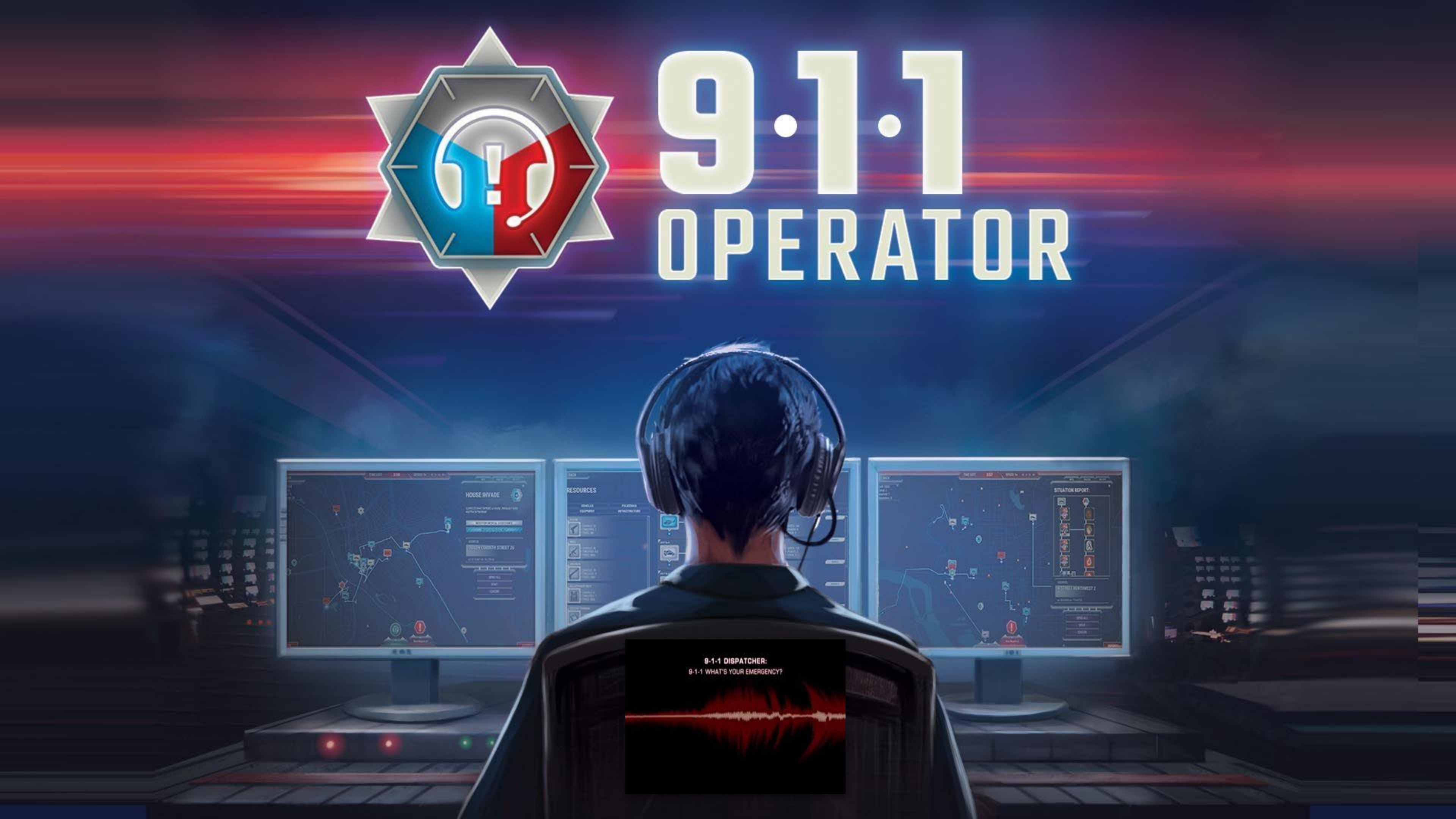
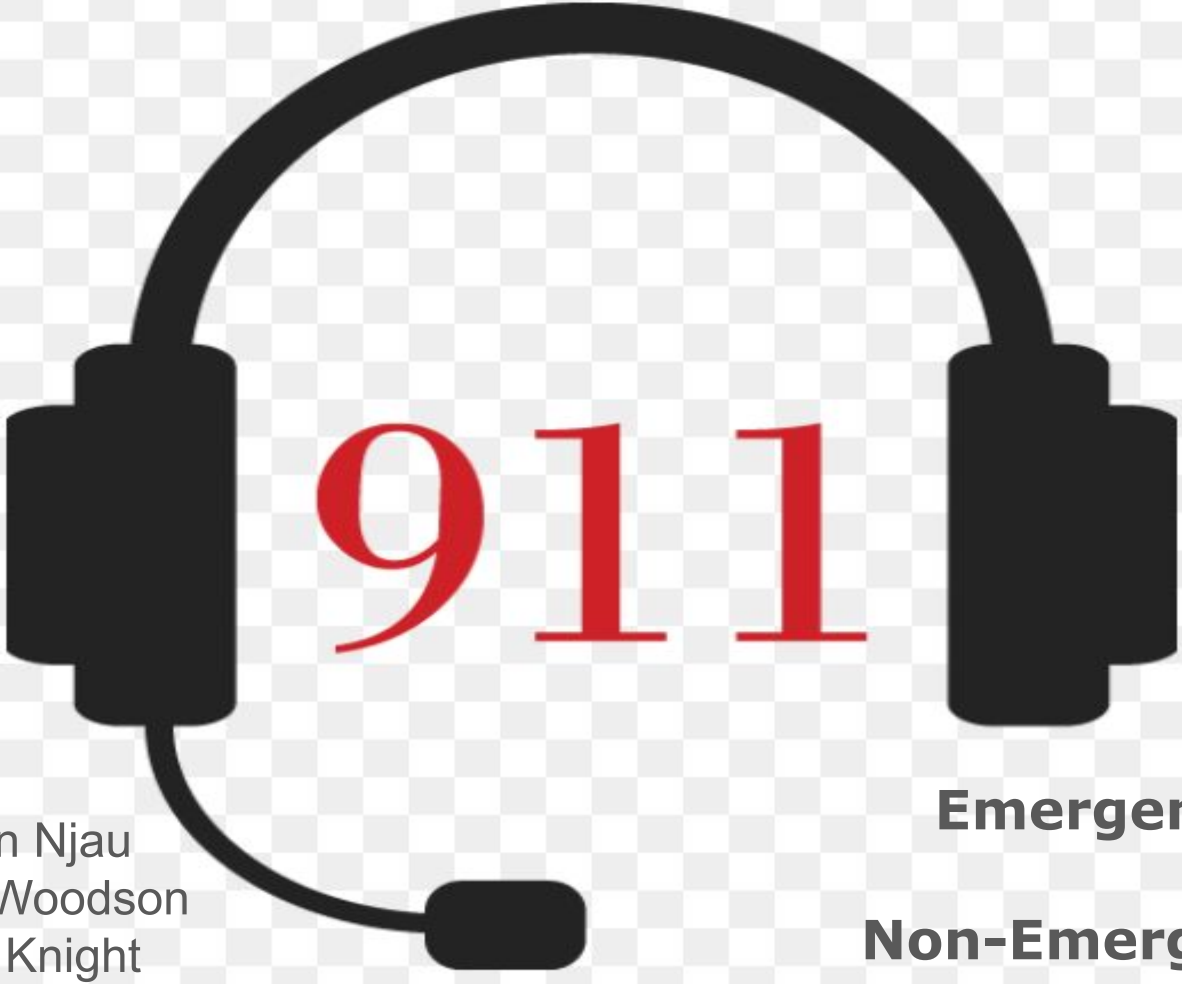

9·1·1
OPERATOR
HOUSE INVADE
RESOURCES
SITUATION REPORT:
9-1-1 DISPATCHER:
9-1-1 WHAT'S YOUR EMERGENCY?



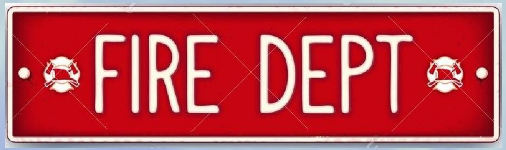
Team 4 - Nickson Njau
Symia Woodson
Wanda Knight

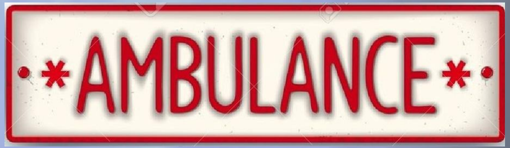
Emergency

Non-Emergency

Distinguishing Emergency Calls vs Non-Emergency Calls

- ★ Implementing a Machine Learning Model to classify 911 calls
- ★ 911 call transcripts to automate the categorization of emergency call conversations
- ★ Use data for training 911 operators





Overview of the Data Collection

★ Researched several website sources:
  - Devpost
  - Hugging Face
  - Kaggle

[Hugging Face]

Importing the Data

```
[3]: from datasets import load_dataset

     ds = load_dataset("spikecodes/911-call-transcripts")
```

```
Loading widget...
Loading widget...
Loading widget...
```

Imported the data from Kaggle

Data Cleanup

```
df
```

[6]:

	messages
0	[{'role': 'assistant', 'content': '9-1-1, what...
1	[{'role': 'assistant', 'content': '9-1-1, what...
2	[{'role': 'assistant', 'content': '9-1-1, what...
3	[{'role': 'assistant', 'content': '9-1-1, what...
4	[{'role': 'assistant', 'content': '9-1-1, what...
...	...
513	[{'role': 'assistant', 'content': '9-1-1, what...
514	[{'role': 'assistant', 'content': '9-1-1, what...
515	[{'role': 'assistant', 'content': '9-1-1, what...
516	[{'role': 'assistant', 'content': '9-1-1, what...
517	[{'role': 'assistant', 'content': '9-1-1, what...

518 rows × 1 columns

Results after importing the data

Data Cleanup

Steps taken:

1. Split the messages column into:
    a. Messages
    b. Text
    c. Original Index
    d. Role
    e. Label - 0 for emergency, 1 for non-emergency

Data Cleanup

Classify Into Two:(0 for emergency and 1 for non emergency)

```
# Dictionary of emergency calls
non_emergency_call_numbers = [8,32,33,40,44,50,53,64,78,81,88,121,124,133,181,189,192,199,205,219,240,246,247,258,259,261,26

# Add a column called label
df['label'] = 0

# if index matches number in non_emergency_calls, convert the df['label'] to 1
for index, row in df.iterrows():
    if index in non_emergency_call_numbers:
        df.at[index, 'label'] = 1
```

Almost there!

Data Cleanup

```
# First look at what we're removing
print("Entries being removed:")
print(expanded_df[expanded_df.index.get_level_values('original_index').isin([78,

# Then remove them
expanded_df = expanded_df[~expanded_df.index.get_level_values('original_index').i
```

```
Entries being removed:
                    role                                        content  \
original_index
78             assistant                  9-1-1, what's your emergency?
78                  user                             1620 Green Place.
78             assistant                                    1620 what?
78                  user                                  Green Place.
78             assistant                Green Place. Green like the color?
...                  ...                                           ...
451                 user                              It's the police.
451            assistant  Okay. Step outside and do what they say. Just ...
451                 user                                   They're here.
451            assistant  Okay. Just put the phone down and do what they...
451                 user                                          Okay.

               label
```

More Data Cleanup:

1. Dropped Null values
2. Popped the 78th and 451st Conversations

Data Cleanup

```
expanded_df
```

[21]:

	role	content	label
original_index			
0	assistant	9-1-1, what's your emergency?	1
0	user	I'm at West High School. There's a guy with a ...	0
0	assistant	Which high school?	1
0	user	West High.	0
0	assistant	Okay, we have the police dispatched. Can you g...	1
...	...	...	...
517	assistant	Are you on a cordless phone?	1
517	user	I have a cordless phone, but I use a walker.	0
517	assistant	You can go ahead and hang up with me and go ah...	1
517	user	All right. Bye.	0
517	assistant	Bye.	1

25799 rows × 3 columns

Final Product!

Data Exploration Processes

+ Code + Markdown

```
train_texts, val_texts, train_labels, val_labels = train_test_split(expanded_df['
```

+ Code + Markdown

Split the data into Training and Testing

Data Exploration Processes

- Defined a custom PyTorch Dataset class called “EmergencyCallDataset”
- Trained the model

Code snippets shown in the following slides!

Data Exploration Processes

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

Constructor (`__init__` method):

Initialized the dataset with texts, labels, a tokenizer, and a maximum length.

The tokenizer - a pre-trained tokenizer (from BERT)

`max_length` sets the maximum number of tokens for each text.

Data Exploration Processes

```python
def __len__(self):
    return len(self.texts)

def __getitem__(self, idx):
    text = self.texts[idx]
    label = self.labels[idx]

    encoding = self.tokenizer.encode_plus(
        text,
        add_special_tokens=True,
        max_length=self.max_length,
        return_token_type_ids=False,
        padding='max_length',
        truncation=True,
        return_attention_mask=True,
        return_tensors='pt',
    )

    return {
        'input_ids': encoding['input_ids'].flatten(),
        'attention_mask': encoding['attention_mask'].flatten(),
        'labels': torch.tensor(label, dtype=torch.long)
    }
```

`__len__` method:

Returns the total number of items in the dataset.

This is used by PyTorch's Data Loader to know how many

items are available.

`__getitem__` method:

- This is the core of the dataset class. It defines how to

retrieve and process a single item:

Retrieves the text and label for a given index.

Uses the tokenizer to encode the text:

Returns a dictionary containing:

`input_ids`: The tokenized and encoded text.

`attention_mask`: A mask indicating which tokens are padding (0) and which are actual content (1).

`labels`: The label converted to a PyTorch tensor.

Data Exploration Processes

```
from transformers import AdamW, get_linear_schedule_with_warmup

# Set up tokenizer and model
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
model = BertForSequenceClassification.from_pretrained('bert-base-uncased', num_la
```

- Importing optimizer and learning rate scheduler
- Setting up the tokenizer
- Setting up the model

Data Exploration Processes

27]:

```
# Create datasets
train_dataset = EmergencyCallDataset(train_texts, train_labels, tokenizer, max_le
val_dataset = EmergencyCallDataset(val_texts, val_labels, tokenizer, max_length=5
```

- Created a training and validation dataset
- For each dataset, passed in:
    - the texts, the corresponding labels for each text, the BERT tokenizer initialized earlier, the maximum length of each input sequence as 512 tokens

Data Exploration Processes

```
# Create data loaders
train_loader = DataLoader(train_dataset, batch_size=8, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=8, shuffle=False)
```

- Created data loaders for training and validation that did the following:
    - Batching - grouped data into batches of 8 samples, crucial for efficient training and utilization of GPU memory
    - Shuffling(for training) - prevent overfitting and ensures the model sees different sample orders across epochs
    - Iteration
    - Memory efficiency

Data Exploration Processes

```
# Set up optimizer and scheduler
optimizer = AdamW(model.parameters(), lr=2e-5, correct_bias=False)
total_steps = len(train_loader) * 10  # 10 epochs
scheduler = get_linear_schedule_with_warmup(optimizer, num_warmup_steps=0, num_tr
```

- Setting up the optimizer
- Calculating total steps
- Setting up the learning rate scheduler

Data Exploration Processes

```python
# Set up device (GPU if available, otherwise CPU)
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model.to(device)

# Training loop
for epoch in range(10):
    # Training phase
    model.train()
    for batch in train_loader:
        # Clear previous gradients
        optimizer.zero_grad()

        # Move data to device
        inputs = {
            'input_ids': batch['input_ids'].to(device),
            'attention_mask': batch['attention_mask'].to(device),
            'labels': batch['labels'].to(device)
        }

        # Forward pass and calculate loss
        outputs = model(**inputs)
        loss = outputs.loss

        # Backward pass and optimization
        loss.backward()
        optimizer.step()
        scheduler.step()

    # Validation phase
    model.eval()  # Set model to evaluation mode
    val_loss = 0
    correct = 0
    total = 0

    # No gradient calculation needed for validation
    with torch.no_grad():
        for batch in val_loader:
            # Move data to device
            inputs = {
                'input_ids': batch['input_ids'].to(device),
                'attention_mask': batch['attention_mask'].to(device),
                'labels': batch['labels'].to(device)
            }

            # Get model predictions
            outputs = model(**inputs)

            # Calculate validation loss
            val_loss += outputs.loss.item()

            # Calculate accuracy
            _, predictions = torch.max(outputs.logits, 1)
            total += inputs['labels'].size(0)
            correct += (predictions == inputs['labels']).sum().item()

    # Print epoch results
    avg_val_loss = val_loss / len(val_loader)
    accuracy = (correct / total) * 100
    print(f'Epoch {epoch+1}:')
    print(f'  Validation Loss: {avg_val_loss:.4f}')
    print(f'  Accuracy: {accuracy:.2f}%')
    print('-' * 50)
```

Nick

Device Setup:

- Checked if a GPU is available; if so, use it; otherwise, default to the CPU
- Moved the model to the selected device for efficient computation

Training Loop:

- Iterated over 10 epochs

Training Phase:

- Set the model to training mode to enable features like dropout
- Looped through batches of training data
- Clear previous gradients to prevent accumulation

SEE NEXT SLIDE!

Data Exploration Processes

```python
# Set up device (GPU if available, otherwise CPU)
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model.to(device)

# Training loop
for epoch in range(10):
    # Training phase
    model.train()
    for batch in train_loader:
        # Clear previous gradients
        optimizer.zero_grad()

        # Move data to device
        inputs = {
            'input_ids': batch['input_ids'].to(device),
            'attention_mask': batch['attention_mask'].to(device),
            'labels': batch['labels'].to(device)
        }

        # Forward pass and calculate loss
        outputs = model(**inputs)
        loss = outputs.loss

        # Backward pass and optimization
        loss.backward()
        optimizer.step()
        scheduler.step()

    # Validation phase
    model.eval()  # Set model to evaluation mode
    val_loss = 0
    correct = 0
    total = 0

    # No gradient calculation needed for validation
    with torch.no_grad():
        for batch in val_loader:
            # Move data to device
            inputs = {
                'input_ids': batch['input_ids'].to(device),
                'attention_mask': batch['attention_mask'].to(device),
                'labels': batch['labels'].to(device)
            }

            # Get model predictions
            outputs = model(**inputs)

            # Calculate validation loss
            val_loss += outputs.loss.item()

            # Calculate accuracy
            _, predictions = torch.max(outputs.logits, 1)
            total += inputs['labels'].size(0)
            correct += (predictions == inputs['labels']).sum().item()

    # Print epoch results
    avg_val_loss = val_loss / len(val_loader)
    accuracy = (correct / total) * 100
    print(f'Epoch {epoch+1}:')
    print(f'  Validation Loss: {avg_val_loss:.4f}')
    print(f'  Accuracy: {accuracy:.2f}%')
    print('-' * 50)
```

Validation Phase:

- Set the model to evaluation mode to disable dropout and other training-specific behaviors.
- Used a context manager to disable gradient calculations for efficiency during validation.
- Looped through batches of validation data. Move input data to the appropriate device.

Results Reporting:

- After each epoch, calculated and printed:
    - The average validation loss by dividing total loss by the number of batches.
    - The accuracy by comparing correct predictions to total samples, expressed as a percentage

Approach taken to achieve Project Goals

Why PyTorch?

1. Easy to use and seamless integration with Python libraries.
2. Strong Community and Documentation
3. Excellent support for GPU computation
4. To learn a new technology
5. It’s more MATH😂

Approach taken to achieve Project Goals

Pivoting from Whisper

- Lengthy calls
- Distinguishing between caller and operator
- Couldn’t pick up people’s accents
- People with heightened emotions don’t speak clearly

Results of the analysis

```
Epoch 1:
  Validation Loss: 0.2095
  Accuracy: 93.06%
--------------------------------------------------
Epoch 2:
  Validation Loss: 0.2063
  Accuracy: 92.75%
--------------------------------------------------
Epoch 3:
  Validation Loss: 0.2279
  Accuracy: 93.14%
--------------------------------------------------
Epoch 4:
  Validation Loss: 0.2560
  Accuracy: 92.83%
--------------------------------------------------
Epoch 5:
  Validation Loss: 0.2650
  Accuracy: 92.79%
--------------------------------------------------
Epoch 6:
  Validation Loss: 0.2941
  Accuracy: 93.02%
--------------------------------------------------
Epoch 7:
  Validation Loss: 0.3317
  Accuracy: 92.77%
--------------------------------------------------
Epoch 8:
  Validation Loss: 0.3426
  Accuracy: 92.95%
--------------------------------------------------
Epoch 9:
  Validation Loss: 0.3848
  Accuracy: 92.79%
--------------------------------------------------
Epoch 10:
  Validation Loss: 0.3996
  Accuracy: 92.89%
--------------------------------------------------
```

At first, achieved the best accuracy of 93.14%

Saved the fine tuned model

Hyperparameter Tuning

```
Training with lr=1e-05, dropout=0.1, batch_size=16
Some weights of BertForSequenceClassification were not initialized from the model checkpoint at bert-base-uncased and are newly initialized: ['classifier.bias', 'classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
Epoch 1, Loss: 0.2439514753878001, Val Accuracy: 0.9310077519379845
Epoch 2, Loss: 0.17127876650152166, Val Accuracy: 0.9310077519379845
Epoch 3, Loss: 0.12663770005713368, Val Accuracy: 0.9292635658914729
Training with lr=1e-05, dropout=0.1, batch_size=32
Some weights of BertForSequenceClassification were not initialized from the model checkpoint at bert-base-uncased and are newly initialized: ['classifier.bias', 'classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
Epoch 1, Loss: 0.26065972010179084, Val Accuracy: 0.9242248062015503
Epoch 2, Loss: 0.18256274174985498, Val Accuracy: 0.931782945736434
Epoch 3, Loss: 0.14309063632358876, Val Accuracy: 0.9333333333333333
Training with lr=1e-05, dropout=0.3, batch_size=16
Some weights of BertForSequenceClassification were not initialized from the model checkpoint at bert-base-uncased and are newly initialized: ['classifier.bias', 'classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
Epoch 1, Loss: 0.2781811633833157, Val Accuracy: 0.9263565891472868
Epoch 2, Loss: 0.21381505455301944, Val Accuracy: 0.9282945736434108
Epoch 3, Loss: 0.18315540627511434, Val Accuracy: 0.9302325581395349
Training with lr=1e-05, dropout=0.3, batch_size=32
Some weights of BertForSequenceClassification were not initialized from the model checkpoint at bert-base-uncased and are newly initialized: ['classifier.bias', 'classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
Epoch 1, Loss: 0.29024026490921195, Val Accuracy: 0.9261627906976744
Epoch 2, Loss: 0.21607518527627914, Val Accuracy: 0.9271317829457364
Epoch 3, Loss: 0.19356134149801824, Val Accuracy: 0.9325581395348838
Training with lr=1e-05, dropout=0.5, batch_size=16
Some weights of BertForSequenceClassification were not initialized from the model checkpoint at bert-base-uncased and are newly initialized: ['classifier.bias', 'classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
Epoch 1, Loss: 0.36499219374196934, Val Accuracy: 0.914922480620155
Epoch 2, Loss: 0.27158724481398744, Val Accuracy: 0.921124031007752
Epoch 3, Loss: 0.25338554060332075, Val Accuracy: 0.9215116279069767
Training with lr=1e-05, dropout=0.5, batch_size=32
Some weights of BertForSequenceClassification were not initialized from the model checkpoint at bert-base-uncased and are newly initialized: ['classifier.bias', 'classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
Epoch 1, Loss: 0.3855125158332115, Val Accuracy: 0.9067829457364341
Epoch 2, Loss: 0.2810584941979989, Val Accuracy: 0.9207364341085271
Epoch 3, Loss: 0.2600413029334804, Val Accuracy: 0.9251937984496124
Training with lr=5e-05, dropout=0.1, batch_size=16
Some weights of BertForSequenceClassification were not initialized from the model checkpoint at bert-base-uncased and are newly initialized: ['classifier.bias', 'classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
Epoch 1, Loss: 0.2398287250061369, Val Accuracy: 0.9236434108527132
Epoch 2, Loss: 0.17666642335640598, Val Accuracy: 0.9298449612403101
```

Hyperparameter Tuning

- Decided to tune the:
  - Learning Rates
  - Batch sizes
  - Dropout rates

- Ultimately achieved a final best accuracy of 93.3333% with the following hyperparameters:
  - Learning Rate of 1e-05
  - Batch size of 32
  - Dropout rates of 0.1

Additional questions that surfaced, what Team 4 might research next if time was available or a plan for future development

If more time was available, we would want to create a model that classifies Emergency calls:

- ★ Into:
  - ○ Levels 1, 2, 3
  - ○ Fire
  - ○ Medical Emergency
  - ○ Police
  - ★ Parallel processors

References

https://wallpaperaccess.com/full/1551152.jpg

https://www.youtube.com/watch?v=li_XIBevDhs

https://flyclipart.com/emergency-dispatch-clip-art-emergency-preparedness-clipart-733297

https://clipground.com/images/emergency-services-clipart-free-19.jpg

https://devpost.com

https://www.kaggle.com

https://huggingface.co

https://medium.com/@joe_19081/ai-will-not-save-you-fb501a990d2d

https://www.criticalltestprep.com/artificial-intelligence-machine-learning-in-911/


Future of 911
Dispatching with AI
and Machine
Learning