Natural Language Processing with Disaster Tweets

By Paul Kelly, Daniel Felberg, and Cora Martin

Disaster Tweets

Non-disaster Tweet:

```
train_df[train_df["target"] == 0]["text"].values[1]
```

```
‘Today I took a walk’
```

Disaster Tweet:

```
train_df[train_df["target"] == 1]["text"].values[1]
```

```
‘Code red air quality alert issued in NYC'
```

Natural Language Processing

DistilBERT model

```
tokenizer = DistilBertTokenizer.from_pretrained('distilbert-base-uncased', do_lower_case=True)
```

```
model = TFDistilBertForSequenceClassification.from_pretrained('distilbert-base-uncased’)
```

`distilbert-base-uncased` is a pre-trained tokenizer trained on lowercase text.

Data

df_train:

	id	keyword	location	text	target
0	1	NaN	NaN	Our Deeds are the Reason of this #earthquake M...	1
1	4	NaN	NaN	Forest fire near La Ronge Sask. Canada	1
2	5	NaN	NaN	All residents asked to 'shelter in place' are ...	1
3	6	NaN	NaN	13,000 people receive #wildfires evacuation or...	1
4	7	NaN	NaN	Just got sent this photo from Ruby #Alaska as ...	1

	# rows	# columns
0	7613	5

df_test:

	id	keyword	location	text
0	0	NaN	NaN	Just happened a terrible car crash
1	2	NaN	NaN	Heard about #earthquake is different cities, s...
2	3	NaN	NaN	there is a forest fire at spot pond, geese are...
3	9	NaN	NaN	Apocalypse lighting. #Spokane #wildfires
4	11	NaN	NaN	Typhoon Soudelor kills 28 in China and Taiwan

	# rows	# columns
0	3263	4

Goal

Target: whether a tweet is a disaster tweet (target = 1) or a non-disaster tweet (target = 0).

Submission:

	id	target
0	0	0
1	2	1
2	3	1
3	9	0
4	11	1

Libraries and Data Pre-processing

```
!pip install transformers
!pip install datasets
!pip install keras-tuner
```

```
#drop columns
df_train.drop(['id','keyword','location'],axis=1,inplace=True)
df_test.drop(['id','keyword','location'],axis=1,inplace=True)
```

```
#importing libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from transformers import DistilBertTokenizer, TFDistilBertForSequenceClassification
import tensorflow as tf
import keras_tuner as kt
from google.colab import drive
import sys
```

	text	target
0	Our Deeds are the Reason of this #earthquake M...	1
1	Forest fire near La Ronge Sask. Canada	1
2	All residents asked to 'shelter in place' are ...	1
3	13,000 people receive #wildfires evacuation or...	1
4	Just got sent this photo from Ruby #Alaska as ...	1

Tokenizer and pre-trained model

Here, we make sure that the required packages for our experiment (transformers, datasets, and keras-tuner) are installed.

```
#tokenizer: https://www.techtarget.com/searchenterpriseai/definition/BERT-language-model
#switched from BertTokenizer and TFBertForSequenceClassification to DistilBert
tokenizer = DistilBertTokenizer.from_pretrained('distilbert-base-uncased', do_lower_case=True)
model = TFDistilBertForSequenceClassification.from_pretrained('distilbert-base-uncased')
```

Split Features and Target

We now split our features and target in train (and identify our target in test), then tokenize them (padding to maximum ensures that each feature is the same length).

```
# Batch tokenize our tweet field
X_train = tokenizer.batch_encode_plus(df_train.text, pad_to_max_length=True, return_tensors="tf")
X_test = tokenizer.batch_encode_plus(df_test.text, pad_to_max_length=True, return_tensors="tf")

# Get our target
y_train = df_train['target'].to_numpy()
```

Utilizing an Optimizer

Before we train our model, we need some method of selecting its parameters. Instead of guessing, we build a class that harnesses Keras Tuner to help us. Note that we chose the **AdamW optimizer** because it performed better than Adam in some casual pre-testing.

```
#Technique sourced from tensorflow documentation: https://www.tensorflow.org/tutorials/keras/keras_tuner
#Create class to allow Keras Tuner to build the best model from our pre-defined options
def model_builder(hp):
    model = TFDistilBertForSequenceClassification.from_pretrained('distilbert-base-uncased')
    #optimal learning rates sourced from "BERT" paper: https://arxiv.org/abs/1810.04805
    hp_learning_rate = hp.Choice('learning_rate', values=[2e-5, 3e-5, 4e-5, 5e-5])
    #AdamW optimizer instead of Adam: https://towardsdatascience.com/why-adamw-matters-736223f31b5d
    model.compile(optimizer=tf.keras.optimizers.AdamW(learning_rate=hp_learning_rate),
                  loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                  metrics=tf.keras.metrics.SparseCategoricalAccuracy('accuracy'))

    return model
```

Keras Hyperband Tuner and Early Stopping

Then, we set the objective for our training, best validation accuracy, and the folder to store our tuning data. Then we set a callback to ensure that if our model doesn't improve after 5 epochs, tuning stops. Note the 80/20 train-validation split.

```
#Invoke Keras Tuner's Hyperband tuner.
tuner = kt.Hyperband(model_builder,
                     objective='val_accuracy',
                     max_epochs=10,
                     factor=3,
                     directory='ml_1',
                     project_name='final_project')
```

```
#Add early stop callback for the tuner search.
stop_early = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=5)
```

Learning Rate Tuner

Using a learning rate tuner, we discover that the optimal learning rate for the optimizer is 2e-05.

```
#Begin tuning with learning rate
tuner.search(x=X_train['input_ids'], y=y_train, epochs=10, validation_split=0.2, callbacks=[stop_early])

best_hps=tuner.get_best_hyperparameters(num_trials=1)[0]

print(f"""
The hyperparameter search is complete. The optimal learning rate for the optimizer is {best_hps.get('learning_
""")
```

```
Trial 4 Complete [00h 26m 12s]
val_accuracy: 0.8319107294082642

Best val_accuracy So Far: 0.8351936936378479
Total elapsed time: 01h 45m 04s
INFO:tensorflow:Oracle triggered exit

The hyperparameter search is complete. The optimal learning rate for the optimizer is 2e-05.
```

Learning Rate Tuner

We then carry over the selected learning rate and tune again for best epoch. We select an epoch level of 10. Again, note the 80/20 train-validation split. We discover that the best epoch is 3.

```
#Continue tuning for best epoch
model = tuner.hypermodel.build(best_hps)
history = model.fit(x=X_train['input_ids'], y=y_train, epochs=10, validation_split=0.2)

val_acc_per_epoch = history.history['val_accuracy']
best_epoch = val_acc_per_epoch.index(max(val_acc_per_epoch)) + 1
print('Best epoch: %d' % (best_epoch,))
```

```
Best epoch: 3
```

Model

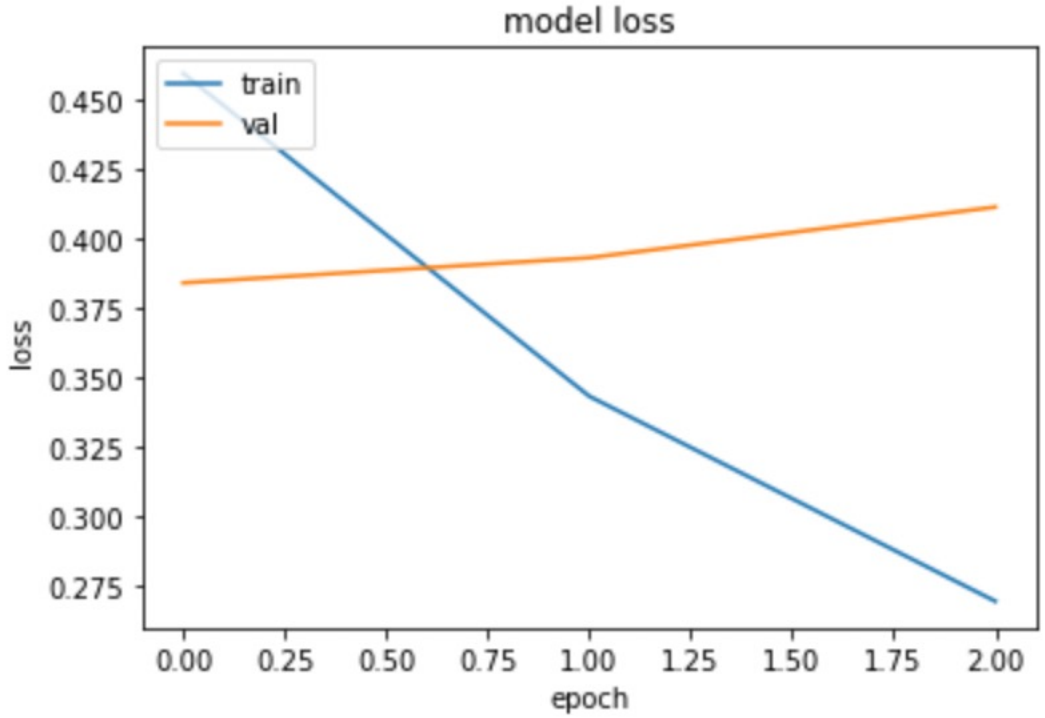
We then feed our best learning rate and epoch into one final training session. Again,note the 80/20 train-validation split.

```
#Retrain with best hyperparameters
hypermodel = tuner.hypermodel.build(best_hps)
#Once again implementing train/val split
history = hypermodel.fit(x=X_train['input_ids'], y=y_train, epochs=best_epoch, validation_split=0.2)
```

```
Epoch 1/3
191/191 [==============================] - 815s 4s/step - loss: 0.4592 - accuracy: 0.7913 - val_loss: 0.3840 - val_accuracy: 0.8299
Epoch 2/3
191/191 [==============================] - 786s 4s/step - loss: 0.3433 - accuracy: 0.8626 - val_loss: 0.3930 - val_accuracy: 0.8293
Epoch 3/3
191/191 [==============================] - 788s 4s/step - loss: 0.2697 - accuracy: 0.8985 - val_loss: 0.4112 - val_accuracy: 0.8359
```

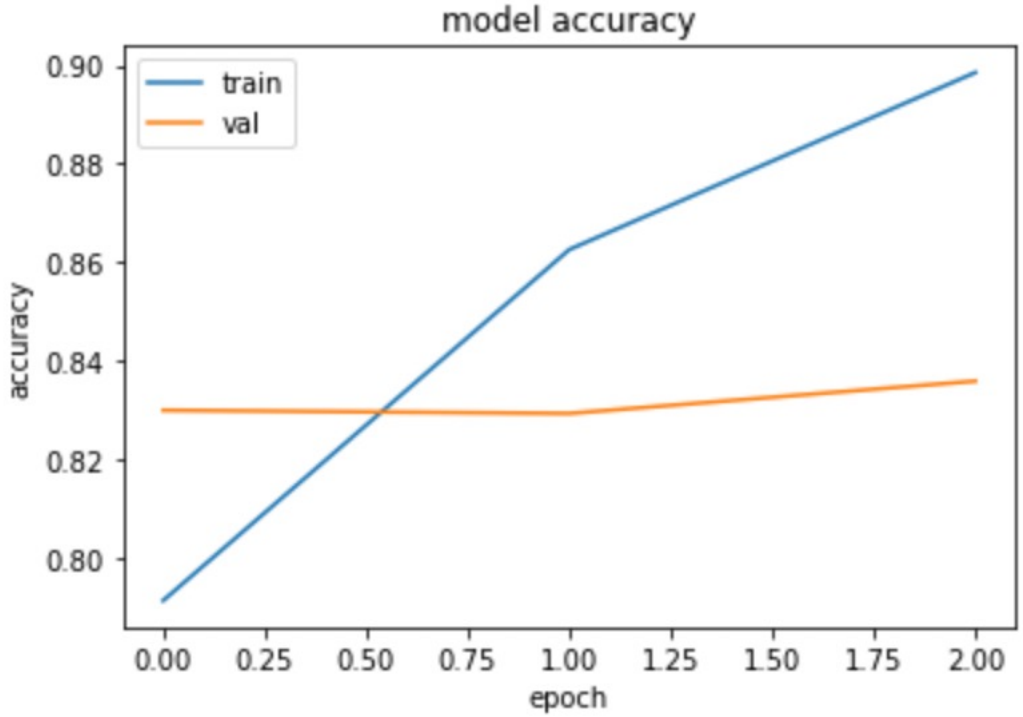
Model Loss

Validation loss increases from epoch 1 to 3, but not to a concerning degree. It does not appear that model is overfitting.



Model Accuracy

Validation accuracy is increasing with epoch. It does not appear that model is overfitting, and we know from our hyperparameter training that it would begin to fall with additional epochs.



Predictions

```
#predict
predictions = model.predict(X_test['input_ids'])
predictions_label = [np.argmax(x) for x in predictions[0]]
#print results
results = pd.DataFrame({'id': df_raw_test['id'], 'target': predictions_label})
results['target'] = results['target'].astype('int')
results.to_csv('predictions.csv', index=False)
#profit!
```

Executive Summary

We preprocessed our data, tuned then trained our model for a val_accuracy of 0.8359 and val_loss of 0.4112. Then, we ran the model on our test data to make predictions. Note that if re-running this notebook, results may differ slighly due to the train/val split method (i.e. no random seed - keras does not utilize it).

Conclusion

There are several takeways from this experiment. Firstly, using DistilBert instead of Bert for binary classification saved us a lot of time (indeed, Bert often crashed our computers). Secondly, we found the assertion in Bert's documentation (and by extension DistilBert's) that their pretrained model usually performs best in epochs 1 through 3 to be true (in our case, epoch 3 was best). Thirdly, after spending a lot of time trying different hyperparameter combinations manually through trial and error, Keras Tuner proved invaluable.

Future Research

Future work might include performing additional text preprocessing to see how model performance is affected.