Nationality Identification using Name

Wei Ren
weiren@stanford.edu

ZhenHuan Hong
jhong812@stanford.edu

Jiate Li
jiateli@stanford.edu

June 10, 2020

Abstract

For international students, it can be stressful to work with teammates who are all native speakers. On the other hand, working with fellow students speaking the same primary language can make communication easier and promote team chemistry by bonding outside the work (same cultural background). The motivation of this project is to explore the viability of identifying a person's nationality solely based on his/her name using deep learning concept.
Keywords: *Name; Nationality; Deep Learning; RNN.*

1 Introduction

The goal of this project is to investigate the possibility of identifying a person's nationality solely based on his/her name. We explored two different deep learning approaches for the problem:

- multinomial logistic regression using deep neural network
- Long Short Term Memory (LSTM) with character embeddings

We used multinomial logistic regression as our baseline model. By comparing the prediction accuracy of the two models, we saw that the LSTM architecture performs significantly better than the baseline model. From this understanding, we further explored the influence of adding extra features to the architecture.

2 Dataset

The datasets we used were from GitHub (github.com/d4em0n/nationality-classify). The original author crawled the name data from Wikipedia. Each dataset contains numerous names of people coming from the same nationality. In total, there are 18 nationalities (Russia, China, Korea, Poland, Scotland, Italy, US/UK, France, Japan, Greece, Spanish, India, Turkey, Indonesia, Vietnam, Czech Republic) with a total of 191736 names. Since the data distribution (number of data entry per country) is close to uniform (each nationality contains around 9800 names), there was no need to clean the datasets. Before proceeding to data preprocessing, we combine all the datasets into one csv file with two columns of data.

3 Methodology

3.1 Data preprocessing

For traditional natural language processing problem, the normal method is to divide an example (sentence) into individual words and convert each word into a one-hot key vector and feed them into the model. For our problem, this is not feasible since each example (name) only contains at most four words. In order to extract more information from a name, we need to divide each name into characters. The n-gram language model as mentioned in Jinhyuk Lee et al's paper proposes a solution. The general idea is to separate individual words into n-gram (n depends on choice. in our case, n = 1,2,3) character sequences using sliding window fashion. We constructed the training corpus for the three n-gram sequences (uni-gram: single character, bi-gram: two-character combination, tri-character combination) based on our datasets. With these dictionaries, we can map each name example to three distinct sequences.

3.2 Softmax Regression Model

For the softmax regression model, we can combine the three sequences into one input vector and feed them into deep neural nets (NN). The final layer of the NN is a softmax layer that can generate probabilities of each possible identification outcome (18 countries). The loss function for the softmax layer is as followed:

$$L(\hat{y}, y) = \sum_{n=1}^{18} y_i \log \hat{y}$$

Where $\hat{y}$ is the predicted nationality and y is the actual nationality.

The difficult part for implementing the traditional deep NN for the problem is that we have to unify the length of the input. Since examples tend to have different lengths, their n-gram sequences' sizes also vary. To address the issue, we decided to pad each n-gram vector with zeros to match their length with the maximum length we have in the datasets. After the padding, for each example, we were able to concatenate the three n-gram vectors and treat it as the input vector.

For the baseline model, we tuned several hyperparameters (Mini-batch size, number of epochs, number of layers, number of units in each layer) to explore the best performer.

3.3 LSTM Model

For the LSTM model, we referenced Jinhyuk Lee et al's work (github.com/jhyuklee/ethnicity-tensorflow) as the starting point for our model architecture. The general idea is as followed:

- Use char2vec approach (skip-grams algorithm) to find embeddings for every n-gram character(s) we have in the training corpus.
- For each example, feed the three embedding sequences (unigram, bigram, trigram) into three distinct one-layer LSTM models (many-to-one structure layer) respectively.
- Concatenate the three outputs from the LSTM layers and feed the result into a fully-connected layer (FC) before putting the logits into the final softmax layer.

- The cost function is the same as what we used for the baseline model. The model is trained with back propagation through time.

Figure 1 in the Appendix shows the workflow diagram of the model. It was taken from Jinhyuk Lee et al's paper.

Besides running the model architecture published by Jinhyuk Lee et al, we also tested the influence of replacing the hidden fully-connected layer with attention layer on the overall performance. Input feature wise, in addition to the three n-gram embeddings, we also tried to add a quadgram embedding. Results from the model will be discussed in the next section.

4 Results

We presented the results from the baseline model (softmax regression model), the LSTM model, and the LSTM model with attention layer. For each model, we ran trials with different hyperparameters to explore the best performer.

4.1 Baseline model results

For the softmax regression model, we ran 4 tests and the results are shown in Table 1. We chose a learning rate of 0.0001 for all the trials.

Table 1: Baseline model result summary.

Trials	Mini-batch size	Number of epochs	Number of layers	Layer dimensions	Training accuracy	Testing accuracy
Trial 1	512	1500	5	100,80,50,30,18	0.58	0.55
Trial 2	1024	5000	5	100,80,50,30,18	0.63	0.60
Trial 3	1024	10000	5	100,80,50,30,18	0.51	0.46
Trial 4	1024	10000	6	300,150,80,50,30,18	0.81	0.66
Trial 5	1024	10000	6	150,100,80,50,30,18	0.81	0.69

The best performing trial is Trial 5. It uses a mini-batch size of 1024 and ran 10000 epochs. The model has a total of 6 layers with 150, 100, 80, 50, 30, 18 units in each layer by order (the last layer corresponds to the softmax layer). The best training and testing accuracy achieved by the baseline model are 0.81 and 0.69 respectively.

4.2 LSTM model results

For the LSTM model, due to the fact that the model runs extremely slow (the original authors wrote the architecture using Tensorflow 1.0.1), we were forced to truncate the input data size by a factor of ten. Even then it took more than 20 minutes to run through 1 epoch. Because of this limitation, we changed the number of epochs to be 30 for every trial run. Table 2 summarizes some of the important hyperparameters we used for our model.

Table 2: Hyperparameter summary

Hyperparameters	Value
Training Epoch	30
Mini-batch size	256
Learning rate	0.0035
Decay rate	0.09
Gradient Clipping	[-5,5]
LSTM dropout probs	0.5
Hidden Layer dimension	200
Hidden layer dropout probs	0.5

Using the set hyperparameters, Loss for training, validation, and testing sets was calculated. Figure 2 in the Appendix shows the three learning curves for the model. Regarding accuracy, training, validation, and testing top 1 accuracy (The actual nationality matches the highest probability prediction) are 0.984, 0.802, and 0.832 respectively. To evaluate specific prediction performance, we generated a confusion matrix which is presented in the Appendix (Figure 3)

Attention algorithm is a widely used method for machine translation for its ability to focus on part of text for every word translation. We tried to incorporate the algorithm into the model, but the overall performance is not satisfying. The model ran normally for 10 epochs before the losses increase drastically (training loss goes from 0.446 to 77.579). For the 10th epoch, the top 1 accuracy for training, validation, and testing are 0.872, 0.69, 0.717 respectively.

When adding a quadgram embedding feature to our input (one extra LSTM layer), the model was able to run more epochs and achieved 0.999, 0.817, and 0.815 for training, validation, and testing respectively. However, after training through 9 epochs, the accuracy dropped dramatically. To compare two models with different n-gram feature, we attached a performance table in the Appendix for reference (Table 3)

5 Analysis

Compared to the baseline model, the LSTM model performs much better (Using training and testing accuracy as our metrics). This is expected since recurrent neural network (RNN) is known to have better performance in capturing features within sequence input. Due to the limitation of our computing power, we could not run the model through more iterations. Given the dataset and the current performance, it would be reasonable to estimate that the model can ultimately reach a testing accuracy higher than 90% with more epochs (thousands of epochs). Something to note from Figure 2 is that the training loss keeps decreasing while the validation and testing loss are increasing. This is common in model training and we expect both losses to decrease after running more epochs.

Using the confusion matrix acquired from our test run, we found that this LSTM model was good at recognizing vietnam names since its accuracy is 1. Chinese names had a very high precision and recall as well. As for the Japanese name, its recall was 1, but precision was below 1. Czech, English, and Polish names were confused with many other nationalities' names. It seems that English names were most confused with Scottish, French and German names. In addition, quite a few Indian names were recognized as Arabic. Overall,

names from East Asia were easier to distinguish.

The application of attention layer is not very helpful for our purpose as it slows down the model speed significantly and tends to incur gradient explosion. On the other hand, the addition of quadgram embedding increases the computation cost and the performance is relatively similar to the model without the additional embedding. However, we can see some positive impacts brought by the additional feature. Referring to Table 3, the influence of adding quadgram embedding is positive for most classes (increase of F1), except for Chinese, Russian, and Turkish name. As for recall and precision values, in most cases, adding quadgram embedding increases the model's prediction accuracy. From the overall record, we can clearly see that 4-gram model outperforms 3-gram model.

6 Discussion and Future Work

Overall, we can see that predicting a person's nationality from his/her name is relatively achievable using the model (if we can train the model through more epochs). From the results, we saw that RNN network performs better than the traditional neural network in dealing with sequence data. We concluded that attention mechanism might not be a good algorithm for our application and adding additional n-gram embedding feature can improve the model performance but at the cost of requiring more computing power.

Regarding future work, we should incorporate larger dataset with more countries to allow our model to fit wider range of users. In addition, we should consider using other algorithms such as hierarchical RNNs (extract higher and more complex representation, transformer.

7 Contribution

Wei Ren developed idea for quadgram and tuned the model, Jackie preprocessed the dataset and drafted the majority of the report, Jiate Li developed idea for attention and ran test trials.

8 Acknowledgement

Appreciate Professor Andrew Ng and Kian Katanforrosh for teaching this amazing course and teaching assistant Jonathan Li for answering our questions and holding office hours.

9 Appendix

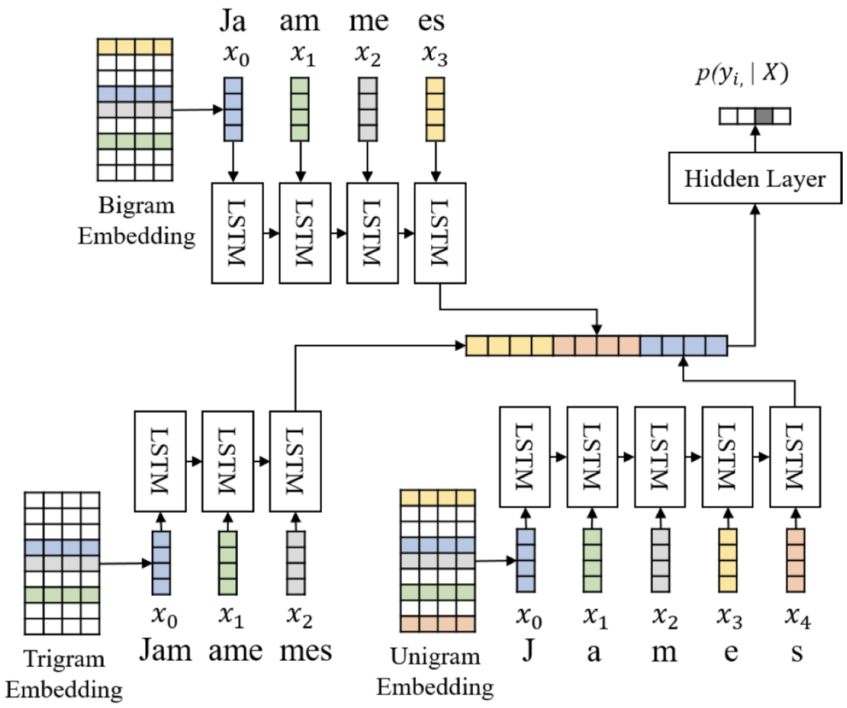

Figure 1: Workflow diagram for the RNN-LSTM model with character embedding

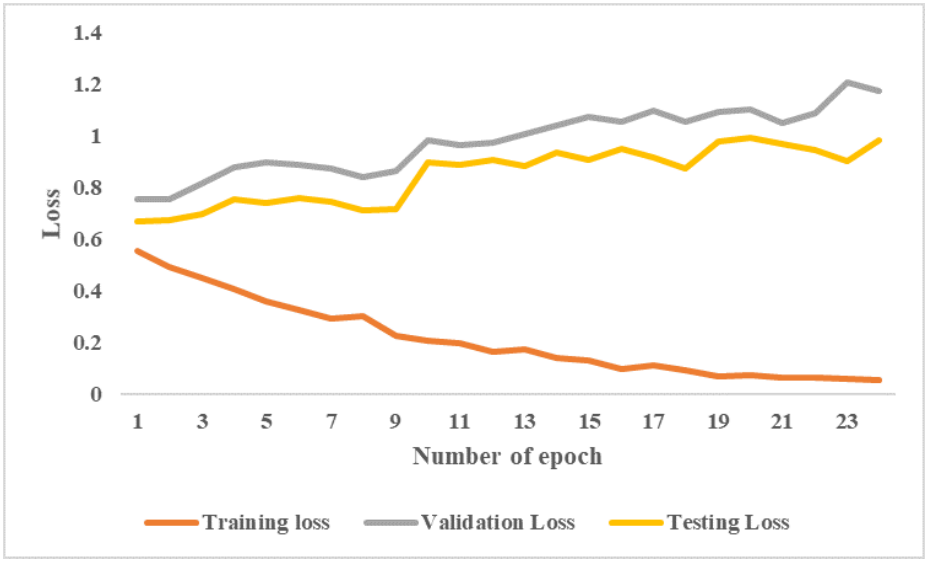

Figure 2: Learning curves for the LSTM model

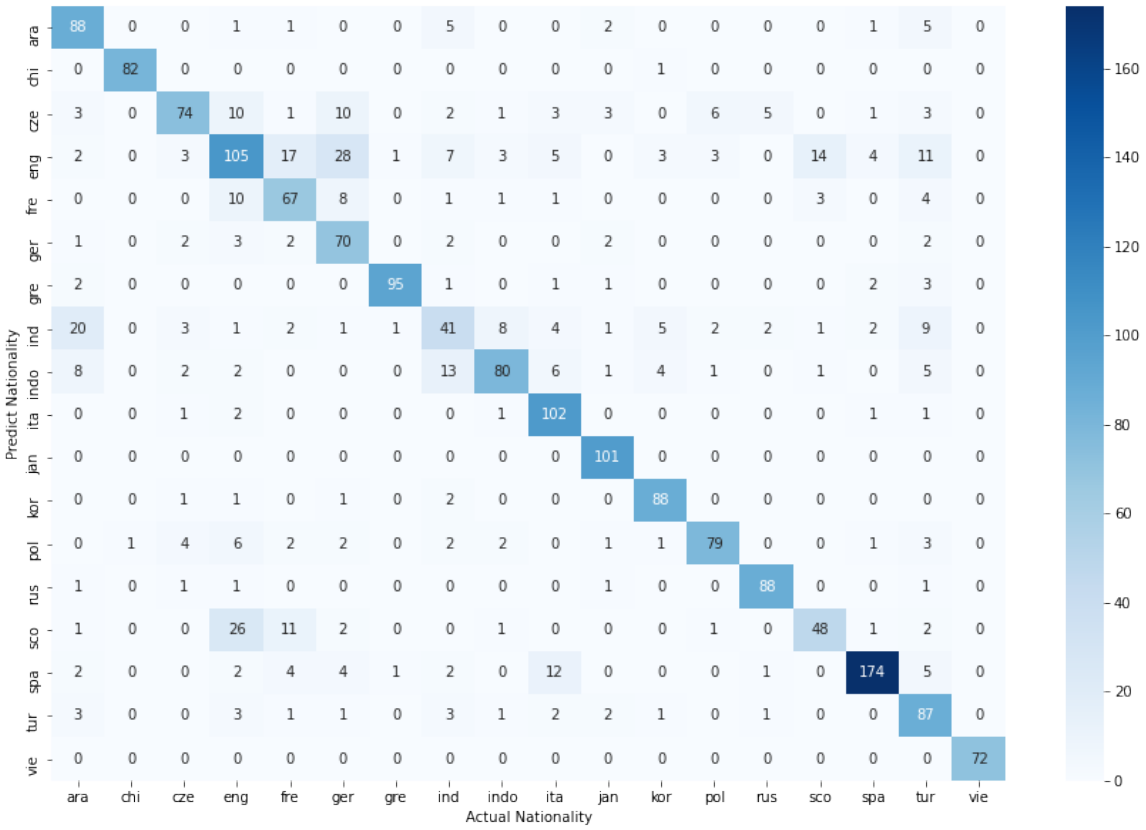

Figure 3: Confusion matrix

Table 3: Name Nationality Classification Analysis

	3-gram model				4-gram model			
	Count	F1	Precision	Recall	**Count**	F1	Precision	Recall
Arabic	103	0.75	0.67	0.85	208	0.83	0.8	0.86
Chinese	83	0.99	0.99	0.99	167	0.97	0.95	0.99
Czech	122	0.69	0.81	0.61	197	0.78	0.76	0.79
English	206	0.55	0.61	0.51	387	0.64	0.64	0.64
French	95	0.66	0.62	0.71	219	0.74	0.81	0.68
German	84	0.66	0.55	0.83	184	0.72	0.71	0.73
Greece	105	0.94	0.97	0.9	179	0.94	0.92	0.97
India	103	0.45	0.51	0.4	178	0.6	0.77	0.49
Indonesia	123	0.72	0.82	0.65	197	0.75	0.72	0.78
Italian	108	0.84	0.75	0.94	202	0.94	0.93	0.95
Japanese	101	0.94	0.88	1	193	0.95	0.96	0.94
Korean	93	0.9	0.85	0.95	189	0.93	0.94	0.92
Polish	104	0.81	0.86	0.76	181	0.82	0.75	0.91
Russian	93	0.93	0.91	0.95	213	0.92	0.91	0.92
Scottish	93	0.6	0.72	0.52	203	0.77	0.67	0.9
Spanish	207	0.88	0.93	0.84	401	0.91	0.93	0.88
Turkish	105	0.71	0.62	0.83	178	0.69	0.84	0.58
Vietnamese	72	1	1	1	158	1	0.99	1
Overall	2000	0.767	0.777	0.771	3834	0.82	0.827	0.823

References

Jinhyuk Lee, M. K. D. C. J. C. J. K., Hyunjae Kim. (2017, August). Name nationality classification with recurrent neural networks.. (Proceedings of the 26th International Joint Conference on Artificial Intelligence)

Pucktada Treeratpituk, C. L. G. (2012). Name-ethnicity classification and ethnicity-sensitive name matching. (in AAAI)

Tomas Mikolov, K. C. G. S. C. J. D., Ilya Sutskever. (2013). Distributed representations of words and phrases and their compositionality.. (in Advances in neural information processing systems, pages 3111-3119)

Tomas Mikolov, L. B. J. C. S. K., Martin Karafiat. (2010). Recurrent neural network based language model.. (In Inter-speech, volume 2, page 3, 2010)

Yoon Kim, D. S. A. M. R., Yacine Jernite. (2015). Character-aware neural language models.