INTERNSHIP REPORT

On the topic

“Jigsaw Multilingual Toxic Comment Classification”

DEPARTMENT OF MINING ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY (BHU) VARANASI

VARANASI, UTTAR PRADESH – 221005

submitted by

Kumar Saurav

16154012 (IDD Part V)

Department of Mining Engineering

CERTIFICATE OF INTERNSHIP

This is to certify that

Kumar Saurav

has completed "**Work From Home**" internship on Machine Learning with topic "**JIGSAW MULTILINGUAL TOXIC COMMENT CLASSIFICATION**" with acceptable good accuracy. We wish him/her good luck for future career.

Period: 1 month (April 1, 2020 to April 30, 2020)

IOT ACADEMY
Connecting THE Unconnected

Rabi Kant Das

RABI KANT DAS
Authorized Signatory

uct

Jigsaw Toxic Comment Classification Challenge

Introduction

The various forms of abusive and offensive content online pose risks to the users of social media platforms. One such example is cyberbullying which has been linked to depression and suicide risk among teenagers. As offensive language becomes pervasive in social media, scholars and companies have been working on developing systems capable of identifying offensive posts, which can be set aside for human moderation or permanently deleted. The use of robust NLP methods in these systems is paramount to cope with the many ways such content can be presented. While direct abuse and insults containing profanity are fairly easy to be identified, recognizing indirect insults for example, which often include metaphors and sarcasm, are a challenge to human annotators and, as a consequence, to most state-of-the-art systems. In light of these important challenges, a recent growing interest of the NLP community in detecting abusive and offensive content online has been observed. This is evidenced by previous work focusing on the identification of abusive content, aggression, cyberbullying, hate speech, and offensive language.

Methods and Data

The dataset contains the text of comments from Wikipedia's talk page edits with a toxic column that is the target to be trained on. We use only minimal pre-processing methods to make the system portable to all languages in the dataset. For classification, we used and compared different neural network architectures suited to this task.

Dataset

You are provided with a large number of Wikipedia comments which have been labelled by human ratters for toxic behaviour. The types of toxicity are:

`toxic` `severe_toxic` `obscene` `threat` `insult` `identity_hate` You must create a model which predicts a probability of each type of toxicity for each comment.

File descriptions

train.csv - the training set, contains comments with their binary labels
test.csv- the test set, you must predict the toxicity probabilities for these comments. To deter hand labelling, the test set contains some comments which are not included

in scoring.
sample_submission.csv - a sample submission file in the correct format
test_labels.csv - labels for the test data; value of -1 indicates it was not used for scoring; (Note: file added after competition close!)

We analyse a dataset published by Google Jigsaw in December 2017 over the course of the 'Toxic Comment Classification Challenge' on Kaggle. It includes 223,549 annotated user comments collected from Wikipedia talk pages and is the largest publicly available for the task. These comments were annotated by human raters with the six labels 'toxic', 'severe toxic, 'insult', 'threat', 'obscene' and 'identity hate'. Comments can be associated with multiple classes at once, which frames the task as a multi-label classification problem. Jigsaw has not published official definitions for the six classes. But they do state that they defined a toxic comment as "a rude, disrespectful, or unreasonable comment that is likely to make you leave a discussion".

The dataset features an unbalanced class distribution, shown in Table 1. 201,081 samples fall under the majority 'clear' class matching none of the six categories, whereas 22,468 samples belong to at least one of the other classes. While the 'toxic' class includes 9.6% of the samples, only 0.3% are labelled as 'threat', marking the smallest class. Comments were collected from the English Wikipedia and are mostly written in English with some outliers, e.g., in Arabic, Chinese or German language. The domain covered is not strictly locatable, due to various article topics being discussed.

Contents

Filename	Description
EDA.ipynb	Exploratory Data Analysis of the dataset
preprocessing.ipynb	Data Pre-processing steps and garbage removal
model.ipynb	Comparative study of multiple ML algorithms on the training set
testing.ipynb	Model training and result generation and accuracy testing

Dependencies

We assume you have python 3.5+ installed. Follow the instructions below to set up your conda environment (we have provided the .yaml file) and finally, download all of

nltk's packages by using the following steps on a python command line. 2. `import nltk` 3. `nltk.download()`
Before you run our LSTM implementation, you also need to download the GloVe embeddings, which are about 5.4gb unzipped. Here is the link. Download the file for the Common Crawl 840b embeddings, 300 dimensions.

Pipeline

Our entire process consists of three steps. At the first step, we preprocess the data, by exploring different techniques of removing stopwords, lemmatization, etc. The best performing one goes to our next step, which is word embeddings. At this step, we explore different techniques such as count-based vectorizers, tf-idf vectorizers and word2vec. Again, the best performing technique is used in many different classifiers.

Comparison of human performance

We also compared our algorithms performance with an average human's performance over 1000 test samples. We found that most of our algorithms with tuned parameters outperform a human at the same task.

Text Pre-processing

As mentioned previously, the data pre-processing for this task was kept fairly minimal to make it portable for all the languages. More specifically, we perform only three specialised tasks for this data, followed by tokenisation. The tasks include removing usernames, removing URLs, and converting all tokens to lower case.

First, we completely remove all usernames from the texts, without inserting a placeholder. This is carried out by removing all strings beginning with the @ symbol, as this is how usernames are denoted on Twitter. The reasoning behind this step is mainly to remove noisy text, as it is highly unlikely that there would be any embeddings for the usernames. In addition to that, we believe that these usernames don’t add much semantic meaning. Moreover, if, for instance, a majority of offensive tweets were written by one user, this could lead to bias in the system against one user.

After that, we remove all the URLs from the texts. All the tweets contain an URL which also refers to the URL of the particular tweet. All these URLs start with https://t.co/ followed by an unique ID. Therefore, we used a regular expression to remove all strings beginning with the https://t.co/. Similar to the usernames, URLs too do not add any semantic meaning and can be considered as noisy.

Final prepossessing step is only applied to the architectures that used character embeddings. The fast Text pretrained character embedding models that we used only contain lower-cased letters. Therefore, we convert the text to lower case letters. However, the BERT models we used are cased. This pre-processing step was not used to the BERT based architecture.

Recurrent Neural Networks

Recurrent Neural Networks (RNNs) interpret a document as a sequence of words or character ngrams. We use four different RNN approaches: An LSTM (Long-Short-Term-Memory Network), a bidirectional LSTM, a bidirectional GRU (Gated Recurrent Unit) architecture and a bidirectional GRU with an additional attention layer.

LSTM

Our LSTM model takes a sequence of words as input. An embedding layer transforms one-hot-encoded words to dense vector representations and a spatial dropout, which randomly masks 10% of the input words, makes the network more robust. To process the sequence of word embeddings, we use an LSTM layer with 128 units, followed by a dropout of 10%. Finally, a dense layer with a sigmoid activation makes the prediction for the multi-label classification and a dense layer with softmax activation makes the prediction for the multi-class classification.

Bidirectional LSTM and GRU

Bidirectional RNNs can compensate certain errors on long range dependencies. In contrast to the standard LSTM model, the bidirectional LSTM model uses two LSTM layers that process the input sequence in opposite directions. Thereby, the input sequence is processed with correct and reverse order of words. The outputs of these two layers are averaged. Similarly, we use a bidirectional GRU model, which consists of two stacked GRU layers. We use layers with 64 units. All other parts of the neural network are inherited from our standard LSTM model. As a result, this network can recognize signals on longer sentences where neurons representing words further apart from each other in the LSTM sequence will 'fire' more likely together.

Ensemble Learning

Each classification method varies in its predictive power and may conduct specific errors. For example, GRUs or LSTMs may miss long range dependencies for very long sentences with 50 or more words but are powerful in capturing phrases and complex context information. Bi-LSTMs and attention-based networks can compensate these errors to a certain extent. Sub word Embeddings can model even misspelled or obfuscated words.

Therefore, we propose an ensemble deciding which of the single classifiers is most powerful on a specific kind of comment. The ensemble observes features in comments, weights and learns an optimal classifier selection for a given feature combination. For achieving this functionality, we observe the set of out-of-fold predictions from the various approaches and train an ensemble with gradient boosting decision trees. We perform 5- fold cross-validation and average final predictions on the test set across the five trained models.

Evaluation

Our hypothesis is that the ensemble learns to choose an optimal combination of classifiers based on a set of comment features. Because the classifiers have different strengths and weaknesses, we expect the ensemble to outperform each individual classifier. Based on results from previous experiments mentioned in Section 2 we expect that the state-of-the-art models have a comparable performance and none outperforms the others significantly. This is important because otherwise the ensemble learner constantly prioritizes the outperforming classifier. We expect our ensemble to perform well on both online comments and Tweets despite their differing language characteristics such as comment length and use of slang words.

Experimental Results

our ensemble outperforms the strongest individual method on the Wikipedia dataset by approximately one percent F1-measure. We see that the difference in F1-measure between the best individual classifier and the ensemble is higher on the Wikipedia dataset as on the Twitter dataset. This finding is accompanied by the results in Table 4 which show that most classifier combinations present a high correlation on the Twitter dataset and are therefore less effective on the ensemble. An explanation for this effect is that the text sequences within the Twitter set show less variance than the ones in the Wikipedia dataset. This can be reasoned from 1) their sampling strategy based on a list of terms, 2) the smaller size of the dataset and 3) less disparity within the three defined classes than in the six from the Wikipedia dataset. With less variant data one selected classifier for a type of text can be sufficient. As the results in Table 4 show, ensembling is especially effective on the sparse classes ”threat” (Wikipedia) and “hate” (Twitter). The predictions for these two classes have the weakest correlation. This can be exploited when dealing with strongly imbalanced datasets, as often the case in toxic comment classification and related tasks. Table 4 gives us indicators for useful combinations of classifiers. Combining our shallow learner approach with Neural Networks is highly effective. Contrary to that we see that the different word embeddings used do not lead to strongly differing predictions.

How to use our code

Since we are using python, the best way to test out code will be using a virtual environment. We recommend installing anaconda first.

Once you are done with that, open a command line console and type the following. (Make sure Anaconda is in your path!) `conda env create -f toxic-comments.yaml` This will create a new environment, named "toxic-comments" which you can then activate before running our code. To activate an environment, go to a command line and type: `source activate toxic-comments` on Mac/Linux `activate toxic-comments` on Windows
Now you can run any of our programs!

1. Conversion from kaggle data to a binary class problem is done by a file inside the data folder -> Raw_data -> data_from_kaggle.ipynb

2. Preprocessing the data is done by the notebook in preprocessed data folder.

3. The variety of feature extraction techniques are in the feature extraction folder.

4. The various techniques that we tried are in the model folder.

5. Our human benchmark is in the human performance folder.

Acknowledgements: Our LSTM implementation has been adapted from this kaggle kernel:https://www.kaggle.com/qqgeogor/keras-lstm-attention-glove840b-lb-0-043/comments. All other work is largely our own. The notebook has a few mistakes but talks a lot about LSTMs with attention, therefore please use our version which is fixed.

In [ ]:
```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load in

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the "../input/" directory.
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directo

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# Any results you write to the current directory are saved as output.
```

In [ ]:
```
DATA_PATH = "/kaggle/input/jigsaw-multilingual-toxic-comment-classification/"
os.listdir(DATA_PATH)
TEST_PATH = DATA_PATH + "test.csv"
VAL_PATH = DATA_PATH + "validation.csv"
TRAIN_PATH1 = DATA_PATH + "jigsaw-toxic-comment-train.csv"
TRAIN_PATH2=DATA_PATH+"jigsaw-unintended-bias-train.csv"
```

In [ ]:
```
valid = pd.read_csv(VAL_PATH)
test = pd.read_csv(TEST_PATH)
train1 = pd.read_csv(TRAIN_PATH1)
train2=pd.read_csv(TRAIN_PATH2)
```

In [ ]:
```python
import os
import tensorflow as tf
from tensorflow.keras.layers import Dense, Input
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.models import Model
from tensorflow.keras.callbacks import ModelCheckpoint
from kaggle_datasets import KaggleDatasets
import transformers
from tqdm import tqdm
from tokenizers import BertWordPieceTokenizer

from keras.models import Sequential
from keras.layers.recurrent import LSTM, GRU,SimpleRNN
from keras.layers.core import Dense, Activation, Dropout
from keras.layers.embeddings import Embedding
from keras.layers.normalization import BatchNormalization
from keras.utils import np_utils
from keras.preprocessing import sequence, text
from keras.layers.embeddings import Embedding
```

In [ ]:
```python
try:
    # TPU detection. No parameters necessary if TPU_NAME environment variable is
    # set: this is always the case on Kaggle.
    tpu = tf.distribute.cluster_resolver.TPUClusterResolver()
    print('Running on TPU ', tpu.master())
except ValueError:
    tpu = None

if tpu:
    tf.config.experimental_connect_to_cluster(tpu)
    tf.tpu.experimental.initialize_tpu_system(tpu)
    strategy = tf.distribute.experimental.TPUStrategy(tpu)
else:
    # Default distribution strategy in Tensorflow. Works on CPU and single GPU.
    strategy = tf.distribute.get_strategy()

print("REPLICAS: ", strategy.num_replicas_in_sync)
```

In [ ]:
```
def fast_encode(texts, tokenizer, chunk_size=256, maxlen=512):
    """
    Encoder for encoding the text into sequence of integers for BERT Input
    """
    tokenizer.enable_truncation(max_length=maxlen)
    tokenizer.enable_padding(max_length=maxlen)
    all_ids = []

    for i in tqdm(range(0, len(texts), chunk_size)):
        text_chunk = texts[i:i+chunk_size].tolist()
        encs = tokenizer.encode_batch(text_chunk)
        all_ids.extend([enc.ids for enc in encs])

    return np.array(all_ids)
```

In [ ]:
```
AUTO = tf.data.experimental.AUTOTUNE


# Configuration
EPOCHS = 3
BATCH_SIZE = 16 * strategy.num_replicas_in_sync
MAX_LEN = 192
```

In [ ]:
```
tokenizer = transformers.DistilBertTokenizer.from_pretrained('distilbert-base-multilingual-cased')
# Save the loaded tokenizer locally
tokenizer.save_pretrained('.')
# Reload it with the huggingface tokenizers library
fast_tokenizer = BertWordPieceTokenizer('vocab.txt', lowercase=False)
fast_tokenizer
```

In [ ]:
```
x_train = fast_encode(train1.comment_text.astype(str), fast_tokenizer, maxlen=MAX_LEN)
x_valid = fast_encode(valid.comment_text.astype(str), fast_tokenizer, maxlen=MAX_LEN)
x_test = fast_encode(test.content.astype(str), fast_tokenizer, maxlen=MAX_LEN)

y_train = train1.toxic.values
y_valid = valid.toxic.values
```

In [ ]:
```
model = Sequential()
model.add(Embedding(120000, 300, input_length=192))
model.add(LSTM(200, dropout=0.3, recurrent_dropout=0.3))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam',metrics=['accuracy'])
model.summary()
```

In [ ]:
```
model.fit(x_train, y_train,validation_data=(x_valid,y_valid), nb_epoch=5, batch_size=64)
```

In [ ]:
```
sub = pd.read_csv('/kaggle/input/jigsaw-multilingual-toxic-comment-classification/sample_submission.csv')
```

In [ ]:
```
y_pred=model.predict(x_test)
```

In [15]:
```
y_pred
```

Out[15]:
```
array([[1.3760336e-01],
       [2.7810903e-03],
       [6.7464681e-04],
       ...,
       [8.1742340e-01],
       [3.5086882e-03],
       [3.5104018e-02]], dtype=float32)
```

In [14]:
```
sub['toxic']=y_pred
```

In [16]:
```
sub.to_csv('submission.csv', index=False)
```

REFRENCES

1) Pinkesh Badjatiya, Shashank Gupta, Manish Gupta, and Vasudeva Varma. 2017. Deep learning for hate speech detection in tweets. In WWW.
2) Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching word vectors with subword information.
3) Pete Burnap and Matthew L. Williams. 2015. Cyber hate speech on twitter : An application of machine classification and statistical modeling for policy and decision making.
4) Pete Burnap and Matthew L. Williams. 2016. Us and them: identifying cyber hate on twitter across multiple protected characteristics. EPJ Data Science.
5) Ying Chen, Yilu Zhou, Sencun Zhu, and Heng Xu. 2012. Detecting offensive language in social media to protect adolescent online safety. SOCIALCOMPASSAT. Maral Dadvar, Dolf Trieschnigg, Roeland Ordelman, and Franciska de Jong. 2013. Improving cyberbullying detection with user context. In ECIR.
6) Thomas Davidson, Dana Warmsley, Michael W. Macy, and Ingmar Weber. 2017. Automated hate speech detection and the problem of offensive language. In ICWSM.
7) Karthik Dinakar, Birago Jones, Catherine Havasi,Henry Lieberman, and Rosalind W. Picard. 2012. Common sense reasoning for detection, prevention, and mitigation of cyberbullying.
8) Bjorn Gamb ¨ ack and Utpal Kumar Sikdar. 2017. Using convolutional neural networks to classify hatespeech.
9) Basile, V., Bosco, C., Fersini, E., Nozza, D., Patti, V., Pardo, F.M.R., Rosso, P., Sanguinetti, M.: Semeval-2019 task 5: Multilingual detection of hate speech against immigrants and women in twitter. In: Proceedings of the 13th International Workshop on Semantic Evaluation.
10) Bauman, S., Toomey, R.B., Walker, J.L.: Associations among bullying, cyberbullying, and suicide in high school students.
11) Chelmis, C., Zois, D.S., Yao, M.: Mining patterns of cyberbullying on twitter. In: 2017 IEEE International Conference on Data Mining Workshops (ICDMW).
12) Chung, J., C¸ aglar G¨ul¸cehre, Cho, K., Bengio, Y.: Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling.