THE GEORGE WASHINGTON UNIVERSITY

WASHINGTON, DC

DATS-6312 NLP for Data Science

Fake and Real News Dataset

Prof. Amir Jafari

Final Term Project

Aasish Kumar Immadisetty

Table of Contents

Introduction:

News is a kind of communication that keeps us informed about current events, issues, and individuals all around the world, and staying informed about what's going on in the world is vital. We offer a variety of ways to stay informed about current events. It is critical for everyone to understand if the news they are reading is real or not.

As a result, we took this problem statement into account to work on our project to classify whether the news we read or hear is accurate or false.

Dataset Overview:

- There are two datasets:
    1. True.csv – which has correct news
    2. Fake.csv – which has bogus news
- The true articles were retrieved via crawling articles from Reuters.com which has been compiled from real-world sources. The ‘True.csv’ has more than 12,600 Reuters.com articles.
- The fake articles or information is gathered from various sources which some are untrustworthysuch as Politifact. The ‘Fake.csv’ has almost 12,600 articles.
- These both datasets have the articles mostly about politics and world events.

Data Pre-processing:

1. Removing Special Characters:

In this pre-processing step we remove special characters.

Original Text:
```
the Russia conspiracy with Mustafa Tameez, former
```

Function:
```python
def remove_characters(text):
    return re.sub('[^a-zA-Z]', ' ', text)
```

Cleaned Text:
```
the Russia conspiracy with Mustafa Tameez  former
```

2. Removing Stop words:

In this step we remove stop words from the text.

Original Text:

```
former consultant for the Department of Homeland Security
```

Function:

```
def remove_stopwords(text):
    return ' '.join([word for word in nltk.word_tokenize(text) if word not in stop_words])
```

Cleaned text:

```
former consultant Department Homeland Security
```

3. Stemming:

The process of reducing a word to its word stem, which affixes to suffixes and prefixes or to the roots of words known as a lemma, is known as stemming.

Original text:

```
Tucker Carlson debates the Russia conspiracy with Mustafa Tameez,
```

Function:

```
def stemming_words(text):
    return ' '.join(stemmer.stem(word) for word in text.split())
```

Cleaned text:

```
tucker carlson debat the russia conspiraci with mustafa tameez
```

4. Lemmatization:

Lemmatization is the process of combining a word's several inflected forms into a single item that can be studied. Lemmatization is like stemming, but it gives the words context. As a result, it connects words with similar meanings into a single term.

Original text:

```
Tucker goes off the rails after this political hack tries to pull
```

Function:

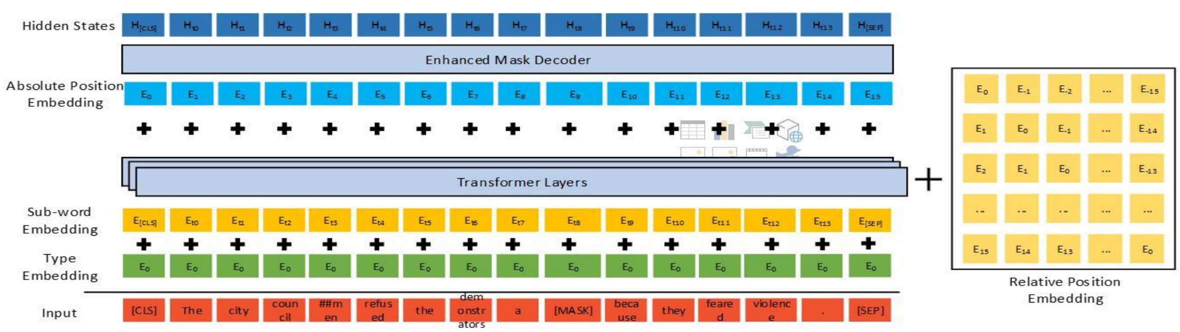
```
def lemmatize_words(text):
    return ' '.join(lemmatizer.lemmatize(word) for word in text.split())
```

Cleaned text:

```
tucker goe off the rail after thi polit hack tri to pull
```

Models:

DeBERTa Model:



- To improve the BERT and RoBERTa models, DeBERTa (Decoding-enhanced BERT with Disentangled Attention) employs two distinct techniques.
- The first is the disentangled attention mechanism, in which each word is represented by two vectors that transmit its content and position, and attention weights are generated between words using disentangled matrices on their contents and relative positions.
- Second, an updated mask decoder is used instead of the output SoftMax layer to anticipate the masked tokens for model pretraining. We demonstrate that these two solutions considerably increase model pre-training efficiency and downstream task performance.

Description for the portion of work:

- All tasks were performed on AWS cloud.
- All the tasks were performed with 3,4 epochs and with batch size 3,4.
- Number of trainable parameters: 139,193,858

```
#DEBERTA
checkpoint = "microsoft/deberta-base"
tokenizer = DebertaTokenizer.from_pretrained(checkpoint)

# Initializing the hyperparameters

NUM_LABELS = 2
BATCH_SIZE = 4
MAX_LEN = 256
EPOCHS = 3
LEARNING_RATE = 1e-5

# Generating id's and attenton mask from train and validation dataframe
train = tokenizer(list(train_df.text.values), truncation=True, padding=True, max_length=MAX_LEN)
train_input_ids = train['input_ids']
train_masks = train['attention_mask']

validation = tokenizer(list(validation_df.text.values), truncation=True, padding=True, max_length=MAX_LEN)
validation_input_ids = validation['input_ids']
validation_masks = validation['attention_mask']
```

```
# Train data TO TENSOR
train_inputs = torch.tensor(train_input_ids)
train_masks = torch.tensor(train_masks)
train_labels = torch.tensor(train_df.target.values)

# Validation data to TENSOR
validation_labels = torch.tensor(validation_df.target.values)
validation_inputs = torch.tensor(validation_input_ids)
validation_masks = torch.tensor(validation_masks)


# DataLoader for our training set.
train_data = TensorDataset(train_inputs, train_masks, train_labels)
train_sampler = RandomSampler(train_data)
train_dataloader = DataLoader(train_data, sampler=train_sampler, batch_size=BATCH_SIZE)

# DataLoader for our validation set.
validation_data = TensorDataset(validation_inputs, validation_masks, validation_labels)
validation_sampler = SequentialSampler(validation_data)
validation_dataloader = DataLoader(validation_data, sampler=validation_sampler, batch_size=BATCH_SIZE)
```

```
# Model Building
model = DebertaForSequenceClassification.from_pretrained(checkpoint, num_labels=NUM_LABELS, output_hidden_states=False, output_attentions=False)
model = model.to(device) # copying all tensor variables to GPU as specified by the device
```

```python
# Training Model
def train_fn(model, train_loader, optimizer, device, scheduler, criterion=None):
    ''' Define the training function '''
    model.train() #set the model on train mode
    total_loss, total_acc = 0, 0

    for batch in train_loader:
        input_ids = batch[0].to(device)
        input_mask = batch[1].to(device)
        labels = batch[2].to(device)
        optimizer.zero_grad() # Removing gradients from last batch
        outputs = model(input_ids, attention_mask=input_mask, labels=labels) # Retrieve Predictions
        loss = outputs.loss
        total_loss += loss.item() # Average of losses
        loss.backward() # Compute the gradients
        optimizer.step() # Updating parameters
        scheduler.step()
        logits = outputs.logits # Compute logits
        total_acc += eval_metric(logits, labels)

    loss_per_epoch = total_loss/len(train_loader) # Compute loss per epoch
    acc_per_epoch = total_acc/len(train_loader)
    return loss_per_epoch, acc_per_epoch
```

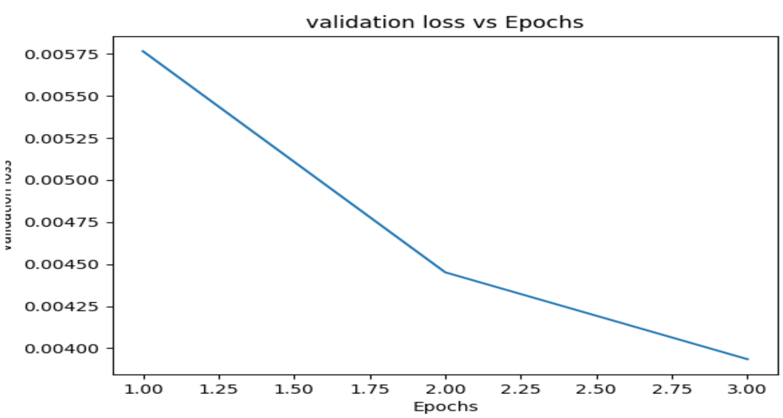
```python
# Evaluation function

def eval_fn(model, data_loader, device, criterion=None):
    ''' Define the evaluation function '''
    model.eval() # set the model on eval mode
    total_loss, total_acc = 0, 0

    with torch.no_grad():
        for batch in data_loader:
            input_ids = batch[0].to(device)
            input_mask = batch[1].to(device)
            labels = batch[2].to(device)
            outputs = model(input_ids, attention_mask=input_mask, labels=labels) # get predictions
            loss = outputs.loss
            total_loss += loss.item() # Average of losses
            logits = outputs.logits
            total_acc += eval_metric(logits, labels)

    loss_per_epoch = total_loss/len(data_loader)
    acc_per_epoch = total_acc/len(data_loader)
    return loss_per_epoch, acc_per_epoch
```

Results:

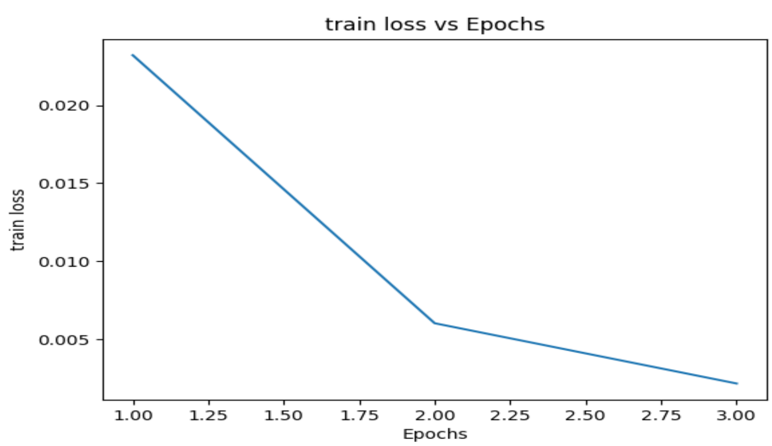
Model	Optimizer	Epoch	Batch SIze	Max_Len	Learning Rate	Train Accuracy	Validation Accuracy
DeBERTa	AdamW	4	3	30	0.001	52.13	52.29
DeBERTa	AdamW	3	4	256	1.00E-05	99.95	99.91

Model	Precision		recall		f1-score		Total F1 Score
	0's	1's	0's	1's	0's	1's	
DeBERTa	0.52	0	1	0	0.69	0	
DeBERTa	0.98	1	1	0.98	0.99	0.99	0.98845

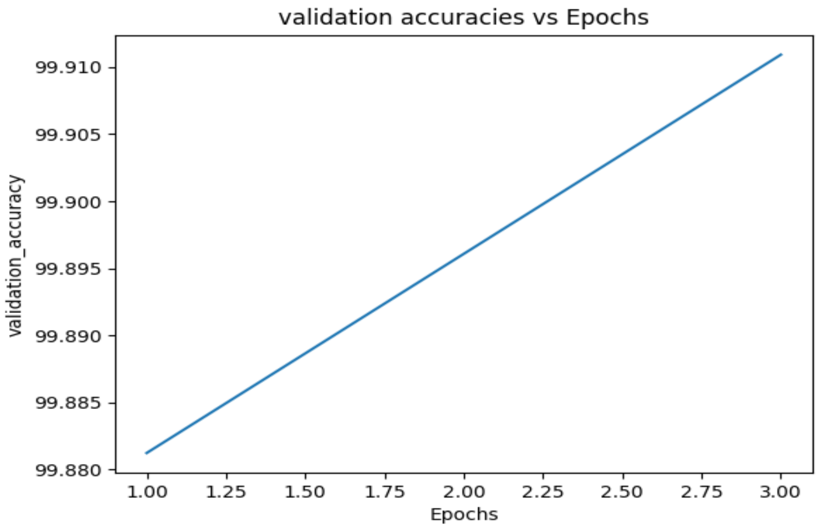
Validation loss vs Epoch:



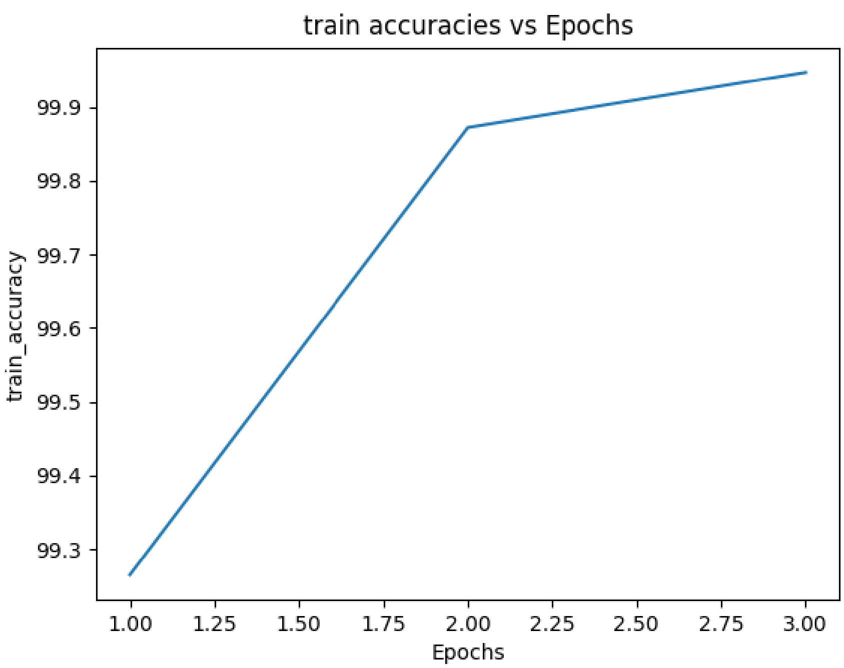
Train loss vs Epochs:



Validation Accuracy vs Epoch:



Train Accuracy vs Epoch:



Summary and Conclusion:

- To Summarize, DeBERTa model has an f1 score of "0.9884575026232949" with "139,193,858" trainable params.

Percentage of Code Written:

- The information about codes were referred from Hugging Face. Approximately 39% of the code was referred from the internet.

References:

- https://huggingface.co/docs/transformers/model_doc/deberta
- https://www.analyticsvidhya.com/blog/2021/07/detecting-fake-news-with-natural-language-processing/