**Problem Statement:**

Predicting whether a given tweet is about a real disaster or not.

The purpose of the project is to determine from tweets (which are microblogs with a limit of 280 characters only) whether it is pointing towards an actual disaster or not.

The dataset contains total 7613 records.

Below are description of the attributes and target label in our dataset.

The dataset contains 5 columns –

|  |  |
| --- | --- |
| id - a unique identifier for each tweet | keyword - a particular keyword from the tweet (may be blank) |
| text - the text of the tweet | target - this denotes whether a tweet is about a real disaster (1) or not (0) |
| location - the location the tweet was sent from (may be blank) |  |

**Solution:**

The solution has been designed using Python 3.7 and Jupyter Notebook as IDE.

# Step 1: Import Libraries and Necessary Installation

Load the necessary libraries and do required installation.

**Step 2: Read file**

Loaded the Twitter dataset ("tweets\_final\_project\_data.csv") using pandas read\_csv into a dataframe named “df”.

**Step 3: Exploratory Data Analysis (EDA)**

Step 3i: Missing Values and Unique values

The 0.8 % of the keywords are missing in the full dataset. Number of Unique values are 221.

33% of the dataset do not contain a location. Number of unique values are 3341. With such a huge number of unique values and a high percentage of missing values, location does not look like insightful for the solution purpose.

Step 3ii: Tweet distribution and cardinality with respect to keywords

Some of the critical observations here are the certain keywords which are solely showing up for a tweet related to actual disaster. Hence it looks evident that the % of keyword value population in the dataset is high and it is also playing an important role in labelling a tweet as disaster/non-disaster.

Step 3iii: Handling Metafeatures

Distributions of meta features in classes and datasets can be helpful to identify disaster tweets. We plotted the distribution of meta features. It looks like disaster tweets are written in a more formal way with longer words compared to non-disaster tweets because most of them are coming from news agencies. Non-disaster tweets have more typos than disaster tweets because they are coming from individual users.

The meta features used for the analysis are:

Table

Description automatically generated

Step 3iv: Class Distribution of Target

Class distributions are 57% for 0 (Not Disaster) and 43% for 1 (Disaster). Classes are almost equally distributed. Hence it does not require any stratification by target in cross validation.

Step 3v: Unigrams, Bigrams and Trigrams

The distribution of Unigrams, Bigrams and Trigrams are plotted with respect to Target variable to identify any hidden pattens. Though the unigrams did not show much of significance , but bigrams and trigrams are showing considerable patterns.

**Step 4: Feature Engineering**

Step 4i: Tweet Cleansing

As part of tweet cleaning, we cleaned the following by hardcoding:

* Contractions
* Character Entity references
* Typos, slang and informal abbreviations
* Hashtags and usernames
* Acronyms

We cleaned the following programmatically using regular expression package in Python and escape character:

* URLs
* Words with Punctuation or Special character
* …and..

Step 4ii: Vocabulary and text coverage Pre-cleaning vs post-Cleaning

We have used Glove and FastText in order to check the vocabulary and text coverage.

GloVe stands for global vectors for word representation. It is an unsupervised learning algorithm developed by Stanford for generating word embeddings by aggregating global word-word co-occurrence matrix from a corpus. The resulting embeddings show interesting linear substructures of the word in vector space. It gives the global statistics. Hence it’s better to use Glove when compared to Word2Vec as the latter gives local statistics.

FastText is a library for learning of word embeddings and text classification created by Facebook's AI Research lab. The model allows one to create an unsupervised learning or supervised learning algorithm for obtaining vector representations for words. It gives Local statistics. However, unlike Word2Vec, which under the hood uses words to predict words, FastText operates at a more granular level with character n-grams.

The Vocabulary and text coverage percentage are given below:

Pre-Cleaning: GloVe and FastText embeddings have more than 50% vocabulary and 80% text coverage without cleaning.

Post-Cleaning: GloVe and FastText embeddings have more than 80% vocabulary and 90% text coverage after cleaning.

Step 4iii: Detection and correction of mislabeled tweets

18 unique tweets were detected which have multiple instances labelled differently for the target values. The target values have been relabeled to minimize any impact on the accuracy. The relabeling has been done by hardcoding, as the number of such tweets are considerably low.

**Step 5: Train-Test Split & Cross Validation**

Train-test split has been done in 90-10% ratio. 6851 records has been included in the Train dataset and remaining 762 records in test dataset.

X\_Train and Y\_Train has been merged again to have a full Train dataset in order to perform Cross-validation.

Cross-validation is a resampling procedure used to validate machine learning models on a limited data set. The procedure has a single parameter called K that refers to the number of groups that a given data sample is to be split into, that’s the reason why it´s called K-fold.

The K-value considered for this purpose is 2. It has been kept low intentionally to maintain considerable re-sampling subset.

**Step 6: Model Training**

A key component of any NLP project is the ability to rapidly test and iterate using techniques. Keras offers a very quick way to prototype state-of-the-art deep learning models and is therefore an important tool we use in our work.

We tried using Logistic Regression Model, but the accuracy was less than 85%. Hence, we used Bert to achieve the required accuracy. We decided to use BERT because it is designed to help computers understand the meaning of ambiguous language in text by using surrounding text to establish context. The BERT framework was pre-trained using text from Wikipedia and can be fine-tuned with question-and-answer datasets.

[BERT](https://arxiv.org/abs/1810.04805), which stands for **Bidirectional Encoder Representations from Transformers**, is based on Transformers, a deep learning model in which every output element is connected to every input element, and the weightings between them are dynamically calculated based upon their connection. It is a language model introduced by Google, uses transformers and pre-training to achieve state-of-the-art on many language tasks. It has recently been added to TensorFlow hub, which simplifies integration in Keras models.

This model uses the implementation of BERT from the TensorFlow Models repository on GitHub at tensorflow/models/official/nlp/bert. It uses L=12 hidden layers (Transformer blocks), a hidden size of H=768, and A=12 attention heads. This model has been pre-trained for English on the Wikipedia and BooksCorpus. Inputs have been "uncased", meaning that the text has been lower-cased before tokenization into word pieces, and any accent markers have been stripped. In order to download this model.

Parameters such as lr (learning rate), epochs (the number of epochs is a hyperparameter of gradient descent that controls the number of complete passes through the training dataset) and batch\_size (the batch size is a hyperparameter of gradient descent that controls the number of training samples to work through before the model’s internal parameters are updated) are used for controlling the learning process. The BERT author’s recommended values are used for these parameters. The Bert authors recommend the following parameters.

batch sizes: 8, 16, 32, 64, 128

learning rates: 3e-4, 1e-4, 5e-5, 3e-5

There are no dense or pooling layers added after last layer of BERT. SGD is used as optimizer since others have hard time while converging.

The training evaluation is given below:

Epoch 10/10

108/108 [==============================] - ETA: 0s - loss: 0.3193 - accuracy: 0.8739

Epoch: 10 - Training Precision: 0.886997 - Training Recall: 0.867059 - Training F1: 0.873511

Epoch: 10 - Validation Precision: 0.848813 - Validation Recall: 0.831763 - Validation F1: 0.837226

108/108 [==============================] - 8227s 76s/step - loss: 0.3193 - accuracy: 0.8739 - val\_loss: 0.3846 - val\_accuracy: 0.8444

Testing evaluation and prediction is given below:

The Accuracy and F1 score for 10 random train and test split are as following:

1. Accuracy score of the Testing Data in random Split 1 is 0.8674540682414699

F1 score of the Testing Data in random Split 1 is 0.8336079077429983

1. Accuracy score of the Testing Data in random Split 2 is 0.8608923884514436

F1 score of the Testing Data in random Split 2 is 0.8215488215488216

1. Accuracy score of the Testing Data in random Split 3 is 0.8595800524934383

F1 score of the Testing Data in random Split 3 is 0.8183361629881153

1. Accuracy score of the Testing Data in random Split 4 is 0.8648293963254593

F1 score of the Testing Data in random Split 4 is 0.8308702791461412

1. Accuracy score of the Testing Data in random Split 5 is 0.8700787401574803

F1 score of the Testing Data in random Split 5 is 0.8352745424292847

1. Accuracy score of the Testing Data in random Split 6 is 0.8740157480314961

F1 score of the Testing Data in random Split 6 is 0.8405315614617941

1. Accuracy score of the Testing Data in random Split 7 is 0.8582677165354331

F1 score of the Testing Data in random Split 7 is 0.8105263157894737

1. Accuracy score of the Testing Data in random Split 8 is 0.8622047244094488

F1 score of the Testing Data in random Split 8 is 0.8292682926829268

1. Accuracy score of the Testing Data in random Split 9 is 0.8727034120734908

F1 score of the Testing Data in random Split 9 is 0.8358714043993232

1. Accuracy score of the Testing Data in random Split 10 is 0.8648293963254593

F1 score of the Testing Data in random Split 10 is 0.8308702791461412

The average Accuracy Score and F1 scores are **86.59%** and **0.83** respectively.