```
##Problem statement
#The aim is to develop a machine learning algorithm to predict whether a tweet is about a real disaster or not.
```

Aproach: Transfer learning technique is used to perform the text classification problem. We load pretrained BERT model and finetune the weights.

Advantages of fine-tuning Time - Pretrained BERT model weights already encode a lot of information. As a result, it takes much less time to finetune the model

Data - As the pretrained model is trained on large text, the model performs well even with small datasets.

We don't go into the details of BERT architecture. Here is an overview about how BERT is pretrained, and how it can be used for classification.

BERT (Bidirectional Encoder Representations from Transformers): Language modeling is a common method of pretraining on unlabeled text (self supervised learning). Most of the language models learned by iteratively predicting next word in a sequence auto regressively across enormous data sets of text like wikepedia. This can be left to right, right to left or bi-directional.

There are two strategies of applying pretrained language representations to downstream tasks:

Feature based approach Fine tuning approach The feauture based approach, such as ELMo uses task specific architectures that include the pretrained representations as additional features.

The fine tuning approach, such as OpenAI GPT, introduces minimal task specific parameters, and is trained on the downstream task by fine tuning all the pretrained parameters.

BERT model can be used for both the approaches. BERT reformulates the language modeling pretrained task of iteratively predicting the next word in sequence to instead incorporate bidirectional context and predict mask of intermediate tokens of the sequence and predict the mask token. BERT presented a new self supervised learning task for pretaining transformers inorder to fine tune them for different tasks. They major difference between BERT and prior methods of pretraining transformer models is using the bidirectional context of language modeling. Most of the models either move left to right or right to left to predict next word in sequence, where BERT tries to learn intermediate tokens (by MASK), making the name Bidirectional Encoder.

BERT uses Masked language model and also use "Next sentence prediction" task.

BERT uses 3 embeddings to compute the input representations. They are token embeddings, segment embeddings and position embeddings.

BERT Transformer will preserve the length of the (dimention of the) input. The final output will take this vector and pass these to seperate tasks (classification, in this case).

BERT for Classification: BERT consists of stacked encoder layers. Just like the input of encoder of the transformer model, BERT model takes the sequence of numeric representation of the tokens as input. For classification tasks, we must prepend the special [CLS] token to the beginning of every sentence.

Encoder block of transformer outputs a vector with same length as of input. First position of the vector, corresponding to the [CLS] token, can now be used as the input for a classifier.

```
# importing necessary libraries

import numpy as np
import pandas as pd
import time
import datetime
import gc
import random
from nltk.corpus import stopwords
import re
```

```
import torch
import torch.nn as nn
from torch.utils.data import TensorDataset, DataLoader, RandomSampler, SequentialSampler,random_split
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
```

```
!pip install transformers
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: transformers in /usr/local/lib/python3.8/dist-packages (4.26.0)
Requirement already satisfied: tokenizers!=0.11.3,<0.14,>=0.11.1 in /usr/local/lib/python3.8/dist-packages (from transformers) (0.13.2)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.8/dist-packages (from transformers) (1.21.6)
Requirement already satisfied: filelock in /usr/local/lib/python3.8/dist-packages (from transformers) (3.9.0)
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.8/dist-packages (from transformers) (4.64.1)
Requirement already satisfied: huggingface-hub<1.0,>=0.11.0 in /usr/local/lib/python3.8/dist-packages (from transformers) (0.12.0)
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.8/dist-packages (from transformers) (2022.6.2)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.8/dist-packages (from transformers) (21.3)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.8/dist-packages (from transformers) (6.0)
Requirement already satisfied: requests in /usr/local/lib/python3.8/dist-packages (from transformers) (2.25.1)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.8/dist-packages (from huggingface-hub<1.0,>=0.11.0-
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /usr/local/lib/python3.8/dist-packages (from packaging>=20.0->transformers) (
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.8/dist-packages (from requests->transformers) (1.24.3)
Requirement already satisfied: chardet<5,>=3.0.2 in /usr/local/lib/python3.8/dist-packages (from requests->transformers) (4.0.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.8/dist-packages (from requests->transformers) (2022.12.7)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.8/dist-packages (from requests->transformers) (2.10)
```

```
import transformers
from transformers import BertForSequenceClassification, AdamW, BertConfig,BertTokenizer,get_linear_schedule_with_warmup
```

```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
device
```

```
device(type='cuda', index=0)
```

```
#reading the dataset
df = pd.read_csv('/content/train.csv')
```

```
df.head()
```

	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

```
!pip install nltk
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: nltk in /usr/local/lib/python3.8/dist-packages (3.7)
Requirement already satisfied: click in /usr/local/lib/python3.8/dist-packages (from nltk) (7.1.2)
Requirement already satisfied: tqdm in /usr/local/lib/python3.8/dist-packages (from nltk) (4.64.1)
Requirement already satisfied: joblib in /usr/local/lib/python3.8/dist-packages (from nltk) (1.2.0)
Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.8/dist-packages (from nltk) (2022.6.2)
```

```
import nltk
```

```
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
True
```

Data preprocessing:

We are using custom functions to perform the following tasks. Cleaning up the data for modeling should be carried out carefully and with the help of subject matter experts, if possible. This cleaning is done completely based on observation, and can not be considered as a generic preprocessing step for all the NLP tasks. This preprocessing function ensures:

Removing urls from tweet Removing html tags Removing punctuations Removing stopwords Removing emoji

Double-click (or enter) to edit

```
sw = stopwords.words('english')

def clean_text(text):

    text = text.lower()

    text = re.sub(r"[^a-zA-Z?.!,¿]+", " ", text) # replacing everything with space except (a-z, A-Z, ".", "?", "!", ",")

    text = re.sub(r"http\S+", "",text) #Removing URLs
    #text = re.sub(r"http", "",text)

    html=re.compile(r'<.*?>')

    text = html.sub(r'',text) #Removing html tags

    punctuations = '@#!?+&*[]-%.:/();$=><|{}^' + "'`" + '_'
    for p in punctuations:
        text = text.replace(p,'') #Removing punctuations

    text = [word.lower() for word in text.split() if word.lower() not in sw]

    text = " ".join(text) #removing stopwords

    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)
    text = emoji_pattern.sub(r'', text) #Removing emojis

    return text
```

```
df['text'] = df['text'].apply(lambda x: clean_text(x))
```

```
tweets = df.text.values
labels = df.target.values
```

```
tweets
```

```
array(['deeds reason earthquake may allah forgive us',
       'forest fire near la ronge sask canada',
       'residents asked shelter place notified officers evacuation shelter place orders expected',
       ..., 'utc km volcano hawaii http tco zdtoyd ebj',
       'police investigating e bike collided car little portugal e bike rider suffered serious non life threatening injuries',
       'latest homes razed northern california wildfire abc news http tco ymy rskq'],
      dtype=object)
```

```
tweets.shape
```

```
(7613,)
```

```
labels.shape
```

```
(7613,)
```

BERT Tokenizer:

In BERT, WordPiece tokenizer (a subword tokenizer) is used for tokenization. A word can be broken down into more than one sub-word, which helps in dealing with unknown words. For best results, it is adviced to tokenize with the same tokenizer the BERT model was trained on.

`Next, we need to convert each token to an id as present in the tokenizer vocabulary. If there's a token that is not present in the vocabulary, the tokenizer will use the special [UNK] token and use its id.`

```
# Load the BERT tokenizer
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased', do_lower_case=True)
```

```
Downloading (…)solve/main/vocab.txt:  100%  232k/232k [00:00<00:00, 2.19MB/s]
```

```
tokenizer
```

```
BertTokenizer(name_or_path='bert-base-uncased', vocab_size=30522, model_max_length=512, is_fast=False, padding_side='right',
truncation_side='right', special_tokens={'unk_token': '[UNK]', 'sep_token': '[SEP]', 'pad_token': '[PAD]', 'cls_token': '[CLS]',
'mask_token': '[MASK]'})
```

```
print(' Original: ', tweets[0])

# Print the sentence split into tokens.
print('Tokenized: ', tokenizer.tokenize(tweets[0]))

# Print the sentence mapped to token ids.
print('Token IDs: ', tokenizer.convert_tokens_to_ids(tokenizer.tokenize(tweets[0])))
```

```
 Original:  deeds reason earthquake may allah forgive us
Tokenized:  ['deeds', 'reason', 'earthquake', 'may', 'allah', 'forgive', 'us']
Token IDs:  [15616, 3114, 8372, 2089, 16455, 9641, 2149]
```

```
max_len = 0

# For every sentence...
for sent in tweets:

    # Tokenize the text and add `[CLS]` and `[SEP]` tokens.
    input_ids = tokenizer.encode(sent, add_special_tokens=True)

    # Update the maximum sentence length.
    max_len = max(max_len, len(input_ids))

print('Max sentence length: ', max_len)
```

```
 Max sentence length:  45
```

```
input_ids = []
attention_masks = []

# For every tweet...
for tweet in tweets:
    # `encode_plus` will:
    #   (1) Tokenize the sentence.
    #   (2) Prepend the `[CLS]` token to the start.
    #   (3) Append the `[SEP]` token to the end.
    #   (4) Map tokens to their IDs.
    #   (5) Pad or truncate the sentence to `max_length`
    #   (6) Create attention masks for [PAD] tokens.
    encoded_dict = tokenizer.encode_plus(
                        tweet,                      # Sentence to encode.
                        add_special_tokens = True, # Add '[CLS]' and '[SEP]'
                        max_length = max_len,           # Pad & truncate all sentences.
                        pad_to_max_length = True,
                        return_attention_mask = True,   # Construct attn. masks.
                        return_tensors = 'pt',     # Return pytorch tensors.
                   )

    # Add the encoded sentence to the list.
    input_ids.append(encoded_dict['input_ids'])

    # And its attention mask (simply differentiates padding from non-padding).
    attention_masks.append(encoded_dict['attention_mask'])

# Convert the lists into tensors.
input_ids = torch.cat(input_ids, dim=0)
attention_masks = torch.cat(attention_masks, dim=0)
labels = torch.tensor(labels)

# Print sentence 0, now as a list of IDs.
print('Original: ', tweets[0])
print('Token IDs:', input_ids[0])
```

```
Truncation was not explicitly activated but `max_length` is provided a specific value, please use `truncation=True` to explicitly trunca
/usr/local/lib/python3.8/dist-packages/transformers/tokenization_utils_base.py:2339: FutureWarning: The `pad_to_max_length` argument is
  warnings.warn(
```

```
Original:  deeds reason earthquake may allah forgive us
Token IDs: tensor([  101, 15616,  3114,  8372,  2089, 16455,  9641,  2149,   102,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0])
```

Train-validation split:

80% of data is split into train and 20% to validation sets.

Double-click (or enter) to edit

```
# Combine the training inputs into a TensorDataset.
dataset = TensorDataset(input_ids, attention_masks, labels)

# Create a 90-10 train-validation split.

# Calculate the number of samples to include in each set.
train_size = int(0.8 * len(dataset))
#val_size = int(0.2 * len(dataset))
val_size = len(dataset)  - train_size

# Divide the dataset by randomly selecting samples.
train_dataset, val_dataset = random_split(dataset, [train_size, val_size])

print('{:>5,} training samples'.format(train_size))
print('{:>5,} validation samples'.format(val_size))
```

```
6,090 training samples
1,523 validation samples
```

```
# The DataLoader needs to know our batch size for training, so we specify it
# here. For fine-tuning BERT on a specific task, the authors recommend a batch
# size of 16 or 32.
batch_size = 32

# Create the DataLoaders for our training and validation sets.
# We'll take training samples in random order.
train_dataloader = DataLoader(
            train_dataset,  # The training samples.
            sampler = RandomSampler(train_dataset), # Select batches randomly
            batch_size = batch_size # Trains with this batch size.
        )

# For validation the order doesn't matter, so we'll just read them sequentially.
validation_dataloader = DataLoader(
            val_dataset, # The validation samples.
            sampler = SequentialSampler(val_dataset), # Pull out batches sequentially.
            batch_size = batch_size # Evaluate with this batch size.
        )
```

```
# Load BertForSequenceClassification, the pretrained BERT model with a single
# linear classification layer on top.
model = BertForSequenceClassification.from_pretrained(
    "bert-base-uncased", # Use the 12-layer BERT model, with an uncased vocab.
    num_labels = 2, # The number of output labels--2 for binary classification.
                    # You can increase this for multi-class tasks.
    output_attentions = False, # Whether the model returns attentions weights.
    output_hidden_states = False, # Whether the model returns all hidden-states.
)

# if device == "cuda:0":
# # Tell pytorch to run this model on the GPU.
#     model = model.cuda()
model = model.to(device)
```

```
Downloading (…)"pytorch_model.bin";:                    440M/440M [00:03<00:00,
```

```
optimizer = AdamW(model.parameters(),
                  lr = 2e-5, # args.learning_rate - default is 5e-5, our notebook had 2e-5
                  eps = 1e-8 # args.adam_epsilon  - default is 1e-8.
                )
```

```
/usr/local/lib/python3.8/dist-packages/transformers/optimization.py:306: FutureWarning: This implementation of AdamW is deprecated and w
  warnings.warn(
```

Fine tuning the model

```
# Number of training epochs. The BERT authors recommend between 2 and 4.
# We chose to run for 4, but we'll see later that this may be over-fitting the
# training data.
epochs = 4

# Total number of training steps is [number of batches] x [number of epochs].
# (Note that this is not the same as the number of training samples).
total_steps = len(train_dataloader) * epochs

# Create the learning rate scheduler.
scheduler = get_linear_schedule_with_warmup(optimizer,
                                            num_warmup_steps = 0, # Default value in run_glue.py
                                            num_training_steps = total_steps)
```

```
# Function to calculate the accuracy of our predictions vs labels
def flat_accuracy(preds, labels):
    pred_flat = np.argmax(preds, axis=1).flatten()
    labels_flat = labels.flatten()
    return np.sum(pred_flat == labels_flat) / len(labels_flat)
```

```
def format_time(elapsed):
    '''
    Takes a time in seconds and returns a string hh:mm:ss
    '''
    # Round to the nearest second.
    elapsed_rounded = int(round((elapsed)))
    # Format as hh:mm:ss
    return str(datetime.timedelta(seconds=elapsed_rounded))
```

```
seed_val = 42
random.seed(seed_val)
np.random.seed(seed_val)
torch.manual_seed(seed_val)
torch.cuda.manual_seed_all(seed_val)
training_stats = []

# Measure the total training time for the whole run.
total_t0 = time.time()

# For each epoch...
for epoch_i in range(0, epochs):

    # ========================================
    #               Training
    # ========================================
    # Perform one full pass over the training set.
    print("")
    print('======== Epoch {:} / {:} ========'.format(epoch_i + 1, epochs))
    print('Training...')
    # Measure how long the training epoch takes.
    t0 = time.time()
    total_train_loss = 0
    model.train()
    for step, batch in enumerate(train_dataloader):
        # Unpack this training batch from our dataloader.
        #
        # As we unpack the batch, we'll also copy each tensor to the device using the
        # `to` method.
```

```
        #
        # `batch` contains three pytorch tensors:
        #   [0]: input ids
        #   [1]: attention masks
        #   [2]: labels
        b_input_ids = batch[0].to(device)
        b_input_mask = batch[1].to(device)
        b_labels = batch[2].to(device)
        optimizer.zero_grad()
        output = model(b_input_ids,
                            token_type_ids=None,
                            attention_mask=b_input_mask,
                            labels=b_labels)
        loss = output.loss
        total_train_loss += loss.item()
        # Perform a backward pass to calculate the gradients.
        loss.backward()
        # Clip the norm of the gradients to 1.0.
        # This is to help prevent the "exploding gradients" problem.
        torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)
        # Update parameters and take a step using the computed gradient.
        # The optimizer dictates the "update rule"--how the parameters are
        # modified based on their gradients, the learning rate, etc.
        optimizer.step()
        # Update the learning rate.
        scheduler.step()

    # Calculate the average loss over all of the batches.
    avg_train_loss = total_train_loss / len(train_dataloader)

    # Measure how long this epoch took.
    training_time = format_time(time.time() - t0)
    print("")
    print("  Average training loss: {0:.2f}".format(avg_train_loss))
    print("  Training epcoh took: {:}".format(training_time))
    # ========================================
    #               Validation
    # ========================================
    # After the completion of each training epoch, measure our performance on
    # our validation set.
    print("")
    print("Running Validation...")
    t0 = time.time()
    # Put the model in evaluation mode--the dropout layers behave differently
    # during evaluation.
    model.eval()
    # Tracking variables
    total_eval_accuracy = 0
    best_eval_accuracy = 0
    total_eval_loss = 0
    nb_eval_steps = 0
    # Evaluate data for one epoch
    for batch in validation_dataloader:
        b_input_ids = batch[0].to(device)
        b_input_mask = batch[1].to(device)
        b_labels = batch[2].to(device)
        # Tell pytorch not to bother with constructing the compute graph during
        # the forward pass, since this is only needed for backprop (training).
        with torch.no_grad():
            output= model(b_input_ids,
                                token_type_ids=None,
                                attention_mask=b_input_mask,
                                labels=b_labels)
        loss = output.loss
        total_eval_loss += loss.item()
        # Move logits and labels to CPU if we are using GPU
        logits = output.logits
        logits = logits.detach().cpu().numpy()
        label_ids = b_labels.to('cpu').numpy()
        # Calculate the accuracy for this batch of test sentences, and
        # accumulate it over all batches.
        total_eval_accuracy += flat_accuracy(logits, label_ids)
    # Report the final accuracy for this validation run.
    avg_val_accuracy = total_eval_accuracy / len(validation_dataloader)
    print("  Accuracy: {0:.2f}".format(avg_val_accuracy))
    # Calculate the average loss over all of the batches.
    avg_val_loss = total_eval_loss / len(validation_dataloader)
```

```
    # Measure how long the validation run took.
    validation_time = format_time(time.time() - t0)
    if avg_val_accuracy > best_eval_accuracy:
        torch.save(model, 'bert_model')
        best_eval_accuracy = avg_val_accuracy
    #print("  Validation Loss: {0:.2f}".format(avg_val_loss))
    #print("  Validation took: {:}".format(validation_time))
    # Record all statistics from this epoch.
    training_stats.append(
        {
            'epoch': epoch_i + 1,
            'Training Loss': avg_train_loss,
            'Valid. Loss': avg_val_loss,
            'Valid. Accur.': avg_val_accuracy,
            'Training Time': training_time,
            'Validation Time': validation_time
        }
    )
print("")
print("Training complete!")

print("Total training took {:} (h:mm:ss)".format(format_time(time.time()-total_t0)))
```

```
     ======== Epoch 1 / 4 ========
     Training...

       Average training loss: 0.47
       Training epcoh took: 0:00:49

     Running Validation...
       Accuracy: 0.83

     ======== Epoch 2 / 4 ========
     Training...

       Average training loss: 0.37
       Training epcoh took: 0:00:50

     Running Validation...
       Accuracy: 0.85

     ======== Epoch 3 / 4 ========
     Training...

       Average training loss: 0.30
       Training epcoh took: 0:00:51

     Running Validation...
       Accuracy: 0.84

     ======== Epoch 4 / 4 ========
     Training...

       Average training loss: 0.24
       Training epcoh took: 0:00:51

     Running Validation...
       Accuracy: 0.84

     Training complete!
     Total training took 0:03:42 (h:mm:ss)
```

Loading the best model

```
model = torch.load('bert_model')
```

```
df_test = pd.read_csv('/content/test.csv')
```

```
df_test['text'] = df_test['text'].apply(lambda x:clean_text(x))
test_tweets = df_test['text'].values
```

```
df_test['text']
```

```
     0                            happened terrible car crash
     1       heard earthquake different cities, stay safe e...
     2       forest fire spot pond, geese fleeing across st...
```

```
3                  apocalypse lighting spokane wildfires
4                    typhoon soudelor kills china taiwan
                               ...
3258    earthquake safety los angeles safety fasteners...
3259    storm ri worse last hurricane city amp others ...
3260    green line derailment chicago http tco utbxlcbiuy
3261    meg issues hazardous weather outlook hwo http ...
3262    cityofcalgary activated municipal emergency pl...
Name: text, Length: 3263, dtype: object
```

```
test_tweets
```

```
array(['happened terrible car crash',
       'heard earthquake different cities, stay safe everyone',
       'forest fire spot pond, geese fleeing across street, cannot save',
       ..., 'green line derailment chicago http tco utbxlcbiuy',
       'meg issues hazardous weather outlook hwo http tco x rbqjhn',
       'cityofcalgary activated municipal emergency plan yycstorm'],
      dtype=object)
```

```
test_input_ids = []
test_attention_masks = []
for tweet in test_tweets:
    encoded_dict = tokenizer.encode_plus(
                        tweet,
                        add_special_tokens = True,
                        max_length = max_len,
                        pad_to_max_length = True,
                        return_attention_mask = True,
                        return_tensors = 'pt',
                   )
    test_input_ids.append(encoded_dict['input_ids'])
    test_attention_masks.append(encoded_dict['attention_mask'])
test_input_ids = torch.cat(test_input_ids, dim=0)
test_attention_masks = torch.cat(test_attention_masks, dim=0)
```

```
test_dataset = TensorDataset(test_input_ids, test_attention_masks)
test_dataloader = DataLoader(
            test_dataset, # The validation samples.
            sampler = SequentialSampler(test_dataset), # Pull out batches sequentially.
            batch_size = batch_size # Evaluate with this batch size.
        )
```

```
predictions = []
for batch in test_dataloader:
        b_input_ids = batch[0].to(device)
        b_input_mask = batch[1].to(device)
        with torch.no_grad():
            output= model(b_input_ids,
                                token_type_ids=None,
                                attention_mask=b_input_mask)
            logits = output.logits
            logits = logits.detach().cpu().numpy()
            pred_flat = np.argmax(logits, axis=1).flatten()

            predictions.extend(list(pred_flat))
```

```
df_output = pd.DataFrame()
df_output['id'] = df_test['id']
df_output['target'] =predictions
df_output.to_csv('submission.csv',index=False)
```

```
df_output
```

	id	target
0	0	1
1	2	1
2	3	1
3	9	1
4	11	1
...	...	...
3258	10861	1

```
df_output['id']
```

```
0           0
1           2
2           3
3           9
4          11
         ...
3258    10861
3259    10865
3260    10868
3261    10874
3262    10875
Name: id, Length: 3263, dtype: int64
```

Submission

```
pd.read_csv('/content/submission.csv')
```

	id	target
0	0	1
1	2	1
2	3	1
3	9	1
4	11	1
...	...	...
3258	10861	1
3259	10865	1
3260	10868	1
3261	10874	1
3262	10875	1

3263 rows × 2 columns

```
df_output['target']
```

```
0       1
1       1
2       1
3       1
4       1
       ..
3258    1
3259    1
3260    1
3261    1
3262    1
Name: target, Length: 3263, dtype: int64
```