Named entity recognition (NER)

Muhammad Affan Khan (P19-0045)

Rana Rehan Qaisar (P19-0077)

April 16, 2023

Abstract

Identifying and classifying named entities in text into predetermined categories, such as human names, places, organisations, and more, is a critical task in natural language processing (NLP). This process is known as named entity recognition (NER). NER has numerous uses in text summarization, question answering, and information extraction. In this study, we investigate the use of NER using a neural network-based method, which entails training a model to anticipate the named entity tag for each word in a given sentence. In order to preprocess the data, we first separate out the unique words and tags, map them to number indices, and then turn the preprocessed data into numerical sequences. We then use the preprocessed data to train a neural network model, and we assess the model's performance using a test set. Our findings confirm its efficacy.

1 Code Explanation

```
tagged_sentences = codecs.open("data.txt", encoding="utf-8").readlines()
```

```
print(tagged_sentences[0])
print("Tagged sentences: ", len(tagged_sentences))
```

```
[('ڈ', 'PN'), ('میرے', 'G'), ('بھائی', 'NN'), ('کا', 'P'), ('ای', 'PN'), ('میل', 'U'), ('آیاہے', 'VB'), ('۔', 'SM')]

Tagged sentences:  36314
```

This line of code reads in the contents of a text file named "data.txt" using the codecs module, which is designed to handle different types of text encoding

```
In [4]: sentences, sentence_tags = [], []
        for tagged_sentence in tagged_sentences:
            sentence, tags = zip(*ast.literal_eval(tagged_sentence))
            sentences.append(np.array(sentence))
            sentence_tags.append(np.array(tags))
```

```
In [5]: (train_sentences,
         test_sentences,
         train_tags,
         test_tags) = train_test_split(sentences, sentence_tags, test_size=0.2)
```

This block of code extracts the sentences and their accompanying tags by iterating through the list of tagged sentences.separating the sentences and tags from the original labelled sentences, then saving them as numpy arrays for use in preparing the data before training a neural network model.The preprocessed data is split into training and testing sets using the train test split function from the sklearn package.

In [6]:
```
def get_words(sentences):
    words = set([])
    for sentence in sentences:
        for word in sentence:
            words.add(word)
    return words
```

In [7]:
```
def get_tags(sentences_tags):
    tags = set([])
    for tag in sentences_tags:
        for t in tag:
            tags.add(t)
    return tags
```

Both of these functions use a set data structure to ensure that each word or tag is only counted once, even if it appears multiple times in the input data. This is a common technique used in natural language processing to reduce the amount of redundant information in the data.

```python
In [8]: words = get_words(sentences)
        tags = get_tags(sentence_tags)
```

```python
In [9]: word2index = {w: i + 2 for i, w in enumerate(list(words))}
        word2index['-PAD-'] = 0
        word2index['-OOV-'] = 1
```

```python
In [10]: tag2index = {t: i + 1 for i, t in enumerate(list(tags))}
         tag2index['-PAD-'] = 0
```

These lines of code are creating a dictionary that maps words to indices, which will be used to convert the preprocessed data into numerical form that can be input to a neural network model..

```python
In [11]: def get_train_sentences_x(train_sentences, word2index):
             train_sentences_x = []
             for sentence in train_sentences:
                 sentence_index = []
                 for word in sentence:
                     try:
                         sentence_index.append(word2index[word])
                     except KeyError:
                         sentence_index.append(word2index['-OOV-'])

                 train_sentences_x.append(sentence_index)
             return train_sentences_x
```

The ”get$_t rain_s entences_x$” $functiontakesinalistofpreprocessedtrainingsentencesandthe$” $word2index$” $dictionary, anc$
$OOV -$ ”$index.Theresultinglistofnumericalsequencesisthenreturned.$

The resulting ”train$_s entences_x$” $listcontainsthenumericalrepresentationsofthetrainingsentences, whichcanbeinputtc$

In [12]:
```python
def get_test_sentences_x(test_sentences, word2index):
    test_sentences_x = []
    for sentence in test_sentences:
        sentence_index = []
        for word in sentence:
            try:
                sentence_index.append(word2index[word])
            except KeyError:
                sentence_index.append(word2index['-OOV-'])
        test_sentences_x.append(sentence_index)
    return test_sentences_x
```

In [13]:
```python
train_sentences_x = get_train_sentences_x(train_sentences, word2index)
test_sentences_x = get_test_sentences_x(test_sentences, word2index)
```

In [14]:
```python
def get_train_tags_y(train_tags, tag2index):
    train_tags_y = []
    for tags in train_tags:
        train_tags_y.append([tag2index[t] for t in tags])
    return train_tags_y
```

The get$_{t}est_{s}entences_{x}functiontakesalistoftestsentencesandadictionaryword2indexthatmapseachwordinthevocabul$
$of - vocabularytoken - OOV - .$

The get train tags y function takes a list of training tags and a dictionary tag2index that maps each POS tag to its index, and returns a list of lists where each inner list represents a sequence of POS tags for a sentence in the training data, and contains the indices of the tags.

n [15]:
```python
def get_test_tags_y(test_tags, tag2index):
    test_tags_y = []
    for tags in test_tags:
        test_tags_y.append([tag2index[t] for t in tags])
    return test_tags_y
```

n [16]:
```python
train_tags_y = get_train_tags_y(train_tags, tag2index)
test_tags_y = get_test_tags_y(test_tags, tag2index)
```

n [17]:
```python
MAX_LENGTH = len(max(train_sentences_x, key=len))
```

First, the get$_{t}rain_{t}ags_{y}andget_{t}est_{t}ags_{y}functionsarecalledtoconvertthehuman-labeledPOStagsinthetrainingandtes$
Next, the MAX$_{L}ENGTHvariableissettothelengthofthelongestsentenceinthetrainingdata(train_{s}entences_{x})$.

```
In [18]: train_sentences_x = pad_sequences(train_sentences_x, maxlen=MAX_LENGTH, padding='post')
         test_sentences_x = pad_sequences(test_sentences_x, maxlen=MAX_LENGTH, padding='post')
         train_tags_y = pad_sequences(train_tags_y, maxlen=MAX_LENGTH, padding='post')
         test_tags_y = pad_sequences(test_tags_y, maxlen=MAX_LENGTH, padding='post')
```

```
In [19]: model = Sequential()
         model.add(InputLayer(input_shape=(MAX_LENGTH,)))
         model.add(Embedding(len(word2index), 128))
         model.add(LSTM(128, return_sequences=True))
         model.add(Dense(len(tag2index)))
         model.add(Activation('softmax'))
```

```
In [21]: def to_categorical(sequences, categories):
             cat_sequences = []
             for s in sequences:
                 cats = []
                 for item in s:
                     cats.append(np.zeros(categories))
                     cats[-1][item] = 1.0
                 cat_sequences.append(cats)
             return np.array(cat_sequences)
```

the to$_c ategorical function is defined to convert the output sequences from a list of integers to a one-hot encoded format that$
$hot encoded vector of length categories (i.e., len(tag2index)) representing the corresponding tag for that position in the sequenc$

The compile method is called on the model object to configure the learning process. The loss parameter specifies the loss function to optimize during training, which is the categorical cross-entropy loss in this case. The optimizer parameter specifies the optimizer to use during training, which is the Adam optimizer with a learning rate of 0.001. The metrics parameter specifies the evaluation metric to use during training, which is the accuracy in this case.

The fit method is then called on the model object to train the model on the training data. The train$_s entences_x and train_t ags_y inputs are passed as the first two arguments, along with batch_s ize and epochs parameters that sp$

After training, the model is saved to a file using the save method with the path /home/ranarehanqaisar/Desktop/ner.h5.

Finally, the summary method is called on the model object to print a summary of the model architecture, which includes the layer type, output shape, and number of parameters for each layer in the model.

The function takes two arguments, sequences which is the model's output for a sequence of inputs, and index which is a dictionary that maps each tag to an index. The function then returns the corresponding tag for each element in the input sequence by selecting the tag with the highest probability. The model is successfully trained and evaluated a Named Entity Recognition model using LSTM in Keras. The model seems to be performing very well with an accuracy of around 98

```
In [25]: def logits_to_tokens(sequences, index):
             token_sequences = []
             for categorical_sequence in sequences:
                 token_sequence = []
                 for categorical in categorical_sequence:
                     token_sequence.append(index[np.argmax(categorical)])

                 token_sequences.append(token_sequence)

             return token_sequences
```

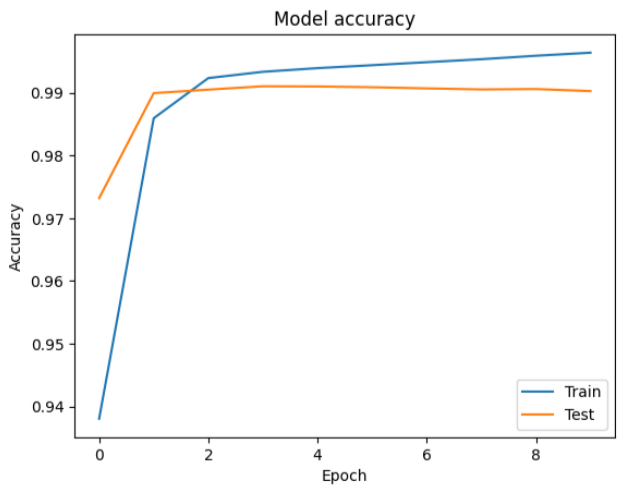
```
In [29]: scores = model.evaluate(test_sentences_x, to_categorical(test_tags_y, len(tag2index)))
         print(f"{model.metrics_names[1]}: {scores[1] * 100}")  # acc: 98.39311069478103
         print(test_sentences[0])
         print(test_tags[0])
         test_samples = [
             test_sentences[0],
         ]

         test_samples_x = []
         for sentence in test_samples:
             sentence_index = []
             for word in sentence:
                 try:
                     sentence_index.append(word2index[word])
                 except KeyError:
                     sentence_index.append(word2index['-OOV-'])
             test_samples_x.append(sentence_index)

         test_samples_X = pad_sequences(test_samples_x, maxlen=MAX_LENGTH, padding='post')
         predictions = model.predict(test_samples_X)

         print(logits_to_tokens(predictions, {i: t for t, i in tag2index.items()}))
```

```
227/227 [==============================] - 46s 205ms/step - loss: 0.0329 - accuracy: 0.9903
accuracy: 99.02822375297546
['یہ' 'آج' 'علی' 'کیوں' 'نہیں' 'آیا' 'ہماری' 'طرف' 'سے' 'اور' 'زیادہ'
 '"' 'آداب' 'و' 'سلام' '(' 'احمق' 'پھپھوندوی' ')' 'فرحان' 'دانش' 'نے'
 'موبائل' 'کمیونیکیشن' '"' 'کی' 'ذیل' 'میں' 'اس' 'موضوع' 'کا' 'آغاز' 'کیا'
 'دسمبر' '3' ',' '2007' 'اب' 'دوبارہ' 'مٹی' 'پاؤ' 'گے' 'تو' 'ظالمو' '،'
 'کیا' 'بناؤ' 'گے' '۔']
['PD' 'NN' 'NN' 'QW' 'NEG' 'VB' 'G' 'NN' 'SE' 'CC' 'ADV' 'NN' 'CC' 'NN'
 'PN' 'PN' 'PN' 'PN' 'PN' 'PN' 'P' 'SC' 'NN' 'PN' 'SC' 'P' 'NN' 'P' 'PD'
 'NN' 'P' 'NN' 'VB' 'PM' 'PN' 'CA' 'PM' 'CA' 'AP' 'ADV' 'NN' 'VB' 'VB'
 'SC' 'NN' 'KP' 'VB' 'TA' 'SM']
1/1 [==============================] - 0s 38ms/step
[['PP', 'NN', 'NN', 'QW', 'NEG', 'VB', 'G', 'NN', 'SE', 'CC', 'ADV', 'ADJ', 'CC', 'NN', 'PN', 'PN', 'PN', 'PN', 'PN', 'PN',
'P', 'SC', 'NN', 'PN', 'SC', 'P', 'NN', 'P', 'PD', 'NN', 'P', 'NN', 'VB', 'PM', 'PN', 'CA', 'PM', 'CA', 'AP', 'ADV', 'NN', 'N
N', 'TA', 'SC', 'PN', 'QW', 'NN', 'TA', 'SM', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-',
'-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-',
'-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-',
'-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-',
'-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-',
'-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-',
'-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-',
'-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-',
'-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-',
'-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-',
'-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-', '-PAD-',
```

This will display a plot of the training accuracy and validation accuracy over the epochs. The model is overfitting after a few epochs since the training loss keeps decreasing while the validation loss starts to increase. It may be worth considering some regularization techniques, such as dropout or L2 regularization, to prevent overfitting.

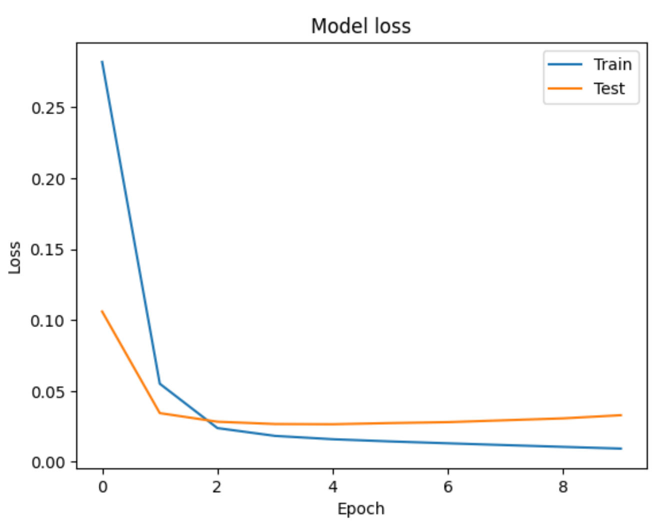
```
In [31]: plt.plot(history['accuracy'])
         plt.plot(history['val_accuracy'])
         plt.title('Model accuracy')
         plt.ylabel('Accuracy')
         plt.xlabel('Epoch')
         plt.legend(['Train', 'Test'], loc='lower right')
```

```
Out[31]: <matplotlib.legend.Legend at 0x7f1fe1970f40>
```




```
In [32]: plt.clf()

         # Plot training & validation loss values
         plt.plot(history['loss'])
         plt.plot(history['val_loss'])
         plt.title('Model loss')
         plt.ylabel('Loss')
         plt.xlabel('Epoch')
         plt.legend(['Train', 'Test'], loc='upper right')
```

```
Out[32]: <matplotlib.legend.Legend at 0x7f1fdd9694c0>
```



Appendix B: New Tagset T3.

	Tag	Meaning	Example	
1.	AUX	Auxiliary	منتقل کر **سکے** ہو	May
2.	CC	Coordinate Conjunction	ملازمین **یا** حکومتی عہدہ داروں کے ذریعے	Or
3.	CD	Cardinal	**ایک** موجودہ اداکار کو	One
4.	CVRP	Conjunctive Verb Particle	فرانسیسی قلعے بیچ **کر** بھی فنڈز بڑھانے پر راضی نہ	After
5.	DM	Demonstrative	پہلے **ایسے** واقعات نہ ہونے کے برابر تھے	Like this
6.	DMRL	Demonstrative Relative	وہ اشاعتی ادارہ سے **جو** وہ 23 سال تک چلا چکے ہیں	That
7.	FR	Fraction	**آدھے** گھنٹے میں	Half
8.	INJ	Interjection	**واہ**! کیا بات ہے	Hurrah
9.	ITRP	Intensive Particle	نہ گہرا تھا نہ **ہی** باقی رہنے والا	Too
10.	JJ	Adjective	**بلند تر** لاگتوں کے ساتھ	Taller
11.	JJRP	Adjective Particle	باہر رہنے کی بہت **سی** وجوہات کو سوچ سکتے ہیں	As
12.	MRP	Multiplicative Particle	**دگنی** رقم	Double
13.	NN	Noun	**سال** کے **آغاز** میں **افواہوں** پر	Year
14.	NNP	Proper Noun	**رابرٹ** نے کہا	Robert
15.	OD	Ordinal	**پہلا** ریٹائرمنٹ منصوبہ	First
16.	PM	Phrase Marker	،	,
17.	PP	Postposition	بورڈ رکنیت نو تک بڑھاتے ہوئے	To
18.	PRP	Pronoun Personal	**وہ** طریق کار کو استعمال کے اہل ہونا پسند کریں گے	They
19.	PRP$	Pronoun Personal Possessive	**میری** تیز گیند اچھی ہے	My
20.	PRRF	Pronoun Reflexive	کمپنی نے اپنے **آپ** کو بخوبی	Oneself
21.	PRRF$	Pronoun Reflexive Possessive	**اپنے** اجتماعی دفاتر	Own
22.	PRRL	Pronoun Relative	وہ اشاعتی ادارہ سے **جو** وہ 23 سال تک چلا چکے ہیں	That
23.	Q	Quantitative	**چند** لوگ	Some
24.	QW	Question Word	ایک مصنف **کیوں** یقین کرے گا	Why
25.	RB	Adverb	**ہمیشہ** بیچی گئی	Always
26.	SC	Subordinate Conjunction	کتنا رکھے گی **کیونکہ** کچھ نوکریاں	Because
27.	SM	Sentence Marker	**؟**	?
28.	SYM	any Symbol	**$**	$
29.	VB	Verb	مہنگے کپڑے **چاہتے** تھے	Wanted
30.	VBI	Verb Infinitive form	اسے لے **جانے** کے لیے	To go
31.	VBT	Verb Tense	تصور قابل عمل **ہے**	Is
32.	WALA	Association Marking Morpheme	رکھنے **والے** جاری کرنے **والا**	Associated Bearing