CS5740: Final
Github group: final-exam-team
https://github.com/cornell-cs5740-18sp/final-exam-team

Ang Li al2386
Iris Zhang wz337

1 Introduction

In this assignment, our task is to build a named-entity recognizer for Twitter text, a key component of our tweet-scanning system. We have used different models for the task, and our best model achieved **0.5634** on leaderboard test set.

2 Approach

We explored different models for the given task, including traditional classification models and neural networks.

2.1 Baseline/Logistic

Logistic Regression is widely used for classification tasks and is known for its simplicity, which, we think, should be suitable for our baseline.

In the Logistic Regression model, L2 penalty is used as loss function and lib-linear algorithm is used to solve the optimization problem. For the optimization problem, binary class L2 penalized logistic regression minimizes the following cost function:

$$\min_{\omega,c} \frac{1}{2}\omega^T\omega + C\sum_{i=1}^{n}\log(\exp(-y_i(X_i^T\omega + c)) + 1). \quad (1)$$

In order to reduce the unbalanced of class data in the dataset, class weights were assigned to each class by the following:

$$\frac{n\_samples}{n\_classes * np.bincount(y)} \quad (2)$$

2.2 CRF

Unlike models such as RNN/LSTM, it can take into account long-term contextual information. It takes more of a linear weighted combination of local features of the entire sentence (scans the entire sentence through a feature template). The key point is that the model of CRF is p(y — x, w). Note that y and x are both sequences. The optimization is a sequence y = (y1, y2, ..., yn). Instead of y-t at a certain moment, we find a sequence with the highest probability y = (y1, y2, ..., yn) so that p(y1, y2, ..., yn— x, w) is the highest, and it is calculated as a union. Probability is to optimize the entire sequence (final target) instead of splicing the best of each moment. In this regard, CRF is better than LSTM.

2.3 Bidirectional-LSTM - CNN - CRF

We found the above methods require handcrafted features that is costly to develop and thereby difficult to generalize. After literature reviews, we found that using combinations of bidirectional LSTM, CNN, and CRF require no feature engineering but is able to achieve state-of-the-art performance. This model is inspired by Ma et al in "End-to-end Sequence Labeling via Bidirectional LSTM-CNNs-CRF".

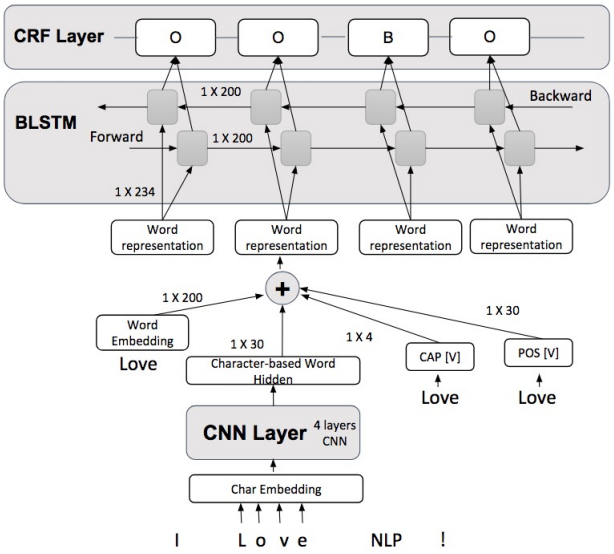


Figure 1: Model Architecture

Figure 1 illustrates the architecture. We first used CNN to encode character level info of a word as CNN can capture morphological information. We used a

vector of size 4 (CAP in Figure 1) to capture whether the word is uppercase, capitalized, lowercase, or unknown. We used a vector of size 30 to represent CMU Twitter POS Tagger(POS in Figure 1) info (27 different taggers). We concatenate all above with word embeddings and feed them into a bi-directional LSTM to model context information from the past and future. Then, we use a sequential CRF to decode labels for the entire sentence, which utilized both transition probability and label emission probability.

2.4 Ensemble Model

The goal of ensemble methods is to combine predictions of several base estimators built with a given learning algorithm in order to improve generalization/robustness over a single estimator. The driving principle is to build several estimators independently and then to average their predictions. On average, the combined estimator is usually better than any of the single base estimator because its variance is reduced. Consider the datasets contains a lot of 'O' and only small amount of 'B' and 'I'. We used two different ensemble way, majority vote or union. More details are illustrated in the following sections.

3 Experimental Setup

3.1 Preprocessing

Traditional Model We vectorized all words' features with one-hot encoding. Suppose we have a prefix feature which contains first 3 characters of a word, then we build a dict that contains all unique prefix and use one-hot encoding. See figure 2 for features we generated for word '@paulwalk'.

```
{'cur-word': '@paulwalk',
 'cur-word-class': 'UNK',
 'next-word': 'It',
 'next-word-class': 'initCap',
 'post-1': 'k',
 'post-2': 'lk',
 'post-3': 'alk',
 'post-4': 'walk',
 'pre-1': '@',
 'pre-2': '@p',
 'pre-3': '@pa',
 'pre-4': '@pau',
 'prev-word': 'START2',
 'prev-word-class': 'containsDigit'}
```

Figure 2: Features Extract

LSTM Each unique word was assigned unique integer, which were then used to index a lookup table that provided embeddings to the LSTM encoder. All unknown words were treated as UNK.

3.2 Logistic model/ CRF

For logistic model and CRF model, we simply took advantage from sklearn library. And we also used random search for hyper-parameters search and got the best hyper-parameters of **Logistic learning rate = 1e7** and **CRF c1=0.0173, c2=0.0052**. Run time for CRF and Logistic model is about 2 minutes on 2017 MACBOOK PRO (CPU machine).

3.3 BLSTM + CNN + CRF

We initialized our parameters based on the paper and did random search for parameters. Our best model used: 4 convolutional layers, 30 unit char embeddings for char CNN; 200 for word embeddings; 4 unit for CAP embeddings; 30 for POS embeddings; 2 layer bidirectional lstm with hidden dimension of 200 (50% dropout); l2 regularization of 1e-8 and gradient clipping of 5; parameter optimization is performed with mini-batch SGD with batch size 10 based on average loss and momentum 0.9 with 0.02 learning rate and 0.05 lr decay. We used early stopping based on performance on dev sets. The best parameter appears around 10 epochs on our CPU machine with an epoch takes around 1 minute.

3.4 Final Model

We ensembles three individual models, sklearn.logistic, sklearn-crfsuite and BLSTM + CNN + CRF to get our best results. We used union to ensemble them, which means if any model predict a word as Named Entity, it would be labeled as Named Entity.

4 Results Analysis

method	train-F1	dev-F1
sklearn.logistic	0.95	0.47
sklearn-crfsuite	0.99	0.49
Bi-LSTM+CRF+CNN	0.96	0.66
ensemble-model	0.97	0.68

Table 1: Dev Set Acc on different model

4.1 Logistic Regression

We performed feature engineering on our logistic model. The following table shows how different features affect model performance. All features were added to baseline: lemma, prev-word, prev-prev-word, next-word, next-next-word As seen from the table, the most important features are prev-word, next-word, prefix, postfix and word-class. One inter-

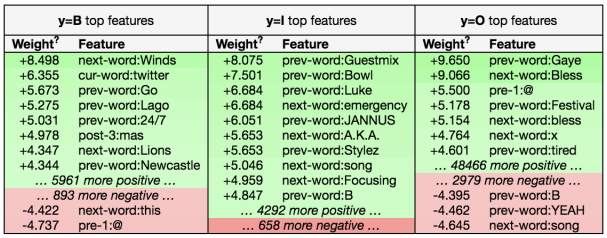
Features	dev-F1
baseline	0.24
prefix(1,2,3,4), postfix(1,2,3,4)	0.43
word-class, length of tweet	0.39
nltk.postag	0.38
final features(figure.2)	0.47

Table 2: LR Dev Set Acc on different features

esting finding is that nltk.postag does not really affect the results. That makes sense as the information for transition probability have already been captured by prev-word and next-word context.

4.2 CRF

For pure CRF model, we also performed feature engineering and we used **eli5** to analyze our different features. As shown in figure3, the features, like prev-word: Go, cur-word: twitter, contribute a lot to B label since lots of twitters includes: Go, Cleveland. And feature like prefix with '@' were always 'O' label since all tweets have '@' symble at the start of the sentence.



y=B top features		y=I top features		y=O top features	
Weight[?]	Feature	Weight[?]	Feature	Weight[?]	Feature
+8.498	next-word:Winds	+8.075	prev-word:Guestmix	+9.650	prev-word:Gaye
+6.355	cur-word:twitter	+7.501	prev-word:Bowl	+9.066	next-word:Bless
+5.673	prev-word:Go	+6.684	prev-word:Luke	+5.500	pre-1:@
+5.275	prev-word:Lago	+6.684	next-word:emergency	+5.178	prev-word:Festival
+5.031	prev-word:24/7	+6.051	prev-word:JANNUS	+5.154	next-word:bless
+4.978	post-3:mas	+5.653	next-word:A.K.A.	+4.764	next-word:x
+4.347	next-word:Lions	+5.653	prev-word:Stylez	+4.601	prev-word:tired
+4.344	prev-word:Newcastle	+5.046	next-word:song	*... 48466 more positive ...*	
... 5961 more positive ...		+4.959	next-word:Focusing	*... 2979 more negative ...*	
... 893 more negative ...		+4.847	prev-word:B	-4.395	prev-word:B
-4.422	next-word:this	*... 4292 more positive ...*		-4.462	prev-word:YEAH
-4.737	pre-1:@	*... 658 more negative ...*		-4.645	next-word:song

Figure 3: CRF Features

Final F1 score of CRF is **0.49** on Dev set.

4.3 Bidirectional-LSTM - CNN - CRF

Features	dev-F1
Baseline	0.215
CharCNN	0.41
CharCNN+POS+CAP	0.54
CharCNN+POS+CAP+TwitterGlove	0.66

Table 3: Dev-F1 with Different Word Representations for BLSTM + CRF

Results in this section is based on the architecture in Figure 1, except word representations is generated differently. Our baseline first created a word dictionary for training and vectorized each word by a one-hot vector. Then, we added feature one at a time to examine how it boosts our performance.

We tried the result with Dev-F1 of 0.66 on leaderboard with confidence yet we only got 0.54 on test set. We suspect that the problem is that the distribution of training, dev, and test is different so that our model overfits to training. Therefore, this inspire us to try an Ensemble model with the models above.

4.4 Ensemble Model (Final Model)

From table 4, we find that all models tend to predict less Name Entity than ground truth. Thus we used union to build our ensemble model which will label a word as Name Entity if any of individual model label it is.

method	Dev-Prec	Dev-Recall
sklearn.logistic	0.6436(186/289)	0.4052(186/459)
sklearn-crfsuite	0.7216(197/273)	0.4294(197/459)
Bi-LSTM+CRF+CNN	0.6436(233/362)	0.5076(233/459)
ensemble-model	0.6212(246/396)	0.5359(246/459)

Table 4: Prec and Recall Rate on Dev

5 Conclusion

Our final test result is **0.5634** on leaderboard.

5.1 Error Analysis

We manually inspect where our model classify wrong and find out that entities consisting of a sequence of very common words and entities sensitive to cases are prone to mistakes.

For example, "demi lovato (predict: O O, true: B I)" is the name of a singer, but our model failed as the original tweeter does not have proper cases. Also, the model tends to fail when detecting names of movies, shows, and songs consisting of very common words (for example, the song name "Save My Life"). Beside, the model does not seem to handle long sequence well. The Drama name "The Fault in Our Start", BI-III, is predicted as BIOBI, as in is a lowercase word. So one mistake in the middle of the sequence leads to another mistake. Therefore, we could have another embeddings to capture n-gram info as an extra input for our word representation. The n could be the average length of Named Entity in training.

Another mistake is caused by some people like to capitalize their sentence on twitter to express their feelings. Some of the errors are resulted from high degree of lexical and character variation on Twitter. We could avoid this problem by examining whether a sequence is ALL CAP in preprocessing. If yes, then we lowercase the words in the sentence.

5.2 Future Work

From the errors above, we thought the most important method to boost performance of our classifiers is to train with more data to help algorithms detect signal better, especially when distribution of the split is different. Another interesting finding is different language may adopt different models due to the nature of the language. For example, CharCNN would not make sense for characters in Chinese, as strokes doe not contain any info. Instead, character-level information were handled by tree-based model for Chinese NER task.