MACHINE LEARNING PROJECT

AMBROSE YIP - 100861372
SURYANSH SAROCH - 100869700
SAMI KHAN 100830101
ARAD AYNTABLI - 100845722
DANIEL FOLINO - 100874060

12.13.2024

TABLE OF CONTENTS

THE PROBLEM

Issue: It can be very difficult to read emotion and intention through text. This leads to frequent miscommunication and frustration on social media and messages software.

Goal: Create a text classification model that predicts emotions (e.g. sadness, joy, love, anger, fear, surprise) from textual data

Input: Text samples (text messages, sentences)

Output: Discrete emotion label for each sample

DATASET COLLECTION

Data Source: https://www.kaggle.com/datasets/nelgiriyewithana/emotions/data

- Named `text.csv`, containing textual samples and corresponding emotional labels

Labeling: Each sample is assigned to a label, indicating one of **6** possible emotions

0. Sadness
1. Joy
2. Love
3. Anger
4. Fear
5. Surprise

Output: Discrete emotion label for each sample

HOW DOES IT WORK?

We preprocess the data, making sure we remove anything unnecessary like numbers or punctuation and we split the data into training and testing sets.

Then we build a vocabulary of the 1000 most common words, giving each word a unique index.

Our model uses LSTM (Long Short-Term Memory) which is an RNN (Recurrent Neural Network). RNN remembers words in a sentence like a sequence.

The model is trained using cross-entropy loss to optimize its ability to detect emotion.

We can evaluate the models performance by calculating the precision, accuracy, recall and F-1 score.

NEURAL NETWORK DESIGN

Model Architecture:

- **Embedding Layer:** First map words to dense vectors
- **LSTM Layer:** Captures sequential patterns in dataset
- **Fully Connected Layers:** Dimensionality reduction with ReLU
- **Dropout & Batch Normalization:** Prevent overfitting
- **Output Layer:** Predicts emotion class (0-5)

```python
# Model
class EmotionClassifier(nn.Module):
    def __init__(self, num_layers, vocab_size, hidden_dim, embedding_dim, output_dim, dropout_prob=0.5):
        super().__init__()
        self.num_layers = num_layers
        self.hidden_dim = hidden_dim
        self.embedding = nn.Embedding(vocab_size, embedding_dim)
        self.lstm = nn.LSTM(embedding_dim, hidden_dim, num_layers=num_layers, batch_first=True)
        self.dropout = nn.Dropout(dropout_prob)
        self.fc1 = nn.Linear(hidden_dim, hidden_dim // 2)
        self.bn1 = nn.BatchNorm1d(hidden_dim // 2)
        self.fc2 = nn.Linear(hidden_dim // 2, hidden_dim // 4)
        self.bn2 = nn.BatchNorm1d(hidden_dim // 4)
        self.output = nn.Linear(hidden_dim // 4, output_dim)

    def forward(self, x, hidden):
        x = self.embedding(x)
        x, hidden = self.lstm(x, hidden)
        x = x[:, -1, :]  # Last time step
        x = self.dropout(x)
        x = F.relu(self.fc1(x))
        x = self.bn1(x)
        x = F.dropout(x, p=0.5, training=self.training)
        x = F.relu(self.fc2(x))
        x = self.bn2(x)
        x = F.dropout(x, p=0.5, training=self.training)
        return self.output(x), hidden

    def init_hidden(self, batch_size):
        h0 = torch.zeros((self.num_layers, batch_size, self.hidden_dim)).to(device)
        c0 = torch.zeros((self.num_layers, batch_size, self.hidden_dim)).to(device)
        return (h0, c0)
```

DEMO RUN

We can add this small function to test how our model works with two test cases

```python
def predict_emotion(sentence, vocab, model, max_seq_length):
    #preprocess the sentence
    emotion = {0:'Sadness',1:'Joy',2:'love',3:'anger',4:'fear',5:'surprise'}
    words = [clean_text(word) for word in sentence.lower().split()]
    indices = [vocab[word] for word in words if word in vocab]

    padded_sequence = pad_sequences([indices], max_seq_length)
    #convert it to a tensor
    input_tensor = torch.Tensor(padded_sequence).to(device, dtype=torch.int64)
    #make the prediction
    model.eval()
    hidden = model.init_hidden(1)
    with torch.no_grad():
        output,_ = model(input_tensor,hidden)
        prediction = torch.argmax(output, axis=1).item()
    return emotion[prediction]
```
✓ 0.0s

TEST CASE 1

"I am scared of chocolate"

```
prediction = predict_emotion("I'm scared of chocolate", vocab, model, max_seq_length)
print(f"The predicted emotion is {prediction}\n")
```

✓ 0.0s

```
The predicted emotion is fear
```

Fear is the correct emotion

TEST CASE 2

“I love machine learning”

```
prediction = predict_emotion("I am so happy I took machine learning", vocab, model, max_seq_length)
print(f"The predicted emotion is {prediction}\n")
✓ 0.0s
The predicted emotion is Joy
```

Joy is the correct emotion

OUTCOMES

Application: A model like this can be used in messaging and social media programs as an intuitive tone indicator that helps users write text that reflects their intentions.

Implementation: The emotion detected by the model can be displayed as a small icon that changes as the user types to reflect the tone of the written message

Next Steps: If development on this model continues, it could be expanded to detect a wider rage of emotions.

Alternatively, it could be improved to detect the end of one statement, and beginning of another in longer pieces of text, allowing it to label specific parts of a message as different emotions.