Project Presentation: Information Extraction using NLP Techniques (NER, Named Entity Recognition)

- PRESENTER: MUHAMMET ALI ÖZTÜRK
- COURSE: CMP711 – NLP
- TOPIC: EXTRACTION OF NAMES FROM SAMPLE E-MAILS
- DATASET LINK: DATASET

- RELEVANT PAPERS:
- PAPER1: EXTRACTING PERSONAL NAMES FROM EMAIL: APPLYING NAMED ENTITY
- PAPER2: DEEP ACTIVE LEARNING FOR NAMED ENTITY RECOGNITION
- PAPER3: NATURAL LANGUAGE PROCESSING FOR INFORMATION EXTRACTION

Introduction

- **Objective:** Our goal is to extract meaningful information from noisy e-mails within the dataset. By employing an LSTM model and various feature techniques, we aim to accurately identify names among parsed words in the e-mails. The focus is on discerning names in challenging and cluttered email data, contributing to more effective natural language processing.

Dataset

- **CSpace Email Corpus (Kraut et al., 2004):**
- Origin: Carnegie Mellon University, 1997 management course.
- Participants: MBA students in simulated companies.
- Structure: Teams of 4-6 members in different market scenarios.
- Relevant Fields: "From," "Subject," and "Time."
- Subcorpora:
    - Mgmt-Game: Five-day period, used for training (day 1), tuning (day 4), and testing (day 5).
    - Mgmt-Teams: Training set involves messages between different teams than the test set.
- Purpose: Represents work-oriented communication in a small to medium-sized company.

Implementation Steps

1. We have defined our custom e-mail features for words such as if word is in a "from" line, or if word is capitalized or if word is followed by the bigram "and I". These suggestions were taken from the 1st reference paper.
2. The e-mails in the found dataset were very noisy. Some words were not meaningful and some of them were not even alpha. In order to deal with this corpora, we had to develop an unique way which is pos tagging using pre-defined models via nltk python library.
3. We extracted e-mail line by line and we have pos-tagged every word and taken into account the words that were completely alpha and the tags were not in (".", "", "(", ")", ":", and so on). By this way, we could eliminate words like antaloupe.srv.cs.cmu.edu!das-news.harvard.edu!noc.near.net!howland.reston.ans.net!usc!cs.utexas.edu!uunet!pipex!sunic!uts!iesd!news.iesd.auc.dk!habl
4. We have saved the words in a dictionary with index, built our words vocabulary after reading every email in the training data.
5. We labeled each word if that word is a name or not, if that word is capitalized or not and so on for each feature and label we need.
6. After reading training emails, there were ~17k sentences (lines) within emails and these sentences were containing around ~20k unique number of words. Around 2.5% of the words were names, ~23% were capitalized, ~4% were in from featured, ~0.13% were after "and I" bigram.
7. After getting our inputs ready and converting them to tensors (for tensorflow library), we built our LSTM model.LSTM model's code will be shared in the next slide.
8. We had to deal with weights since the dataset classes (positive and negative classes within e-mail) were not balanced. Thefore we had to build weight vector for our loss function. We used **Binary Cross Entropy with Logits Loss.**

Custom Feature Functions

```
In 2
def is_word_appearing_in_the_from(word, line):
    if 'from' in line.lower() and word.lower().strip() != 'from':
        return True
    else:
        return False

def followed_by_and_I(word, context):
    # if word is followed by "and I" return true, else false.
    context = context.lower()
    word = word.lower()
    if context.count(f"and i {word}") > 0:
        return True
    else:
        return False

def is_word_capitalized(word):
    return word[0].isupper()

```

LSTM model

```
In 121
```

```python
class LSTMModelWithFeatures(nn.Module):
    def __init__(self, vocab_size, embedding_dim, hidden_size, output_size):
        super(LSTMModelWithFeatures, self).__init__()
        self.embedding = nn.Embedding(vocab_size, embedding_dim)
        self.lstm = nn.LSTM(embedding_dim + 3, hidden_size, batch_first=True)
        self.fc = nn.Linear(hidden_size, output_size)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x, capital, from_field, bigram):
        word_embeddings = self.embedding(x)

        # Create tensors for additional features
        capital_feature = torch.unsqueeze(capital, dim=-1).float()
        from_field_feature = torch.unsqueeze(from_field, dim=-1).float()
        bigram_feature = torch.unsqueeze(bigram, dim=-1).float()

        # Concatenate all features with word embeddings
        combined_features = torch.cat([word_embeddings, capital_feature, from_field_feature, bigram_feature], dim=-1)

        lstm_out, _ = self.lstm(combined_features)
        output = self.fc(lstm_out)  # Assuming you want to use the last time step's output
        output = self.sigmoid(output)
        return output
```

```
Executed at 2024.01.17 11:53:53 in 42ms
```

Input Details

Size of y_tensor ~132k
Embedding dimension = 50
Hidden size = 256
Output size = 1 (Used Sigmoid Activation Function)
Loss: Binary CE with LogitsLoss

```
In 122
# Calculate class weights
positive_class_weight = len(y_tensor) / (y_tensor.sum() + 1e-5)  # Avoid division by zero
negative_class_weight = 1.0

print(positive_class_weight, negative_class_weight)
# Create a weight tensor for the BCEWithLogitsLoss
class_weights = []
for i, y in enumerate(y_tensor):
    if y == 1:
        class_weights.append(1*positive_class_weight*2)
    else:
        class_weights.append(1)

class_weights = torch.tensor(class_weights)
# Hyperparameters
vocab_size = len(word_to_index)
embedding_dim = 50
hidden_size = 256
output_size = 1

# Create the model and move it to CUDA
model_with_features = LSTMModelWithFeatures(vocab_size, embedding_dim, hidden_size, output_size)

# Loss function and optimizer
criterion = nn.BCEWithLogitsLoss(weight=class_weights)
optimizer = optim.Adam(model_with_features.parameters(), lr=0.001)
```

Difficulties

1. Found dataset was very noisy and it was not very easy to find other email datasets due to privacy issues in the field. Dealing with noises and preparing my inputs for my model took a lot of my time.
2. It was not possible to train a not balanced dataset without weights, had to introduce weights idea in order to be able to increase my accuracy and recall at the same time.
3. The technique I used within this project was not similar to the ones I have read in the paper but I tried getting inspired from papers and trying something novel. That's why it was not that easy to come up with a solution.

Results

On training set, there were around ~132k words and on test set, there were ~32k words. The recall and accuracy values after training my model for 20 epochs using learning_rate = 0.001 were as following:

Accuracy on test set = 61.2%

Recall on test set = 79%

25048 out of ~32k words in the test set were recalled correctly.