



DTSA 5511 - INTRODUCTION TO DEEP LEARNING

UNIVERSITY OF COLORADO, BOULDER

MASTER OF SCIENCE IN DATA SCIENCE

Natural Language Processing with Disaster Tweets

1 Introduction

In this competition, we are challenged to build a machine learning model that predicts which Tweets are about real disasters and which one's aren't. We'll have access to a dataset of 10,000 tweets that were hand classified.

We'll be starting with a Keras starter code[2] and try to improve upon it by adding some feature engineering and preprocessing and tuning the hyper parameters. We need to install and import `keras_core` and `keras_nlp`.

2 Exploratory Data Analysis(EDA)

2.1 Data Loading and Libraries Import

```
!pip install keras-core --upgrade
!pip install -q keras-nlp --upgrade

import os
os.environ['KERAS_BACKEND'] = 'tensorflow'

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.
    read_csv)
import tensorflow as tf
import keras_core as keras
import keras_nlp
from sklearn.metrics import ConfusionMatrixDisplay,
    confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
from keras.optimizers import Adam

df_train = pd.read_csv("/kaggle/input/nlp-getting-started/train
    .csv")
df_test = pd.read_csv("/kaggle/input/nlp-getting-started/test.
    csv")
```

2.2 EDA and Preprocessing

We can see that there are not any missing values for id,text and target column. We fill in the missing location and keywords with "unk" for unknown and will drop any duplicates. We also create a new column as the length of the tweet. We also define some hyper parameters that we will be playing with later to see the results. We also split the training set to training and validation set :

```
def print_missing_values(df, name):
    print(f'Missing values in { name}')
    for col in df.columns:
        col_missing = df[col].isna().sum()
        print(f'{col}: {100*col_missing/len(df):.2f}%',)
    print()
print_missing_values(df_train, 'Training dataset')
print_missing_values(df_test, 'Test dataset')

df_train = df_train.fillna('unk')
df_test = df_test.fillna('unk')

cols = ['text', 'target', 'keyword', 'location']
train_df = train_df.drop_duplicates(cols).reset_index(drop =
    True)

df_train["length"] = df_train["text"].apply(lambda x : len(x))
df_test["length"] = df_test["text"].apply(lambda x : len(x))

BATCH_SIZE = 32
NUM_TRAINING_EXAMPLES = df_train.shape[0]
TRAIN_SPLIT = 0.8
VAL_SPLIT = 0.2
STEPS_PER_EPOCH = int(NUM_TRAINING_EXAMPLES)*TRAIN_SPLIT //
    BATCH_SIZE

EPOCHS = 5
AUTO = tf.data.experimental.AUTOTUNE
from sklearn.model_selection import train_test_split

X = df_train["text"]
y = df_train["target"]

X_train, X_val, y_train, y_val = train_test_split(X, y,
    test_size=VAL_SPLIT, random_state=42)
```

```
X_test = df_test["text"]
```

3 Model Selection and Training

We are using the DistilBERT model from `keras_nlp` which is quite smaller than the original BERT and much faster. One may wish to experiment with different models. For the sake of succinctness, we continue with our DistilBERT model and try different hyperparameters to see whether we can make any improvements.

```
# Load a DistilBERT model.
preset= "distil_bert_base_en_uncased"

# Use a shorter sequence length.
preprocessor = keras_nlp.models.DistilBertPreprocessor.
    from_preset(preset,
                sequence_length=160,
                name="preprocessor_4_tweets"
    )

# Pretrained classifier.
classifier = keras_nlp.models.DistilBertClassifier.from_preset(
    preset,
    preprocessor = preprocessor,
    num_classes=2)

classifier.summary()

# Compile
classifier.compile(
    loss=keras.losses.SparseCategoricalCrossentropy(from_logits
        =True),
    optimizer=Adam(1e-5),
    metrics= ["accuracy"]
)

# Fit
history = classifier.fit(x=X_train,
                        y=y_train,
                        batch_size=BATCH_SIZE,
                        epochs=EPOCHS,
                        validation_data=(X_val, y_val))
```

4 Confusion Matrix, Predicting the Labels for Test Tweets and Submission File

```
def displayConfusionMatrix(y_true, y_pred, dataset):
    disp = ConfusionMatrixDisplay.from_predictions(
        y_true,
        np.argmax(y_pred, axis=1),
        display_labels=["Not Disaster", "Disaster"],
        cmap=plt.cm.Blues
    )

    tn, fp, fn, tp = confusion_matrix(y_true, np.argmax(y_pred,
        axis=1)).ravel()
    f1_score = tp / (tp+((fn+fp)/2))

    disp.ax_.set_title("Confusion Matrix on " + dataset + "
        Dataset -- F1 Score: " + str(f1_score.round(2)))

y_pred_train = classifier.predict(X_train)

displayConfusionMatrix(y_train, y_pred_train, "Training")

y_pred_val = classifier.predict(X_val)

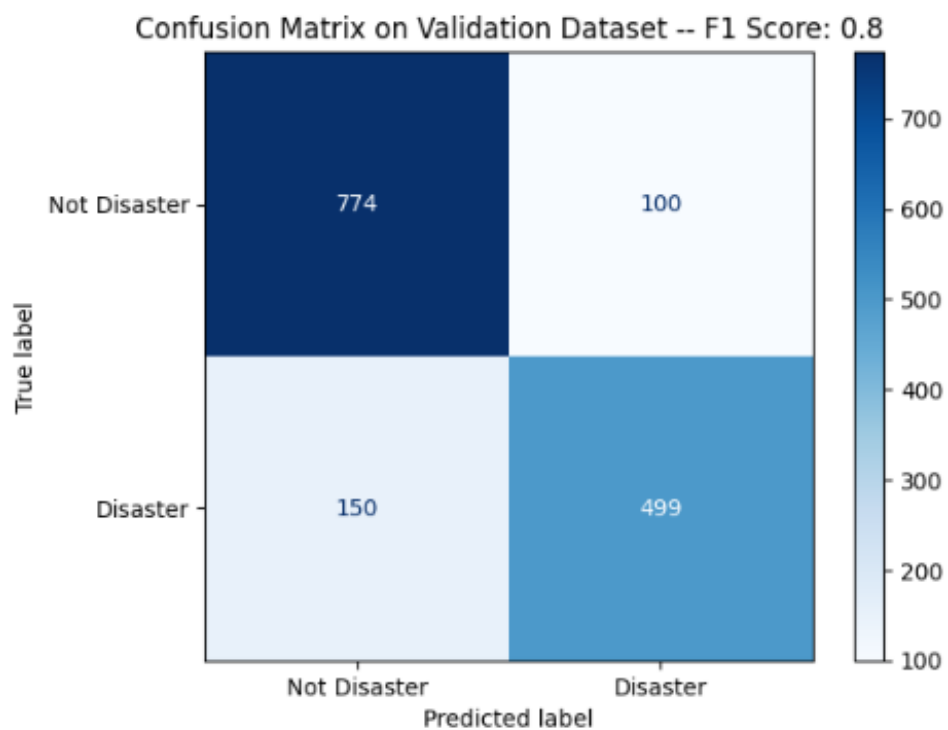
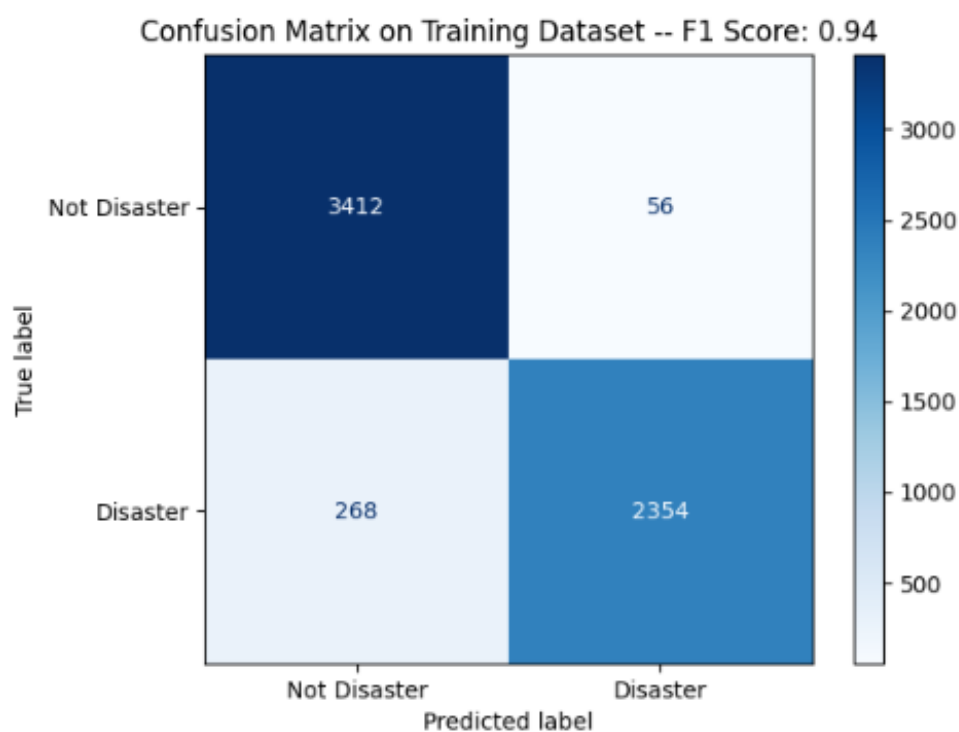
displayConfusionMatrix(y_val, y_pred_val, "Validation")

sample_submission = pd.read_csv("/kaggle/input/nlp-getting-
    started/sample_submission.csv")

sample_submission["target"] = np.argmax(classifier.predict(
    X_test), axis=1)









sample_submission.to_csv("b32_e5_lr1e5.csv", index=False)
```

4 CONFUSION MATRIX, PREDICTING THE LABELS FOR TEST TWEETS AND SUBMISSION FILE



5 Results

Below you can see the public scores our models achieved on this task. (Based on f1_scores)

Submission and Description	Public Score ^①
 b32_e5 Jr1e6.csv Complete · now	0.81703
 b16_e5 Jr1e6.csv Complete · 15s ago	0.82347
 b32_e2 Jr1e6.csv Complete · 1m ago	0.76739
 b16_e2 Jr1e6.csv Complete · 15h ago	0.81244
 b32_e5 Jr1e5.csv Complete · 15h ago	0.82071
 b16_e5 Jr1e5.csv Complete · 15h ago	0.8247
 b32_e2 Jr1e5.csv Complete · 15h ago	0.83236
 b16_e2 Jr1e5.csv Complete · 15h ago	0.83757

6 Conclusion and Future Work

As we saw, our models performed close to each other. In these models, we were mostly focused on the hyperparameter tuning, namely learning rate, batch_size and epochs. They all scored decent f1_scores around 0.8. If we want to take more systematic approach, we could add some callback functions to monitor the loss and use early_stopping to not overfit. We can also take use of learning_rate scheduler to find the optimal one. I also tried some basic cleaning, which is provided below for reference, but they caused a slight degradation to the f1_score(around 0.03) that shows we need to fine_tune our cleaning even more to result in any improvement. One may wish to explore with all the combinations and the list is not exhaustive at all, but here are a few that may be beneficial[4]:

- 1 - Getting chat abbreviations: There are some social "slang" lingo we need to be aware of and we probably need to convert them to regular text for our models to perform best. such as :

```
( '*4u', 'Kiss for you' )  
( '*67', 'unknown' )
```

```
(*eg*', 'evil grin')
('07734', 'hello')
('0day', 'software illegally obtained before it was
released')
('0noe', 'Oh No')
('0vr', 'over')
('10q', 'thank you')
('10tacle', 'tentacle')
('10x', 'thanks')
```

- 2 - We may need to remove the urls in the tweets.
- 3 - There are some hashtags and emojis in the tweets that we need to be aware of.
- 4 - Trying out different optimizers(We chose Adam for our models)

7 References

- [1]<https://www.kaggle.com/c/nlp-getting-started/overview>
- [2] KerasNLP starter notebook Disaster Tweets at <https://www.kaggle.com/code/alexia/kerasnlp-starter-notebook-disaster-tweets>
- [3] Disaster Tweet BERT at <https://www.kaggle.com/code/bkassem/disaster-tweet-bert>
- [4] Disaster Tweets Cleaning at <https://www.kaggle.com/code/bkassem/disastor-tweets-cleaning/notebook>