CSCI/LING 5832: Natural Language Processing

Assignment 3: Named Entity Recognition on S21-gene dataset

Team: **Bhargav Shandilya | Dananjay Srinivas | Arvind Sreenivas**

Table of Contents

Introduction

Named-Entity-Recognition (NER) is a sub-task of information extraction that seeks to identify and classify named entities. To find the references of genes in the given dataset we have tried multiple approaches, getting the best results using a CRF approach to train a model for classifying the IOB tags in the S21-gene-train dataset.

Methods used for NER tagging:

Primary Method:

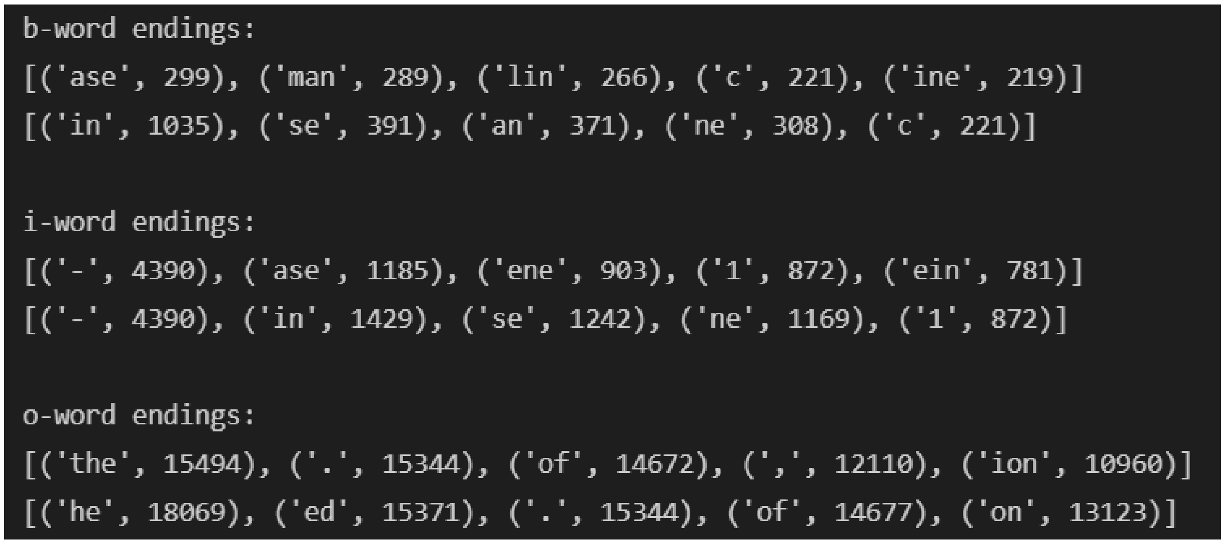
CRF

Description: A Conditional Random Field (CRF) is a model used to predict the most likely sequence of labels that correspond to a sequence of inputs. A CRF takes context into account; the linear chain CRF predicts sequences of labels for sequences of input samples. In order to use CRF, we have an enhanced feature set and more features which can be used by the model to predict the tags correctly.

The primary library used to train the data was the scikit-learn CRF suite. NumPy and Pandas were used for data exploration and feature vector construction. Eli5 was used for visualization. The English stopword list was obtained from the NLTK library.

Data Exploration and Feature Extraction:

Apart from generic features such as uppercase, lowercase, and the presence of digits, we identified more nuanced features through data exploration. Words that were tagged as ‘B’ or ‘I’ in the training data had certain features that made them distinct from ‘O’ tags. For instance, our initial data exploration revealed that ‘-ase’ and ‘-man’ were the most common 3-letter word endings for b-words. Similarly, the most common 3-letter endings for ‘I’ tagged words were ‘-ase’ (words such as ‘lipase’), ‘-ene’, and so on. The image below shows the most common 3-letter and 2-letter word-endings along with the number of occurrences for each class:



```
b-word endings:
[('ase', 299), ('man', 289), ('lin', 266), ('c', 221), ('ine', 219)]
[('in', 1035), ('se', 391), ('an', 371), ('ne', 308), ('c', 221)]

i-word endings:
[('-', 4390), ('ase', 1185), ('ene', 903), ('1', 872), ('ein', 781)]
[('-', 4390), ('in', 1429), ('se', 1242), ('ne', 1169), ('1', 872)]

o-word endings:
[('the', 15494), ('.', 15344), ('of', 14672), (',', 12110), ('ion', 10960)]
[('he', 18069), ('ed', 15371), ('.', 15344), ('of', 14677), ('on', 13123)]
```

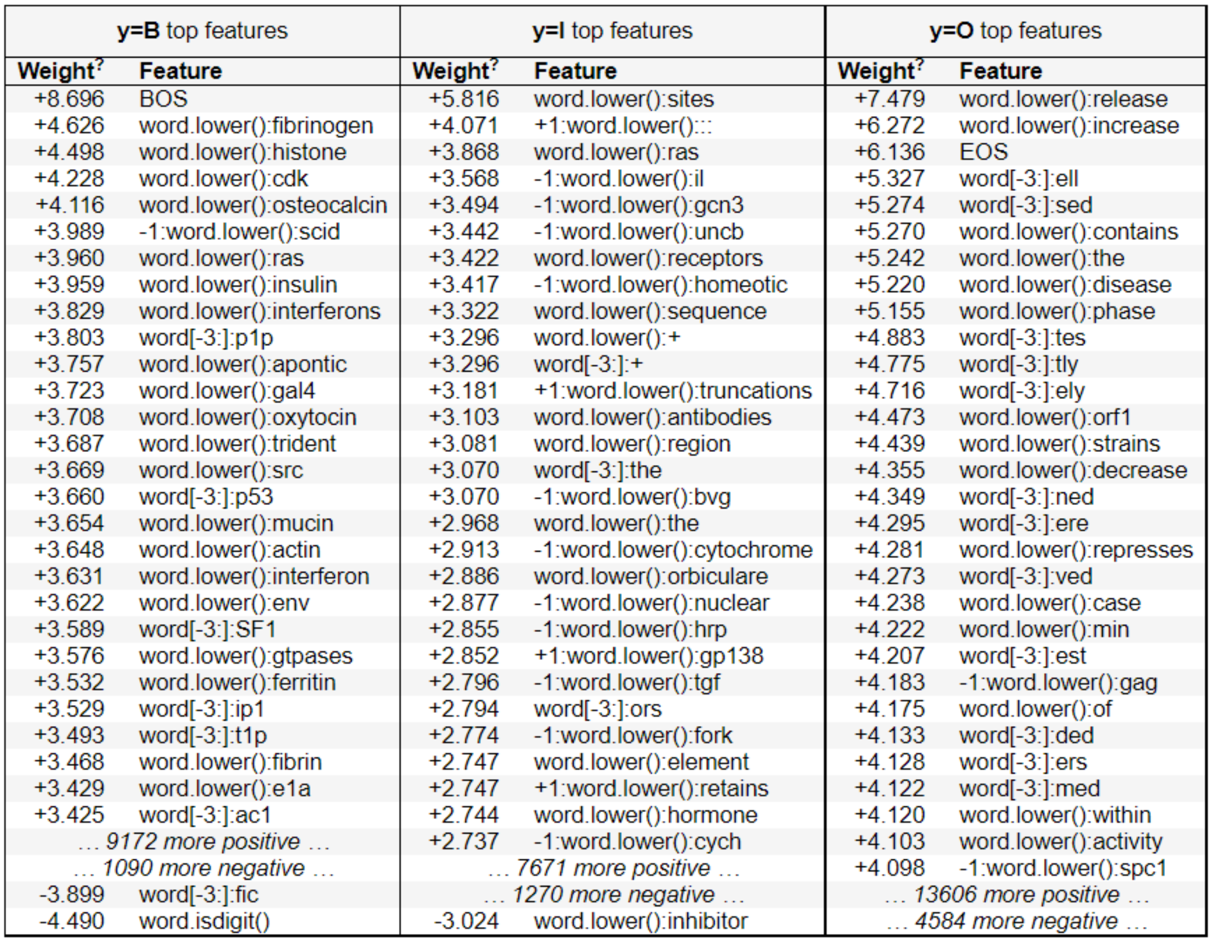
We also printed out the top 20 most frequently occurring words in each of the three label classes. As a result, we found that some of the most common O-words are stopwords (the, of, on, and so on). This led us to define our feature list (mostly boolean values) as follows:

- Uppercase
- Presence of digits in word
- Presence of hyphen in word
- The last 3 letters of the word
- The last 2 letters of the word
- Camel case
- First word capitalized
- Word is at the beginning of a sentence (BOS)

- Word is at the end of a sentence (EOS)

Additionally, we also created a set of the most commonly occurring *unique* words for each of the three tags by taking the set difference for each tag.

The top features selected by the model and their corresponding weights are shown below:



y=B top features		**y=I** top features		**y=O** top features	
Weight?	**Feature**	**Weight?**	**Feature**	**Weight?**	**Feature**
+8.696	BOS	+5.816	word.lower():sites	+7.479	word.lower():release
+4.626	word.lower():fibrinogen	+4.071	+1:word.lower():::	+6.272	word.lower():increase
+4.498	word.lower():histone	+3.868	word.lower():ras	+6.136	EOS
+4.228	word.lower():cdk	+3.568	-1:word.lower():il	+5.327	word[-3:]:ell
+4.116	word.lower():osteocalcin	+3.494	-1:word.lower():gcn3	+5.274	word[-3:]:sed
+3.989	-1:word.lower():scid	+3.442	-1:word.lower():uncb	+5.270	word.lower():contains
+3.960	word.lower():ras	+3.422	word.lower():receptors	+5.242	word.lower():the
+3.959	word.lower():insulin	+3.417	-1:word.lower():homeotic	+5.220	word.lower():disease
+3.829	word.lower():interferons	+3.322	word.lower():sequence	+5.155	word.lower():phase
+3.803	word[-3:]:p1p	+3.296	word.lower():+	+4.883	word[-3:]:tes
+3.757	word.lower():apontic	+3.296	word[-3:]:+	+4.775	word[-3:]:tly
+3.723	word.lower():gal4	+3.181	+1:word.lower():truncations	+4.716	word[-3:]:ely
+3.708	word.lower():oxytocin	+3.103	word.lower():antibodies	+4.473	word.lower():orf1
+3.687	word.lower():trident	+3.081	word.lower():region	+4.439	word.lower():strains
+3.669	word.lower():src	+3.070	word[-3:]:the	+4.355	word.lower():decrease
+3.660	word[-3:]:p53	+3.070	-1:word.lower():bvg	+4.349	word[-3:]:ned
+3.654	word.lower():mucin	+2.968	word.lower():the	+4.295	word[-3:]:ere
+3.648	word.lower():actin	+2.913	-1:word.lower():cytochrome	+4.281	word.lower():represses
+3.631	word.lower():interferon	+2.886	word.lower():orbiculare	+4.273	word[-3:]:ved
+3.622	word.lower():env	+2.877	-1:word.lower():nuclear	+4.238	word.lower():case
+3.589	word[-3:]:SF1	+2.855	-1:word.lower():hrp	+4.222	word.lower():min
+3.576	word.lower():gtpases	+2.852	+1:word.lower():gp138	+4.207	word[-3:]:est
+3.532	word.lower():ferritin	+2.796	-1:word.lower():tgf	+4.183	-1:word.lower():gag
+3.529	word[-3:]:ip1	+2.794	word[-3:]:ors	+4.175	word.lower():of
+3.493	word[-3:]:t1p	+2.774	-1:word.lower():fork	+4.133	word[-3:]:ded
+3.468	word.lower():fibrin	+2.747	word.lower():element	+4.128	word[-3:]:ers
+3.429	word.lower():e1a	+2.747	+1:word.lower():retains	+4.122	word[-3:]:med
+3.425	word[-3:]:ac1	+2.744	word.lower():hormone	+4.120	word.lower():within
… 9172 more positive …		+2.737	-1:word.lower():cych	+4.103	word.lower():activity
… 1090 more negative …		*… 7671 more positive …*		+4.098	-1:word.lower():spc1
-3.899	word[-3:]:fic	*… 1270 more negative …*		*… 13606 more positive …*	
-4.490	word.isdigit()	-3.024	word.lower():inhibitor	*… 4584 more negative …*	

Training parameters:

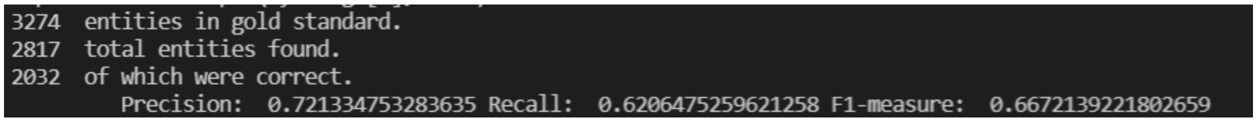
```
algorithm='lbfgs',
    c1=0.1,
    c2=0.1,
    max_iterations=200,
    all_possible_transitions=False
```

Here, c_1 and c_2 are the regularization parameters. The LBFGS (Limited-memory Broyden–Fletcher–Goldfarb–Shanno) algorithm is an iterative means of finding a solution to an unconstrained optimization problem. Similar to the gradient descent algorithm, BFGS also attempts to minimize the loss function by finding the minima of the function using a Hessian matrix. The computational complexity never crosses O(n^2).

Results:

Entity-level (obtained from running the evalNER.py file):

After running the evaluation script on the obtained predictions, we got the results shown below for our test set:



```
3274  entities in gold standard.
2817  total entities found.
2032  of which were correct.
        Precision:  0.721334753283635 Recall:  0.6206475259621258 F1-measure:  0.6672139221802659
```

The precision was 0.72, recall was 0.62, and the F1-measure was 0.67.

Alternate Methods:

Logistic Regression

We also attempted to use a traditional logistic regression model (similar to assignment 2) with the same features that were extracted for the CRF model. To account for the imbalance in the data, we attempted to downsample the O-tagged words. This did improve the overall precision and recall to some extent, but the improvement was somewhat insignificant. Using balanced class weights for training resulted in slightly better evaluation metrics. The model did not perform as optimally as the CRF model as seen in the results below:

	Precision	Recall	F1-Score	Support
B	35.71%	55.78%	0.44	1660
I	35.82%	43.16%	0.39	246
O	94.67%	90.73%	0.93	3449
Entity-level	**0.2314**	**0.3157**	**0.2459**	**5252**

Deep Learning - LSTM

Intuition :

LSTMs can be used to capture the temporal nature of a sentence, and leverage Deep Learning to find the token types (B, I or O labels). There are several architectures and resources that have explored NER tagging, for example : [1], [2] and [3].

Usually, tagging entities also classifies the types of entity (person, organization, place, etc). Our homework however overlooks this distinction - arguably making this approach more lucid and approachable.

We proposed to build a Deep Learning - LSTM model given its efficacy on the other NER-tagging problems.

Constraints :
We faced some difficulties when working with LSTMs, but mostly due to our inexperience and lack of computational resources. Since none of our team members had access to GPUs, we ran training on GCP Cloud Instances with our free credits.

We would like to highlight 2 major issues when it came to training LSTMs :
- Training on CPUs took a lot of time - around 10min/epoch in our final iteration.
- Shallow models could not generalize or failed to learn well, so we needed deeper models.

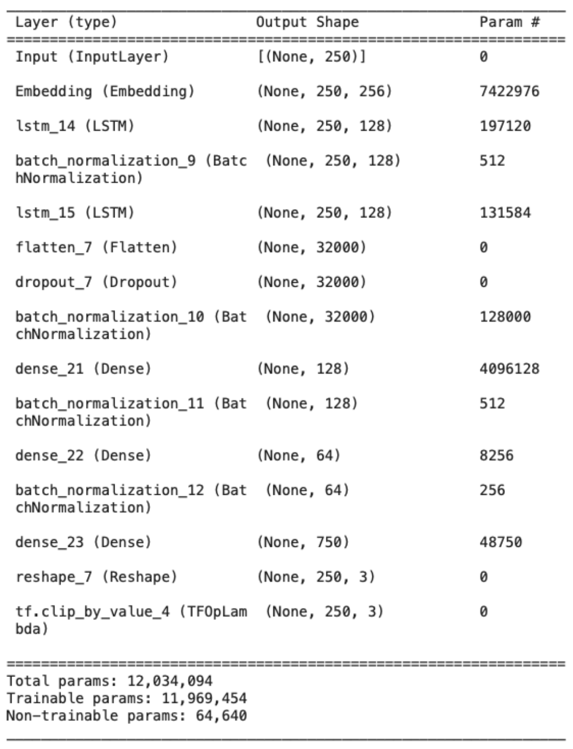
Due to these issues, we needed to find a balance between practical and theoretical challenges of building a model.

Architecture :
From our research, we concluded that we will start with a 2-stacked LSTM and a shallow fully-connected network would suffice. We felt that this architecture was a good trade-off between training time and model depth.

We then used BertTokenizerFast API from HuggingFace to tokenize words. The entity labels provided in the dataset were not readily tokenizable because of a mismatch in the way punctuation was treated. Aside from this reconciliation, we have not changed the way BertTokenizerFast treats tokenization.

The LSTM model outputs a matrix of shape [batch_size, sequence_length, 3]. We have fixed sequence_length at 250 because the maximum sequence length we encountered in training was 219 tokens. The last dimension of 3 corresponds to the tags that we need to predict - “B”, “I”, “O”.

We used the standard CategoricalCrossEntropy loss and Adam optimizer provided by Tensorflow to run the backprop.



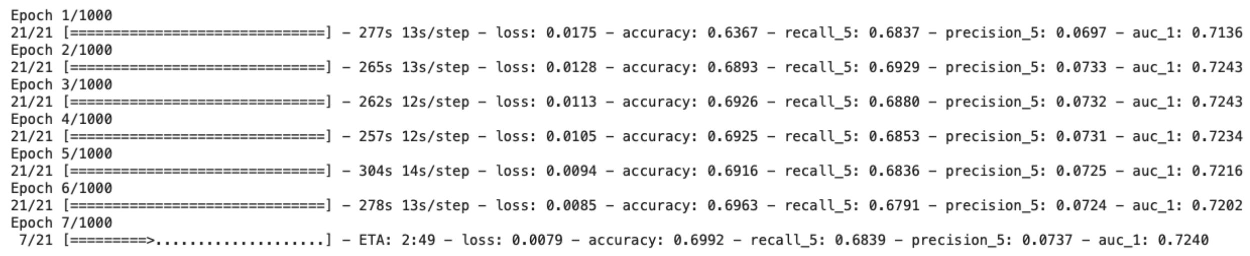
```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 Input (InputLayer)          [(None, 250)]             0

 Embedding (Embedding)       (None, 250, 256)          7422976

 lstm_14 (LSTM)              (None, 250, 128)          197120

 batch_normalization_9 (Batc  (None, 250, 128)         512
 hNormalization)

 lstm_15 (LSTM)              (None, 250, 128)          131584

 flatten_7 (Flatten)         (None, 32000)             0

 dropout_7 (Dropout)         (None, 32000)             0

 batch_normalization_10 (Bat  (None, 32000)            128000
 chNormalization)

 dense_21 (Dense)            (None, 128)               4096128

 batch_normalization_11 (Bat  (None, 128)              512
 chNormalization)

 dense_22 (Dense)            (None, 64)                8256

 batch_normalization_12 (Bat  (None, 64)               256
 chNormalization)

 dense_23 (Dense)            (None, 750)               48750

 reshape_7 (Reshape)         (None, 250, 3)            0

 tf.clip_by_value_4 (TFOpLam  (None, 250, 3)           0
 bda)

=================================================================
Total params: 12,034,094
Trainable params: 11,969,454
Non-trainable params: 64,640
_________________________________________________________________
```

Training :
Since the training was done on a 4 core CPU, it took a long time - and we decided to fix the training epochs to around 100.

We are tracking 3 metrics in addition to accuracy : Recall, Precision and AUC. Due to the bias towards ‘O’ labels, the model initially performs well on precision, presumably over-predicting “O”. After training for a few epochs, there is a tradeoff between Recall and Precision - we see an increase in Recall over epochs as precision reduces.

One of the problems we are running into while training is exploding gradients - causing nan errors. We have implemented multiple BatchNormalizations for this reason.

Results :
We were unable to observe good results



```
Epoch 1/1000
21/21 [==============================] - 277s 13s/step - loss: 0.0175 - accuracy: 0.6367 - recall_5: 0.6837 - precision_5: 0.0697 - auc_1: 0.7136
Epoch 2/1000
21/21 [==============================] - 265s 13s/step - loss: 0.0128 - accuracy: 0.6893 - recall_5: 0.6929 - precision_5: 0.0733 - auc_1: 0.7243
Epoch 3/1000
21/21 [==============================] - 262s 12s/step - loss: 0.0113 - accuracy: 0.6926 - recall_5: 0.6880 - precision_5: 0.0732 - auc_1: 0.7243
Epoch 4/1000
21/21 [==============================] - 257s 12s/step - loss: 0.0105 - accuracy: 0.6925 - recall_5: 0.6853 - precision_5: 0.0731 - auc_1: 0.7234
Epoch 5/1000
21/21 [==============================] - 304s 14s/step - loss: 0.0094 - accuracy: 0.6916 - recall_5: 0.6836 - precision_5: 0.0725 - auc_1: 0.7216
Epoch 6/1000
21/21 [==============================] - 278s 13s/step - loss: 0.0085 - accuracy: 0.6963 - recall_5: 0.6791 - precision_5: 0.0724 - auc_1: 0.7202
Epoch 7/1000
 7/21 [=========>....................] - ETA: 2:49 - loss: 0.0079 - accuracy: 0.6992 - recall_5: 0.6839 - precision_5: 0.0737 - auc_1: 0.7240
```

BioBERT

BioBERT is a biomedical language representation model designed for biomedical text mining tasks such as biomedical named entity recognition, relation extraction, question answering, etc. This use case matches with the domain of our S21 gene dataset. We chose this model hoping to get good results due to the domain similarity.

We trained the model using the github repository mentioned above and our dataset which we split into train, test, dev and devel files as required by the model. This model gave us an overall token level F1 score of 86 on training for 10 epochs. The evaluation at the entity level was unsuccessful due to errors with the code which we were unable to resolve.

References

CRF

- CRFs explained
- aleju/ner-crf: CRF to detect named entities (primarily names of people)
- NER using CRF

BioBERT

- https://github.com/dmis-lab/biobert

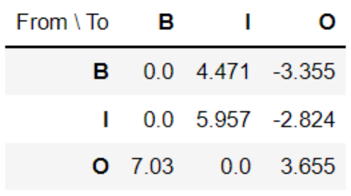
Appendix

Token-level results for primary model:

	Precision	Recall	F1-Score	Support
B	90.62%	91.04%	0.91	1395
I	88.18%	86.43%	0.87	1113

O	99.96%	100.0%	1.00	2758
Entity-level	**0.72**	**0.62**	**0.67**	**3274**

The normalized transition scores obtained from the CRF model for these features are shown below:



From \ To	**B**	**I**	**O**
B	0.0	4.471	-3.355
I	0.0	5.957	-2.824
O	7.03	0.0	3.655