event, and if so, how did that affect the tone of the tweet ?

When determining whether or not a tweet is toxic, there are a few key factors to consider. First, look at the overall tone of the tweet. Is it angry, hateful, or otherwise aggressive ? If so, it is likely to be classified as toxic. Next, consider the language used in the tweet. Are there any slurs or other offensive language ? If so, the tweet is likely to be classified as toxic. Finally, consider the context of the tweet. Is it in response to something ? If so, what is the response ? If it is a hateful or aggressive response, it is likely to be classified as toxic.

If the tweet is hateful, threatening, or otherwise offensive, it is likely to be considered toxic. If the tweet is in response to something, the context should be considered when determining whether the tweet is toxic. If the tweet is written in a joking or sarcastic manner, the tone should be considered when determining whether the tweet is toxic.

3 Theoretical Questions

3.1 What is the purpose of subword tokenization used by transformer models ?

The purpose of the subword tokenization used by transformer models is to to split words into subwords in order to better represent them. The words that are rarely used are changed to their subwords because the probability of their subwords showing is greater than the base word itself. So this can help the model to better learn the relationships between words, improve its ability to generalize to out-of-vocabulary words, and to allow it to better handle inflected forms of words.

3.1.1 What is the effect on the vocabulary size ?

It reduces the vocabulary size by replacing words with subwords. This can be beneficial if the dataset is large and the number of unique words is large. It can also be helpful if the data set contains a lot of rare words.

3.1.2 How does it impact out-of-vocabulary words (words which are not in the training data, but appear in the test data, or production environment) ?

Because the rarest words and composed words are broken into different subwords, you have less chances of facing out-of-vocabulary words. Which means that the model will be more effective and have less difficulties analyzing the input.

3.2 When building an encoder-decoder model using an RNN, what is the purpose of adding attention ?

3.2.1 What problem are we trying to solve ?

Attention is supposed to fix to the limitation of the Encoder-Decoder model encoding the input sequence to one fixed length vector from which to decode each output time step. In the RNN the limitation is due to the vanishing/exploding gradient problem. It remembers the parts which it just saw. So, the longer the input sentence, the worse the output.

3.2.2 How does attention solve the problem ?

Adding attention allows the model to focus on specific parts of the input when predicting a certain part of the output sequence, enabling easier learning and of higher quality. Attention itself adds 'weight', relative importance, to words in the context vector being created. This allows the RNN to ignore irrelevant information and focus on the most important information when making predictions.

3.3 In a transformer model what is the multihead attention used for ?

3.3.1 What are we trying to achieve with self-attention ?

Self-attention is trying to make sense of the input of the model. The main problem is that when processing a long sequence or multiple sequences the model looses information of relevance of each word. To memorize the whole sequence the self-attention is used.

3.3.2 Why do we use multiple head instead of one ?

The multihead attention is used to jointly attend to information from different parts of the input. By doing that, we create a more diverse representations which increases the performances of the transformer model.

3.4 In a transformer model, what is the purpose of positional embedding ?

In a transformer model, positional embedding is used to represent the relative or absolute position of tokens in the sequence. It is impactful because the transformer model is based on self-attention. It allows it to properly learn the relationships between the different positions of words in a sequence.

3.5 What would be the problem if we didn't use it ?

This is necessary because the transformer model does not have an inherent understanding of the order of the inputs and the meaning of it. Without positional embedding, the model would not be able to correctly learn the relationships between the words and would therefore be less accurate.

3.6 What are the are the purpose of benchmarks ?

The purpose of benchmarks is to compare the efficiency of the different models with the same dataset. It is a point of reference, a way of measuring progress in a domain.

3.6.1 And are they reliable ? Why ?

There is no overall reliable benchmark. Each one shows a different aspect of the performances of the model used. And they must be updated very frequently to be accurate.

3.7 What are the differences between BERT and GPT ?

There are several key differences between BERT and GPT:

- BERT is a bidirectional model, meaning it takes into account the context of a word both before and after it in a sentence. GPT is a unidirectional model, only taking into account context after the word.
- BERT is trained on a large amount of unlabeled data, as well as a small amount of labeled data. GPT is only trained on a small amount of labeled data.

- BERT can be used for a variety of tasks, including question answering and next sentence prediction. GPT is primarily used for next sentence prediction.
- BERT has been shown to outperform GPT on a number of natural language understanding tasks. However, GPT has been shown to outperform BERT on a number of natural language generation tasks.

3.7.1 What kind of transformer-based model are they ?

There are both transformer-based model that use self-attention to learn the contextual representation of a text.

3.7.2 How are they pre-trained ?

BERT is pre-trained using a combination of masked language modeling (MLM) and next sentence prediction (NSP) objectives. MLM involves randomly masking some of the tokens in the input, and then predicting the masked tokens. NSP is similar to MLM, it is a binary classification task that is used to predict whether two sentences are consecutive or not.

GPT is pre-trained using a self-supervised learning method called language modeling. In this method, the model is given a large amount of text data and is tasked with predicting the next word in the sequence. This pre-training task helps the model learn the general structure of language, which it can then use to better understand and generate text.

3.7.3 How are they fine-tuned ?

There are a few ways to finetune BERT and GPT. One way is to use a technique called "sequence-level Knowledge Distillation" (SLKD). With SLKD, the idea is to train a larger model (like BERT or GPT) on a task, and then use the output of that model as input to a smaller model. The smaller model is then trained to reproduce the output of the larger model. This technique has been shown to work well for a variety of tasks.

Another way to finetune BERT and GPT is to use the technique called "masked language modeling" (MLM). With MLM, the idea is to randomly mask a percentage

of the tokens in the input, and then have the model predict the masked tokens. This can be done with either the original BERT/GPT model or with a modified version of BERT/GPT that includes a "masked language modeling" head.

Finally, it is also possible to finetune BERT or GPT by training the model on a task-specific dataset. For example, if you want to use BERT for question answering, you can train the model on a dataset of question-answer pairs.

3.8 How are zero-shot and few-shots learning different from fine-tuning ?

In general, fine-tuning is a process of using a pre-trained model on a new dataset. The pre-trained model is first trained on a large dataset, and then the model weights are used to initialize a new model that is trained on the new dataset.

Zero-shot and few-shot learning are similar to fine-tuning in that they both use a pre-trained model to initialize a new model. However, in zero-shot and few-shot learning, the new model is not trained on the new dataset. Instead, the new model is only trained on a few examples from the new dataset. In zero-shot learning, the model is trained on a set of classes and then tested on a set of classes that is disjoint from the training set. In few-shots learning, the model is trained on a set of classes and then tested on a set of classes that is not disjoint from the training set.

3.8.1 How do fine-tuning, zero-shot, and few-shot learning affect the model's weights ?

Fine-tuning, zero-shot, and few-shot learning all affect the model's weights in different ways. Fine-tuning changes the weights of the model so that it better fits the data. Zero-shot learning changes the weights so that the model can better predict labels for data that it has never seen before. Few-shot learning changes the weights so that the model can better predict labels for data that it has only seen a few times (or partially labelled).

3.9 In a few paragraphs, explain how the triplet loss is used to train a bi-encoder model for semantic similarity ?

First, let's define what is a bi-encoder model. A bi-encoder model is a model that encodes two or more different types of information, in our case two or more languages simultaneously. This is helpful when the user wants to encode a text representation of a set of data in two or more languages.

The triplet loss is used to train a bi-encoder model for semantic similarity by optimizing the model to learn embeddings that preserve the similarity between two inputs. This loss function is used to encourage the model to learn a mapping from input data to a latent space in which semantically similar data points are close together and dissimilar data points are far apart. The triplet loss function minimizes the distance between an anchor input and a positive input, while maximizing the distance between the anchor input and a negative input.

This results in a model that is able to map semantically similar inputs to close vectors in the encoding space and semantically dissimilar inputs to far vectors in the encoding space.

3.10 What is the purpose of using an Approximate Nearest Neighbour method to speed up search ?

There are a number of reasons to use an Approximate Nearest Neighbour (ANN) method to speed up search.

First, ANN methods can be used to find the nearest neighbours of a query point more quickly than traditional methods.

Second, ANN methods can be used to find the nearest neighbours of a query point in higher-dimensional space more quickly than traditional methods.

Third, ANN methods can be used to find the nearest neighbours of a query point in a large dataset more quickly than traditional methods.

Finally, ANN methods can be used to find the nearest neighbours of a query point when the distance metric is not Euclidean (i.e., when the distance metric is something other than the straight-line distance between two points).

3.10.1 What does it really reduce ?

ANN reduces the time of search by preprocessing the data into an efficient index, reducing the dimensions of the datasets. This reduction makes it easier to find patterns.