Sentiment Analysis for Marketing

PHASE -4

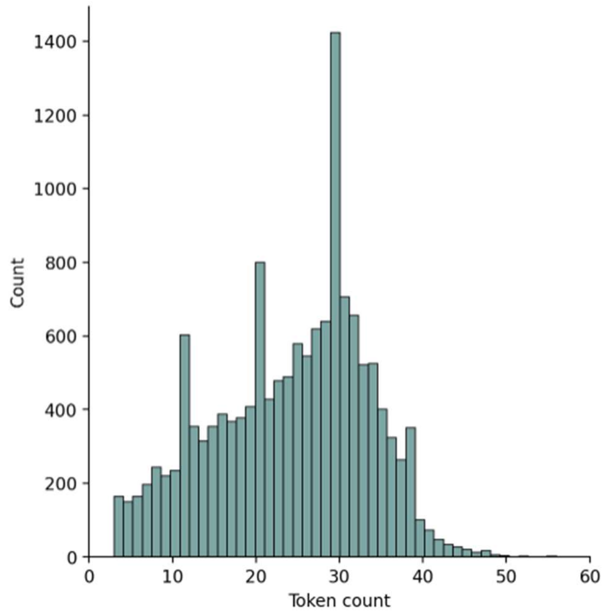
DATE	24 October 2023
TEAM ID	Proj-212173-Team-1
PROJECT NAME	Sentiment analysis for Marketing

```
[13] MODEL = 'bert-base-uncased'
     tokenizer = BertTokenizer.from_pretrained(MODEL, do_lower_case=True)

Downloading (…)okenizer_config.json: 100%  28.0/28.0 [00:00<00:00, 664B/s]
Downloading (…)solve/main/vocab.txt: 100%  232k/232k [00:00<00:00, 819kB/s]
Downloading (…)/main/tokenizer.json: 100%  466k/466k [00:00<00:00, 12.5MB/s]
Downloading (…)lve/main/config.json: 100%  570/570 [00:00<00:00, 19.6kB/s]
```

BERT base model (uncased) Pretrained model on English language using a masked language modeling (MLM) objective. It was introduced in this paper and first released in this repository. This model is uncased: it does not make a difference between english and English.

[14]
```
sns.displot(tokens)
plt.xlim([0, 60]);
plt.xlabel('Token count');
```



Token Count means the number of Tokens required to operate a single instance of a Software product in accordance with the provisions of this Agreement.

```python
from torch.utils.data import Dataset, DataLoader

# Define a custom dataset, more info on how to build custom dataset can be
# found at https://pytorch.org/tutorials/beginner/data_loading_tutorial.html
class CustomDataset(Dataset):

    def __init__(
        self,
        tweets,
        labels,
        tokenizer,
        max_length
    ):
        self.tweets = tweets
        self.labels = labels
        self.tokenizer = tokenizer
        self.max_length = max_length

    def __len__(self):
        return len(self.tweets)

    def __getitem__(self, idx):
        tweet = self.tweets[idx]
        label = self.labels[idx]
```

```python
tokenize = self.tokenizer.encode_plus(
    tweet,
    add_special_tokens=True,
    max_length=self.max_length,
    return_token_type_ids=False,
    padding='max_length',
    return_attention_mask=True,
    return_tensors='pt'
)
return {
    'tweet': tweet,
    'input_ids': tokenize['input_ids'].flatten(),
    'attention_mask': tokenize['attention_mask'].flatten(),
    'targets': torch.tensor(label, dtype=torch.long)}
```

In the above program return the values of tweet,input_ids,attention_mask and target to the program.It is the main part of import the dataset and create the class then define the functions and finally produce the output.

```python
MAX_LENGTH = 64
TEST_SIZE = 0.1
VALID_SIZE = 0.5
BATCH_SIZE = 16
NUM_WORKERS = 2

train_sampler, test_sampler = train_test_split(df, test_size=TEST_SIZE, random_state=RANDOM_STATE)
valid_sampler, test_sampler = train_test_split(test_sampler, test_size=VALID_SIZE, random_state=RANDOM_STATE)

train_set = CustomDataset(
    train_sampler['text'].to_numpy(),
    train_sampler['labels'].to_numpy(),
    tokenizer,
    MAX_LENGTH
)
test_set = CustomDataset(
    test_sampler['text'].to_numpy(),
    test_sampler['labels'].to_numpy(),
    tokenizer,
    MAX_LENGTH
)
valid_set = CustomDataset(
    valid_sampler['text'].to_numpy(),
    valid_sampler['labels'].to_numpy(),
    tokenizer,
    MAX_LENGTH
)
```

```python
train_loader = DataLoader(train_set, batch_size=BATCH_SIZE, num_workers=NUM_WORKERS)
test_loader = DataLoader(test_set, batch_size=BATCH_SIZE, num_workers=NUM_WORKERS)
valid_loader = DataLoader(valid_set, batch_size=BATCH_SIZE, num_workers=NUM_WORKERS)
```

The motivation is quite simple: you should separate your data into train, validation, and test splits to prevent your model from overfitting and to accurately evaluate your model. Accurate training data helps the model learn the right patterns, validation data helps developers fine-tune the model correctly, and test data provides trustworthy metrics so they can confidently deploy their AI solution.

```python
from torch import nn
class AirlineSentimentClassifier(nn.Module):

    def __init__(self, num_labels):
        super (AirlineSentimentClassifier, self).__init__()
        self.bert = BertModel.from_pretrained(MODEL)
        self.dropout = nn.Dropout(p=0.2)
        self.classifier = nn.Linear(self.bert.config.hidden_size, num_labels)

    def forward(self, input_ids, attention_mask):
        outputs = self.bert(
            input_ids=input_ids,
            attention_mask=attention_mask
        )
        pooled_output = outputs[1]
        pooled_output = self.dropout(pooled_output)
        out = self.classifier(pooled_output)
        return out
```

[19]

```python
model = AirlineSentimentClassifier(len(labels_map))
print(model)

# Move tensors to GPU on CUDA enables devices
if device:
    model.cuda()
```

Downloading model.safetensors: 100% 440M/440M [00:04<00:00, 33.2MB/s]

```
AirlineSentimentClassifier(
  (bert): BertModel(
    (embeddings): BertEmbeddings(
      (word_embeddings): Embedding(30522, 768, padding_idx=0)
      (position_embeddings): Embedding(512, 768)
      (token_type_embeddings): Embedding(2, 768)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (encoder): BertEncoder(
      (layer): ModuleList(
        (0-11): 12 x BertLayer(
          (attention): BertAttention(
            (self): BertSelfAttention(
              (query): Linear(in_features=768, out_features=768, bias=True)
              (key): Linear(in_features=768, out_features=768, bias=True)
              (value): Linear(in_features=768, out_features=768, bias=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
            (output): BertSelfOutput(
              (dense): Linear(in_features=768, out_features=768, bias=True)
              (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
```

```
            (intermediate): BertIntermediate(
              (dense): Linear(in_features=768, out_features=3072, bias=True)
              (intermediate_act_fn): GELUActivation()
            )
            (output): BertOutput(
              (dense): Linear(in_features=3072, out_features=768, bias=True)
              (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
        )
      )
      (pooler): BertPooler(
        (dense): Linear(in_features=768, out_features=768, bias=True)
        (activation): Tanh()
      )
    )
    (dropout): Dropout(p=0.2, inplace=False)
    (classifier): Linear(in_features=768, out_features=3, bias=True)
  )
```

CUDA® is a parallel computing platform and programming model invented by NVIDIA. It enables dramatic increases in computing performance by harnessing the power of the graphics processing unit (GPU).

```
n_epochs = 10
learning_rate =2e-5

# Loss function
criterion = nn.CrossEntropyLoss()

# Optimizer
optimizer = AdamW(model.parameters(), lr=learning_rate, correct_bias=False)

# Define scheduler
training_steps = len(train_loader)*n_epochs
scheduler = get_linear_schedule_with_warmup(
    optimizer,
    num_warmup_steps=0,
    num_training_steps=training_steps
)
```

```
/usr/local/lib/python3.10/dist-packages/transformers/optimization.py:411: FutureWarning: This implementation of AdamW is de
  warnings.warn(
```

In machine learning, a loss function and an optimizer are two essential components that help to improve the performance of a model. A loss function measures the difference between the predicted output of a model and the actual output, while an optimizer adjusts the model's parameters to minimize the loss function.

```
# Track changes in validation loss
valid_loss_min = np.Inf

for epoch in range(1, n_epochs+1):

    # Setting training and validation loss
    train_loss = []
    validation_loss = []
    tr_predictions = 0
    acc = 0
    val_predictions = 0

    ####################
    #  Train the model #
    ####################
    model = model.train()
    for data in train_loader:

        # Moving tensors to GPU on CUDA enabled devices
        if device:
            input_ids, attention_mask, targets = data["input_ids"].cuda(), data["attention_mask"].cuda(), data["targets
        # Clear the gradients of variables
        optimizer.zero_grad()
        #### Forward pass
        # Pass input through the model
        output = model(
            input_ids=input_ids,
            attention_mask=attention_mask
        )
        # Compute batch loss
        loss = criterion(output, targets)
        # Convert output probabilities to class probabilities
        _, pred = torch.max(output, 1)
        # Track correct predictions
        tr_predictions += torch.sum(pred == targets)

        #### Backward Pass
        # Compute gradients wrt to model parameters
        loss.backward()
        # To avoid exploding gradients, we clip the gradients of the model
        nn.utils.clip_grad_norm_(model.parameters(), max_norm=1.0)
        # Perform parameter update
        optimizer.step()
        # Update learning rate
        scheduler.step()
        # Update loss per mini batches
        train_loss.append(loss.item())

    ######################
    # Validate the model #
    ######################
    model.eval()
    with torch.no_grad():
        for data in valid_loader:
```

```python
        # Moving tensors to GPU on CUDA enabled devices
        if device:
            input_ids, attention_mask, targets = data["input_ids"].cuda(), data["attention_mask"].cuda(), data["tar

        #### Forward pass
        # Pass input through the model
        output = model(
            input_ids=input_ids,
            attention_mask=attention_mask
        )
        # Compute batch loss
        loss = criterion(output, targets)
        # Convert output probabilities to class probabilities
        _, pred = torch.max(output, 1)
        # Update loss per mini batches
        validation_loss.append(loss.item())
        # Track correct predictions
        val_predictions += torch.sum(pred == targets)
```

```python
    # Compute accuracy
    train_accuracy = tr_predictions.double()/len(train_sampler)
    val_accuracy = val_predictions.double()/len(valid_sampler)

    # Print loss statistics
    print('Epoch: {}/{} \n\tTraining Loss: {:.6f} \n\tValidation Loss: {:.6f} \n\tTrain Accuracy: {:.6f} \n\tVal Accura

    # Save model if validation loss is decreased
    if val_accuracy > acc:
        print('Saving model...')
        torch.save(model.state_dict(), 'bert_base_fine_tuned.pt')
        acc = val_accuracy
```

One of the most widely used metrics combinations is training loss + validation loss over time. The training loss indicates how well the model is fitting the training data, while the validation loss indicates how well the model fits new data.

At times, the validation loss is greater than the training loss. This may indicate that the model is underfitting. Underfitting occurs when the model is unable to accurately model the training data, and hence generates large errors.

Typically validation loss should be similar to but slightly higher than training loss. As long as validation loss is lower than or even equal to training loss one should keep doing more training.

```
Epoch: 1/10
        Training Loss: 0.478485
        Validation Loss: 0.426510
        Train Accuracy: 0.813221
        Val Accuracy: 0.848361
Saving model...
Epoch: 2/10
        Training Loss: 0.251598
        Validation Loss: 0.587404
        Train Accuracy: 0.912720
        Val Accuracy: 0.837432
Saving model...
Epoch: 3/10
        Training Loss: 0.147462
        Validation Loss: 0.694001
        Train Accuracy: 0.958333
        Val Accuracy: 0.848361
Saving model...
Epoch: 4/10
        Training Loss: 0.095958
        Validation Loss: 0.852052
        Train Accuracy: 0.976548
        Val Accuracy: 0.841530
Saving model...
Epoch: 5/10
        Training Loss: 0.062927
        Validation Loss: 0.967488
        Train Accuracy: 0.985504
        Val Accuracy: 0.842896
Saving model...
Epoch: 6/10
        Training Loss: 0.042360
        Validation Loss: 1.066000
        Train Accuracy: 0.990437
        Val Accuracy: 0.840164
Saving model...
Epoch: 7/10
        Training Loss: 0.032142
        Validation Loss: 1.132496
        Train Accuracy: 0.992410
        Val Accuracy: 0.833333
Saving model...
Epoch: 8/10
        Training Loss: 0.024429
        Validation Loss: 1.184951
        Train Accuracy: 0.993777
        Val Accuracy: 0.829235
Saving model...
Epoch: 9/10
        Training Loss: 0.018996
        Validation Loss: 1.230268
        Train Accuracy: 0.994991
        Val Accuracy: 0.831967
Saving model...
Epoch: 10/10
        Training Loss: 0.015075
        Validation Loss: 1.244014
        Train Accuracy: 0.995826
        Val Accuracy: 0.830601
Saving model...
```

when you see that the model performs well on the training data but does not perform well on the evaluation data. This is because the model is memorizing the data it has seen and is unable to generalize to unseen examples.

```python
# Track test loss
test_loss = 0.0
class_predictions = list(0. for i in range(3))
class_total = list(0. for i in range(3))
predictions = []
labels = []

model.eval()
with torch.no_grad():
    for data in test_loader:

        # Moving tensors to GPU on CUDA enabled devices
        if device:
            input_ids, attention_mask, targets = data["input_ids"].cuda(), data["attention_mask"].cuda(), data["targets

        #### Forward pass
        # Pass input through the model
        output = model(
            input_ids=input_ids,
            attention_mask=attention_mask
        )
        # Compute batch loss
        loss = criterion(output, targets)
        # Update loss
        test_loss += loss.item()
        # convert output probabilities to predicted class
        _, pred = torch.max(output, 1)

        predictions.extend(pred)
        labels.extend(targets)

predictions = torch.stack(predictions) if not device else torch.stack(predictions).cpu()
labels = torch.stack(labels) if not device else torch.stack(labels).cpu()
```

```python
print(classification_report(predictions, labels, target_names=['neutral', 'positive','negative']))
```

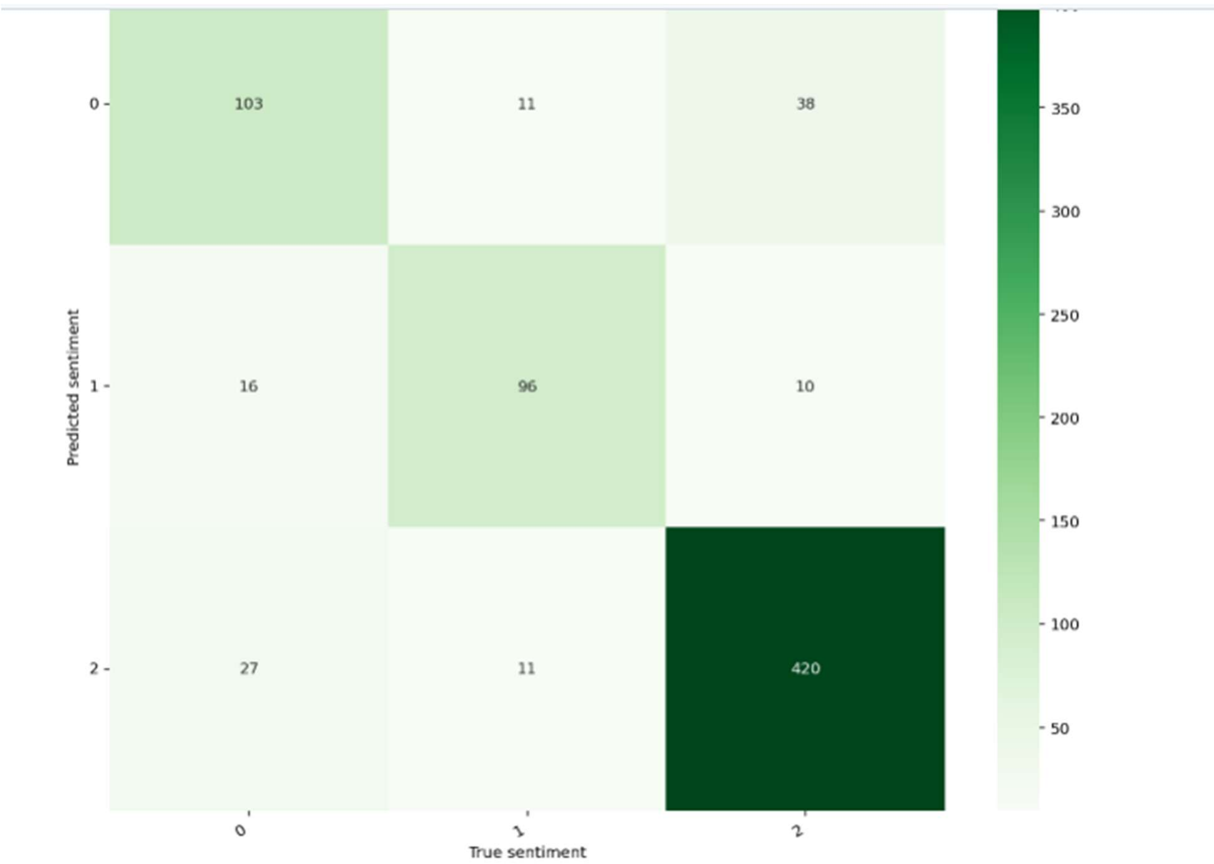
```
              precision    recall  f1-score   support

     neutral       0.68      0.71      0.69       146
    positive       0.79      0.81      0.80       118
    negative       0.92      0.90      0.91       468

    accuracy                           0.85       732
   macro avg       0.79      0.81      0.80       732
weighted avg       0.85      0.85      0.85       732
```

```python
cm = confusion_matrix(labels, predictions)
heatmap = sns.heatmap(cm, annot=True, fmt='d', cmap='Greens')
heatmap.yaxis.set_ticklabels(heatmap.yaxis.get_ticklabels(), rotation=0, ha='right')
heatmap.xaxis.set_ticklabels(heatmap.xaxis.get_ticklabels(), rotation=30, ha='right')
plt.xlabel('True sentiment')
plt.ylabel('Predicted sentiment');
```

He phrase 'the true sentiment' is correct and usable in written English. You can use it to refer to an honest or heartfelt emotion that someone is feeling. For example, "Let us honor the true sentiment behind the movement.".



1. Knowledge Based: This approach includes the classification of text based on words that are associated with emotion.
2. Statistical: This approach utilizes machine learning algorithms such as latent semantic analysis and deep learning for accurate detection of sentiment.