# KI-Lab Project

Project: NLP with Disaster Tweets
Team: NTUST

Member: Yung-Tai Tai, Jou-Chen Liu, Yi-Jing Lian

# Agenda

- Mission
- Dataset
- Preprocessing
- Method and Evaluation
    - XGBoost x TF-IDF
    - LSTM & GRU (with word2Vec)
    - DistilBERT
- Conclusion
- Reference

# Mission

- In this mission, the challenge is to build multiple models to predict which Tweets are about real disasters and which are not.

# Dataset

- The dataset has the following fields: keyword, location, text, target
- Our goal is to use “keyword, location, text” as features to predict the “target.”
- “1” means there is a real disaster, and “0” means there is no disaster.
- Devide train.csv into 80% train, 20% test (since in test.csv, there’s no ground truth)

[1]:
```python
import pandas as pd
dataset = pd.read_csv('data/disaster_train.csv')
dataset
```

[1]:

| | 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 |
| ... | ... | ... | ... | ... | ... |
| 7608 | 10869 | NaN | NaN | Two giant cranes holding a bridge collapse int... | 1 |
| 7609 | 10870 | NaN | NaN | @aria_ahrary @TheTawniest The out of control w... | 1 |
| 7610 | 10871 | NaN | NaN | M1.94 [01:04 UTC]?5km S of Volcano Hawaii. htt... | 1 |
| 7611 | 10872 | NaN | NaN | Police investigating after an e-bike collided ... | 1 |
| 7612 | 10873 | NaN | NaN | The Latest: More Homes Razed by Northern Calif... | 1 |

7613 rows × 5 columns

# Preprocessing

- Lots of null values in “keyword” and “location” columns.
- To make the most use of the dataset, we don’t want to abandon any column.
- Fill “missing” to replace NaN values in “keyword” column.
- Fill “unknown” to replace NaN values in “location” column.

# Preprocessing

- Originally 0.8% values are “NaN” in “keyword” column.
- Originally 33.272% values are “NaN” in “location” column.

[1]:

| | 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 |
| ... | ... | ... | ... | ... | ... |
| 7608 | 10869 | NaN | NaN | Two giant cranes holding a bridge collapse int... | 1 |
| 7609 | 10870 | NaN | NaN | @aria_ahrary @TheTawniest The out of control w... | 1 |
| 7610 | 10871 | NaN | NaN | M1.94 [01:04 UTC]?5km S of Volcano Hawaii. htt... | 1 |
| 7611 | 10872 | NaN | NaN | Police investigating after an e-bike collided ... | 1 |
| 7612 | 10873 | NaN | NaN | The Latest: More Homes Razed by Northern Calif... | 1 |

7613 rows × 5 columns

[2]:
```
dataset.isnull().sum()
```

[2]:
```
id             0
keyword       61
location    2533
text           0
target         0
dtype: int64
```

[2]:

| | id | keyword | location | text | target |
|---|---|---|---|---|---|
| 0 | 1 | missing | unknown | Our Deeds are the Reason of this #earthquake M... | 1 |
| 1 | 4 | missing | unknown | Forest fire near La Ronge Sask. Canada | 1 |
| 2 | 5 | missing | unknown | All residents asked to 'shelter in place' are ... | 1 |
| 3 | 6 | missing | unknown | 13,000 people receive #wildfires evacuation or... | 1 |
| 4 | 7 | missing | unknown | Just got sent this photo from Ruby #Alaska as ... | 1 |
| ... | ... | ... | ... | ... | ... |
| 7608 | 10869 | missing | unknown | Two giant cranes holding a bridge collapse int... | 1 |
| 7609 | 10870 | missing | unknown | @aria_ahrary @TheTawniest The out of control w... | 1 |
| 7610 | 10871 | missing | unknown | M1.94 [01:04 UTC]?5km S of Volcano Hawaii. htt... | 1 |
| 7611 | 10872 | missing | unknown | Police investigating after an e-bike collided ... | 1 |
| 7612 | 10873 | missing | unknown | The Latest: More Homes Razed by Northern Calif... | 1 |

7613 rows × 5 columns

[4]:
```
# 簡單檢查
print(dataset.isnull().sum())
```

```
id          0
keyword     0
location    0
text        0
target      0
dtype: int64
```

# Method 1: XGBoost x TF-IDF

# Step 1. TF-IDF

- Each field (keyword, location, text) is vectorized using TF-IDF to preserve its features.
- It not only extracts frequently occurring words, but also filters out low-information words based on their global importance.
- Compared with the bag-of-words model, TF-IDF is more effective because it considers the global distribution of word frequencies.
- The max_features parameters control the number of features to be retained for each field, and are currently set randomly.

```
# Define features (TF-IDF for each column) and target
vectorizer_keyword = TfidfVectorizer(max_features=100)
vectorizer_location = TfidfVectorizer(max_features=3000)
vectorizer_text = TfidfVectorizer(max_features=5000)

X_keyword = vectorizer_keyword.fit_transform(dataset['keyword'])
X_location = vectorizer_location.fit_transform(dataset['location'])
X_text = vectorizer_text.fit_transform(dataset['text'])
```

# Step 2. Combine features

- Three TF-IDF feature matrices are combined into a complete feature matrix X_combined.
- Different feature sources maintain their independence but can be considered in the model as a whole.

```
# Combine all TF-IDF features
from scipy.sparse import hstack
X_combined = hstack([X_keyword, X_location, X_text])

y = dataset['target']
```

# Step 3. Train-test split

- As we mentioned at beginning, we make it 80/20.

```
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X_combined, y, test_size=0.2, random_state=42)
```

# Step 4. Training XGBoost Classifier

- Training with XGBoost (Extreme Gradient Boosting) Classifier
- eval_metric='logloss': use cross-entropy loss function, suitable for dichotomous problems.

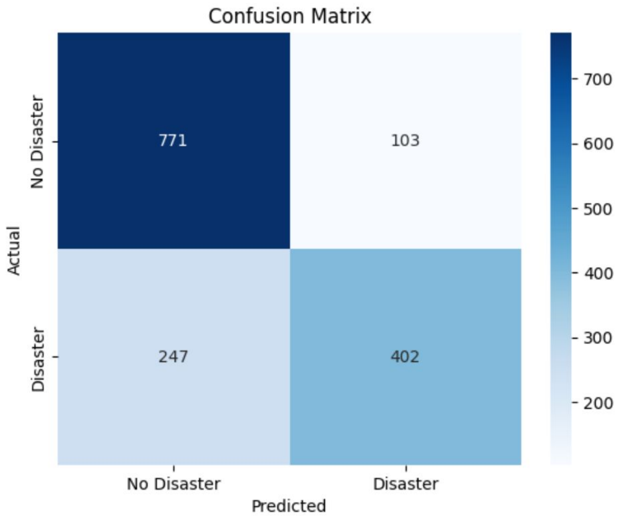
```
# Train XGBoost classifier
xgb_model = xgb.XGBClassifier(use_label_encoder=False, eval_metric='logloss', random_state=42)
xgb_model.fit(X_train, y_train)

# Predictions and evaluation
y_pred = xgb_model.predict(X_test)
print(classification_report(y_test, y_pred))
```

# Step 5. Test if data cleaning is needed

- To check if data cleaning is needed or not, we did a small experiment.
- We kept all the details the same and only made changes to “whether to delete the stop words”.
- "Stop words" are, for example: “a,” “the,” “is,” “are,” etc. Very frequent occurrence of words in the language or very frequent occurrence of words in text data.

# Step 5. Test if data cleaning is needed

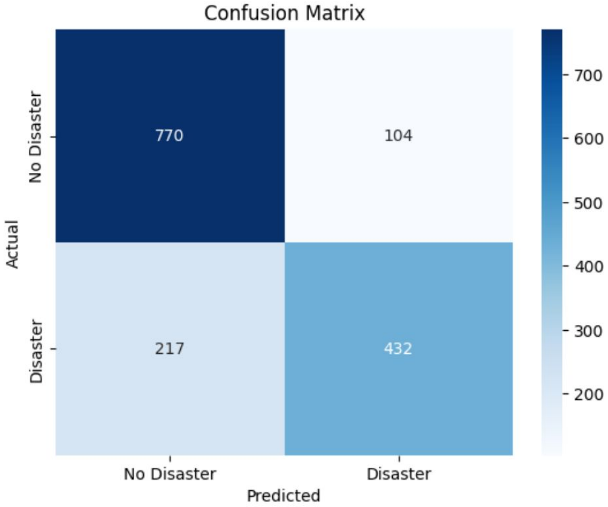
- The result(after deleting stopwords):




| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.76 | 0.88 | 0.82 | 874 |
| 1 | 0.80 | 0.62 | 0.70 | 649 |
| accuracy | | | 0.77 | 1523 |
| macro avg | 0.78 | 0.75 | 0.76 | 1523 |
| weighted avg | 0.77 | 0.77 | 0.76 | 1523 |

# Step 5. Test if data cleaning is needed

- The result(after keeping stopwords):




| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.78 | 0.88 | 0.83 | 874 |
| 1 | 0.81 | 0.67 | 0.73 | 649 |
| accuracy | | | 0.79 | 1523 |
| macro avg | 0.79 | 0.77 | 0.78 | 1523 |
| weighted avg | 0.79 | 0.79 | 0.79 | 1523 |

# Step 6. Parameter Optimization

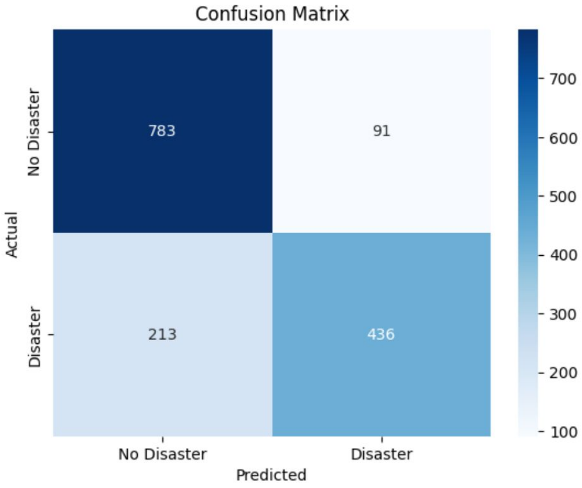
- Since the previous parameter was set randomly, it is now being optimized.
- Use “grid search” to set multiple parameters for optimization.

```
# Grid Search parameters
param_grid = {
    'max_features_keyword': [100, 150, 200, 300],
    'max_features_location': [500, 1000, 2000, 2500],
    'max_features_text': [1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000, 9000, 10000, 11000, 12000, 13000, 14000, 15000, 16000, 17000, 18000, 19000, 200
}
```

- Best Parameters: {'max_features_keyword': 300, 'max_features_location': 500, 'max_features_text': 4000}

# Step 6. Parameter Optimization

- The result(after using optimaized parameters):




| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.79 | 0.90 | 0.84 | 874 |
| 1 | 0.83 | 0.67 | 0.74 | 649 |
| accuracy | | | 0.79 | 1523 |
| macro avg | 0.81 | 0.78 | 0.79 | 1523 |
| weighted avg | 0.80 | 0.80 | 0.80 | 1523 |

# Method 2: LSTM & GRU (with word2Vec)

# LSTM (with word2Vec)

1. Load dataset
2. Data Cleaning
3. Tokenization, Padding and Word2Vec
4. Build a LSTM model
5. Train model
6. Model Evaluation

**LSTM (Long Short-Term Memory)**: A type of recurrent neural network (RNN) designed to handle long-term dependencies in sequence data. It uses memory cells and gates (input, forget, and output gates) to decide what information to keep or discard, making it effective for tasks like text generation and time series prediction.
**Word2Vec**: A technique to represent words as dense numerical vectors based on their context in a corpus. It captures semantic relationships between words, such as similarity and analogy, and is commonly used as input embeddings for NLP models.

# GRU (with word2Vec)

1. Load dataset
2. Data Cleaning
3. Tokenization, Padding and Word2Vec
4. Build a GRU model
5. Train model
6. Model Evaluation

**GRU (Gated Recurrent Unit)**: A simplified version of LSTM that uses fewer gates (update and reset gates) while maintaining similar performance. GRU is computationally faster and easier to train, often used for similar tasks as LSTM.

# Step 1. Data Cleaning

```python
def remove_URL(text):
    url = re.compile(r'https?://\S+|www\.\S+')
    print(url)

    return url.sub('', text)
```

```python
def remove_html(text):
    html = re.compile(r'<.*?>')

    return html.sub('', text)
```

```python
def remove_emoji(text):
    emoji_pattern = re.compile("["
                               u"\U0001F600-\U0001F64F"  # emoticons
                               u"\U0001F300-\U0001F5FF"  # symbols & pictographs
                               u"\U0001F680-\U0001F6FF"  # transport & map symbols
                               u"\U0001F1E0-\U0001F1FF"  # flags (iOS)
                               u"\U00002702-\U000027B0"
                               u"\U000024C2-\U0001F251"
                               "]+", flags=re.UNICODE)

    return emoji_pattern.sub(r'', text)
```

```python
def remove_punct(text):
    table = str.maketrans('', '', string.punctuation)

    return text.translate(table)
```

# Step 2. Tokenize

Use nltk to tokenize word

```
print(nltk.word_tokenize(dataset.loc[0,'text']))
dataset['tokenized_text'] = dataset['text'].apply(lambda x: nltk.word_tokenize(x))
dataset['tokenized_text']
```

# Step 3. word2Vec

Use tokenized text column to build Word2Vec model

```
[ ]  import pandas as pd
     from gensim.models.word2vec import Word2Vec
```

```
[ ]  w2v_model = Word2Vec(dataset['tokenized_text_remove_stopword'])
```

# Step 4. Text to Index

Use word2Vector model to buid vocab_list and word2idx list.

```
vocab_list = [(word, w2v_model.wv[word]) for word in w2v_model.wv.index_to_key]
for i, vocab in enumerate(vocab_list):
        word, vec = vocab
        word2idx[word] = i + 1
```

# Step 4. Text to Index

```
from keras.preprocessing.sequence import pad_sequences
import numpy as np

def text_to_index(corpus, word2idx):
        new_corpus = []
        for doc in corpus:
                new_doc = []
                for word in doc:
                        try:
                                new_doc.append(word2idx[word])
                        except KeyError:
                                new_doc.append(0)    # Use 0 for words not found in the vocabulary
                new_corpus.append(new_doc)
        return new_corpus

PADDING_LENGTH = 200
X = text_to_index(dataset['tokenized_text'], word2idx)
X = pad_sequences(X, maxlen=PADDING_LENGTH)

print("Shape:", X.shape)
print("Sample:", X[0])
```

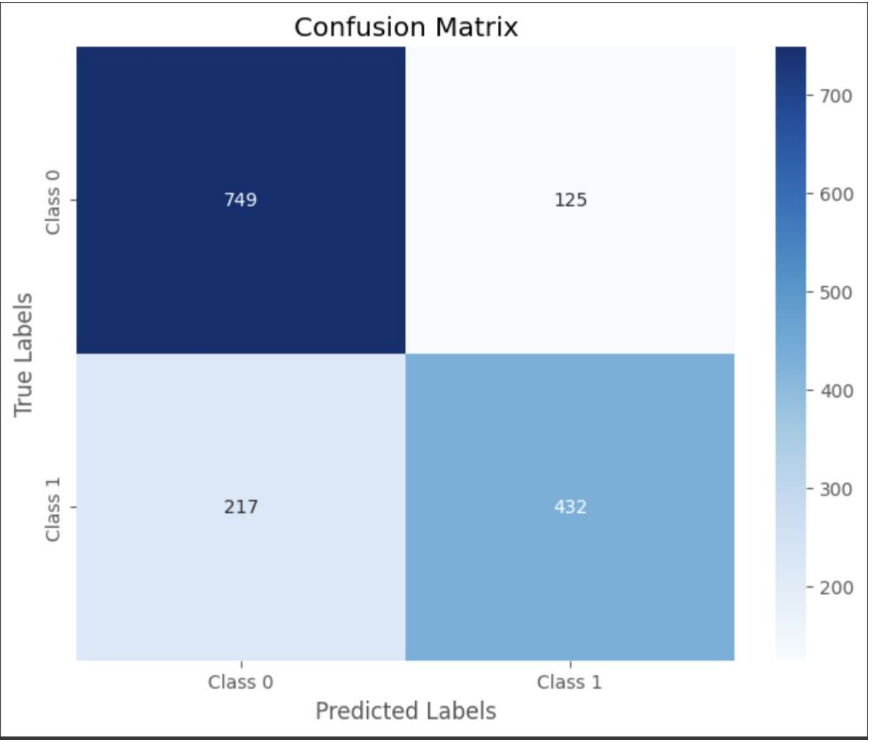
# LSTM model

```
class BinaryLSTMClassifier(nn.Module):
    def __init__(self, embedding_dim, hidden_dim, vocab_size, padding_idx):
        super(BinaryLSTMClassifier, self).__init__()
        self.embedding = nn.Embedding(vocab_size, embedding_dim, padding_idx=padding_idx)
        self.lstm = nn.LSTM(embedding_dim, hidden_dim, batch_first=True, bidirectional=True)
        self.dropout = nn.Dropout(0.5)
        self.fc = nn.Linear(hidden_dim * 2, 1)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        embedded = self.embedding(x)
        lstm_out, _ = self.lstm(embedded)
        lstm_out = self.dropout(lstm_out)
        hidden_state = lstm_out[:, -1, :]
        output = self.fc(hidden_state)
        return self.sigmoid(output)
```

# Early stop

```
# Early Stopping
if test_losses[-1] < best_loss:
        best_loss = test_losses[-1]
        patience_counter = 0
else:
        patience_counter += 1
        if patience_counter >= patience:
                print("Early stopping triggered!")
                break
```

# Result

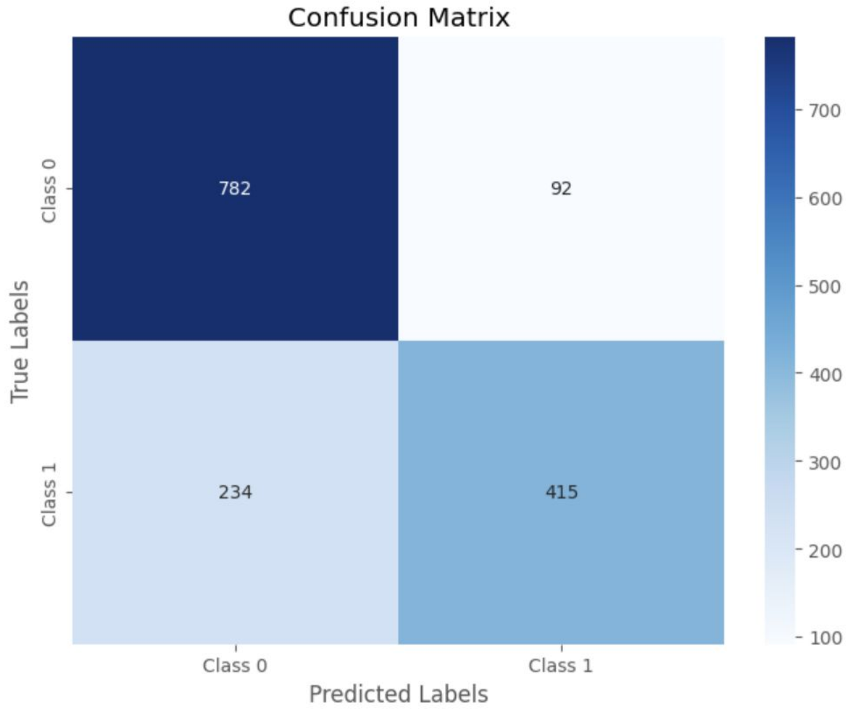



| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.78 | 0.86 | 0.81 | 874 |
| 1 | 0.78 | 0.67 | 0.72 | 649 |
| accuracy | | | 0.78 | 1523 |
| macro avg | 0.78 | 0.76 | 0.77 | 1523 |
| weighted avg | 0.78 | 0.78 | 0.77 | 1523 |

# GRU model

```
class BinaryGRUClassifier(nn.Module):
    def __init__(self, embedding_dim, hidden_dim, vocab_size, padding_idx):
        super(BinaryGRUClassifier, self).__init__()
        self.embedding = nn.Embedding(vocab_size, embedding_dim, padding_idx=padding_idx)
        self.gru = nn.GRU(embedding_dim, hidden_dim, batch_first=True, bidirectional=True)
        self.dropout = nn.Dropout(0.5)
        self.fc = nn.Linear(hidden_dim * 2, 1)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        embedded = self.embedding(x)
        gru_out, _ = self.gru(embedded)
        gru_out = self.dropout(gru_out)
        hidden_state = gru_out[:, -1, :]
        output = self.fc(hidden_state)
        return self.sigmoid(output)
```

# Result




| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.77 | 0.89 | 0.83 | 874 |
| 1 | 0.82 | 0.64 | 0.72 | 649 |
| accuracy | | | 0.79 | 1523 |
| macro avg | 0.79 | 0.77 | 0.77 | 1523 |
| weighted avg | 0.79 | 0.79 | 0.78 | 1523 |

# Method 3: DistilBERT

# DistilBERT

1. Load dataset
2. Load a DistilBERT tokenizer to preprocess the “text” field
3. Create a preprocessing function to tokenize “text” and truncate sequences
4. Apply the preprocessing function over the entire dataset
5. Train model
6. Model Evaluation
7. Hyperparameter Tuning

# Step 1. Data Cleaning

```python
def remove_URL(text):
    url_pattern = re.compile(r'https?://\S+|www\.\S+')

    return url_pattern.sub('', text)

def remove_html(text):
    html_pattern = re.compile(r'<.*?>')

    return html_pattern.sub('', text)

def remove_emoji(text):
    emoji_pattern = re.compile("["
                           u"\U0001F600-\U0001F64F"  # emoticons
                           u"\U0001F300-\U0001F5FF"  # symbols & pictographs
                           u"\U0001F680-\U0001F6FF"  # transport & map symbols
                           u"\U0001F1E0-\U0001F1FF"  # flags (iOS)
                           u"\U00002702-\U000027B0"
                           u"\U000024C2-\U0001F251"
                           "]+", flags=re.UNICODE)
    return emoji_pattern.sub(r' ', text)

def remove_punct(text):
    translator = str.maketrans('', '', string.punctuation)

    return text.translate(translator)
```

# Step 2. Tokenize

- Load a DistilBERT tokenizer: distilbert-base-uncased
- Create a preprocessing function to tokenize “text” and truncate sequences
- Use DataCollatorWithPadding to make the length same

```
from transformers import AutoTokenizer, DataCollatorWithPadding

tokenizer = AutoTokenizer.from_pretrained("distilbert/distilbert-base-uncased")

def preprocess_function(examples):
    return tokenizer(examples["text"], truncation=True)

tokenized_disaster = raw_datasets.map(preprocess_function, batched=True)
data_collator = DataCollatorWithPadding(tokenizer=tokenizer)
print(tokenized_disaster)
```

# Step 3. Construct Model

- Load a Pre-trained Sequence Classification Model: distilbert-base-uncased
- Set the number of categories for the classification task as 2

```python
from transformers import AutoModelForSequenceClassification

model = AutoModelForSequenceClassification.from_pretrained(
    "distilbert/distilbert-base-uncased", num_labels=2
)
```

# Step 4.Train - (1) EarlyStopping

- Stop training when performance plateaued, mainly to prevent overfitting.

```python
stopped_epoch = 0  # record early stopping epochs

class EarlyStoppingCallback(TrainerCallback):
    def __init__(self, patience, min_delta):
        self.patience = patience  # patience=3: Default allows up to 3 epochs without progress.
        self.min_delta = min_delta  # min_delta=0: Any improvement is considered progress.
        self.counter = 0  # Counter for epochs without improvement
        self.early_stop = False
        self.best_score = None  # track the best evaluation score

    def on_evaluate(self, args, state, control, metrics=None, **kwargs):
        global stopped_epoch

        # train_loss = metrics.get("train_loss")
        eval_loss = metrics.get("eval_loss")

        if eval_loss is not None:
            if self.best_score is None or (self.best_score - eval_loss) > self.min_delta:
                self.best_score = eval_loss
                self.counter = 0  # Reset counter if improvement is seen
            else:
                self.counter += 1
                if self.counter >= self.patience:  # stop training
                    self.early_stop = True
                    control.should_training_stop = True
                    stopped_epoch = state.epoch
                    print(f"Early stopping triggered at epoch {stopped_epoch}")
```

# Step 4.Train - (2) Training parameter settings

- Define training hyperparameters

```
training_args = TrainingArguments(
    output_dir="my_model",
    learning_rate=2e-5,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=16,
    num_train_epochs=10,
    weight_decay=0.01,
    eval_strategy="epoch",
    save_strategy="epoch",
    logging_dir='./logs',
    logging_strategy="epoch", # record every epoch
    report_to="none",  # Disable W&B logging
)
```

# Step 4.Train - (3) Optimizer and Scheduler

```
optimizer = optim.Adam(model.parameters(), lr=1e-5, weight_decay=1e-5)

scheduler = get_scheduler(
    name="linear",  # Scheduler type
    optimizer=optimizer,
    num_warmup_steps=0,  # Number of warmup steps
    num_training_steps=training_args.num_train_epochs * len(tokenized_disaster["train"]) // training_args.per_device_train_batch_size
)
```

# Step 4.Train - (4) Evaluation metrics

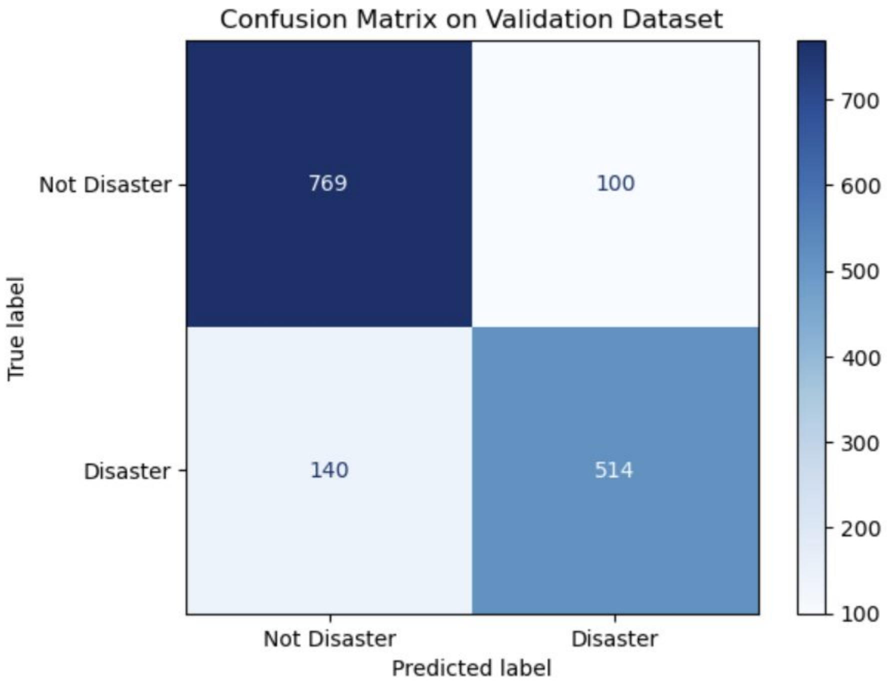
- Create a function that passes the predictions and labels to compute to calculate the accuracy

```
def compute_metrics(eval_pred):
    metric = evaluate.load("glue", "mrpc") # F1 and Accuracy
    predictions, labels = eval_pred
    predictions = np.argmax(predictions, axis=1)
    return metric.compute(predictions=predictions, references=labels)
```

# Step 4.Train - (5) Trainer

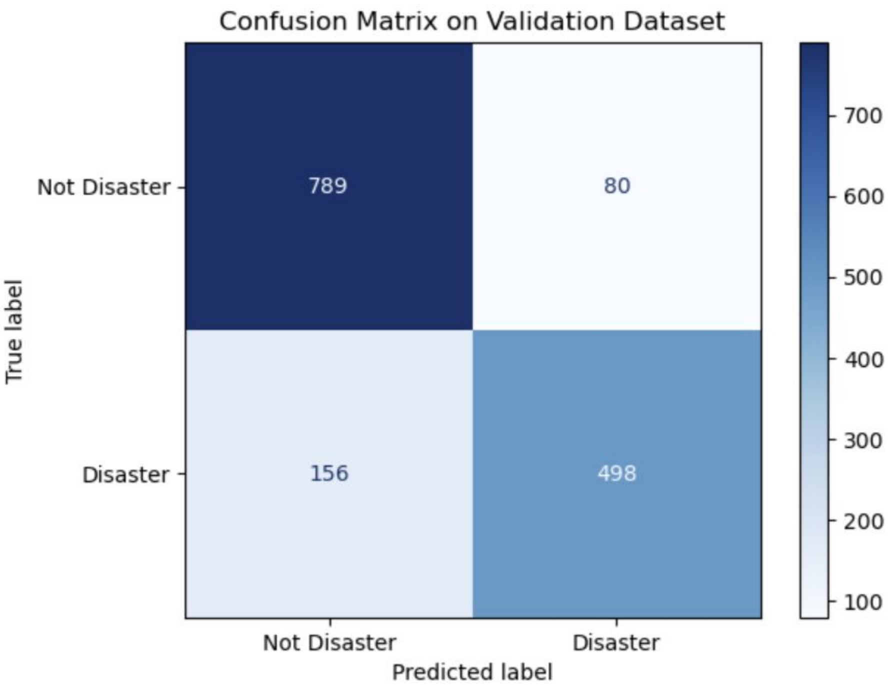
```
trainer = Trainer(
    model=model,
    args=training_args,
    optimizers=(optimizer, scheduler),
    train_dataset=tokenized_disaster["train"],
    eval_dataset=tokenized_disaster["eval"],
    processing_class=tokenizer,
    data_collator=data_collator,
    compute_metrics=compute_metrics,
    callbacks=[EarlyStoppingCallback(patience=3, min_delta=0)],
)

trainer.train()
```

# Result




| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.84 | 0.88 | 0.86 | 869 |
| 1 | 0.83 | 0.78 | 0.81 | 654 |
| accuracy | | | 0.84 | 1523 |
| macro avg | 0.84 | 0.83 | 0.83 | 1523 |
| weighted avg | 0.84 | 0.84 | 0.84 | 1523 |

# Result (Without data cleaning)




| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.83 | 0.90 | 0.87 | 869 |
| 1 | 0.86 | 0.76 | 0.80 | 654 |
| accuracy | | | 0.84 | 1523 |
| macro avg | 0.84 | 0.83 | 0.83 | 1523 |
| weighted avg | 0.84 | 0.84 | 0.84 | 1523 |

# Conclusion

| Method | XGBoost x TF-IDF | LSTM x Word2Vec | GRU x Word2Vec | DistilBERT |
|---|---|---|---|---|
| Accuracy | 0.80 | 0.77 | 0.78 | 0.84 |

# Reference


[1] “Natural Language Processing with Disaster Tweets,” Kaggle. Available: https://www.kaggle.com/competitions/nlp-getting-started

[2] “Kaggle灾难推文的自然语言处理-最佳得分详解-CSDN博客.” Available: https://blog.csdn.net/StrawBerryTreea/article/details/131948632

[3] S. Hochreiter and J. Schmidhuber, "Long short-term memory," Neural Computation, vol. 9, no. 8, pp. 1735-1780, Nov. 1997.

[4] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," in Proc. Int. Conf. Learning Representations (ICLR), 2013, pp. 1-12.

[5] K. Cho et al., "Learning phrase representations using RNN encoder-decoder for statistical machine translation," in Proc. Conf. Empirical Methods in Natural Language Processing (EMNLP), 2014, pp. 1724-1734.

[6] “DistilBERT.” Available: https://huggingface.co/docs/transformers/model_doc/distilbert

[7] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, “DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter,” arXiv.org, Oct. 02, 2019. https://arxiv.org/abs/1910.01108

[8] “Text classification.” Available: https://huggingface.co/docs/transformers/en/tasks/sequence_classification

[9] “early stopping in PyTorch,” Stack Overflow. Available: https://stackoverflow.com/questions/71998978/early-stopping-in-pytorch

Thank you