HW4 Report

April 20, 2020

1 Homework 4 Report

Christopher W. Hong

1.1 Step 1:

- For reproducibility, I wrote a function `load_bert_repo` to download the BERT repository instead of downloading using `click and drop`. The repository was downloaded into *BERT_BASE_DIR*, `$HOME/Documents/repos/bert`.

1.2 Step 2:

- Same as Step1, I wrote a function `load_bert_model` to download the pre-trained BERT model, `BERT-Base, Uncased`, and uncompress it to `*BERT_DATA_DIR*`, `$BERT_BASE_DIR/models/uncased_L-12_H-768_A-12`. Upon completing the uncompressing, the compressed file was removed automatically.

1.3 Step 3:

- A function called `load_handout` was written to download the handout from the website, uncompressed it and removed the compressed file automatically. In this step, I also wrote a function `etl_text` that could extract `train.csv, eval.csv and test.csv` files into a pandas DataFrame from the `*HANDOUT_DATA_DIR*`, `$HOME/Documents/repos/handout/data`, transform and load the `text` column into a `.txt` file that contained each text on a single line into a folder called `bert_input_data`. The folder was in the same directory as the `.ipynb` file.

1.4 Step 4:

- A function called `run_bert_fv` was written to run the sh script `run_bert_fv.sh` located in the `*HANDOUT_DATA_DIR*`. In return, the sh script runned the `extract_feature.py` to generate feature vectors from `.txt` files generated in Step 3. Those generated feature vectors will be saved as `.jsonlines` files and were loaded into a folder called `bert_output_data` that was located in the same directory as folder `bert_input_data`.

1.5 Step 5:

- There are three sub steps in this part. First of all, I used the `extract_feature_vectors` function, which was a wrapper over the `.jsonlines` code provided in the handout, to extract feature vectors from `train, eval and test .jsonlines` that were generated in Step 4 and load them into a pandas DataFrame, respectively.

- Then, I loaded the original `train, eval and test .csv` dataset from the handout directory into a DataFrame, respectively. Grabbed the label column `native_language`, re-encoded classes within the column into integers so that it could feed into a machine learning model.
- In this final sub step, because we tried to solve a single-label, multi class machine learning problem, I trained a `Multinomial LogisticRegression` baseline model over 6,000 training samples.

1.6 Step 6:

- In this final step, I used the previous trained model to make predictions over the test dataset. Evaluattion metrics such as precision, recall, f1-score and accuracy score were evaluated over ground true and predicted labels. Plus, a confusion matrix was plotted into two heatmaps which one of them showed the discrete results and the other showed the percentages. Error analysis below were based on the metrics provided in **Figure 1-4**:
  - There were 2,000 test samples in total with **uniform class distribution** over 10 classes which were `Japanese, Korean, Vietnamese, Mandarin, Russian, Thai, Spanish, Cantonese, Polish, Arabic`. Thus, the accuracy score among many other metrics could be a reasonable measure over the model performance.
  - The model was heavily overfitting because there was big gap between the training accuracy score (65%) and the test one (47%). The performance was undesirable, but it was much higher than random guessing (10%).
  - Class Thai had the highest precision socre (66%) which meaned that in the predicted Thai classes, 66% of them were correctly predicted. Roughly 50% of the predicted classes including Korean, Russian, Spanish and Arabic were correctly predicted. Then, came the Japanese and Polish with 48% precision score. The model had the worst precision performance over classes including Vietnamese (41%), Mandarin (34%) and Cantonese (34%).
  - Around 60% of true Thai and Russian were predicted as true. Then, about 50% true positive rate was for Japanese, Spanish and Polish, respectively. Next, came Arabic and Korean with around (45%). Again, the model had the worst performance over Mandarin and Cantonese with the lowest true positive rate (35%).
  - Based on the combination of precision and recall score, Thai had the highest f1-score (63%) over other classes. Russian came the second (55%). Among the rest, Mandarin and Cantonese had lowest f1-score, 34% and 35%, respectively.
  - Base on the above metrics, the model had comparatively better performance on predicting Thai, Russian and Spanish. However, it did the poorest job at predicting Madarin, Cantonese and Vietnamese. It also implicitely showed that Thai and Russian native language writers had quite distinguished English wrtiting style from others. Yet, Madarin and Cantonese writers had quite the same English writing styles.
  - Around 12% (23 out of 200) of Japanese was misclassified as Korean while 10% Koean was misclassified as Japanese.

- Vietnamese was frequently misclassified (12%) as Madarin over other classes.
- Madarin was frquently misclassified (20%) as Cantonese over others and Cantonese was misclassified as Madarin (24%).
- Russian was frquently misclassified (12%) as Polish over others and 15% of Polish was misclassified as Russian.
- Thai was either mostly misclassified (6.5%) as Vietnamese or Cantonese.
- Spanish was either frequently misclassified (9%) as Vietnamese or Polish.
- Arabic was mostly misclassified (11%) as Spanish over others.
- The above pair-wised comparisons show that most writers whose countries were closed to each other had pretty similar English writing styles. Yet, Spanish native laguange writers were mostly regarded as Vietnamese or Polish. Arabic ones were mostly regared as Spanish.
- To sum up, location and cultural seemed playing a subtle role in non-English native language writers' English writing styles.
- To improve the overall perforamce and mitigate overfitting of the model, the most effective way is to gather more training data. Especially, we should try to gather more training data for the most misclassified classes Madarin, Cantonese and Vietnamese. Other approaches such as model regularization, model selection also help increaase the model performance.

Figure 1

train metrics

	precision	recall	f1-score	support
Japanese	0.66	0.71	0.69	600
Korean	0.65	0.63	0.64	600
Vietnamese	0.62	0.60	0.61	600
Mandarin	0.60	0.58	0.59	600
Russian	0.69	0.73	0.71	600
Thai	0.74	0.69	0.71	600
Spanish	0.64	0.67	0.66	600
Cantonese	0.56	0.58	0.57	600
Polish	0.69	0.66	0.68	600
Arabic	0.69	0.67	0.68	600
accuracy			0.65	6000
macro avg	0.65	0.65	0.65	6000
weighted avg	0.65	0.65	0.65	6000

eval metrics:

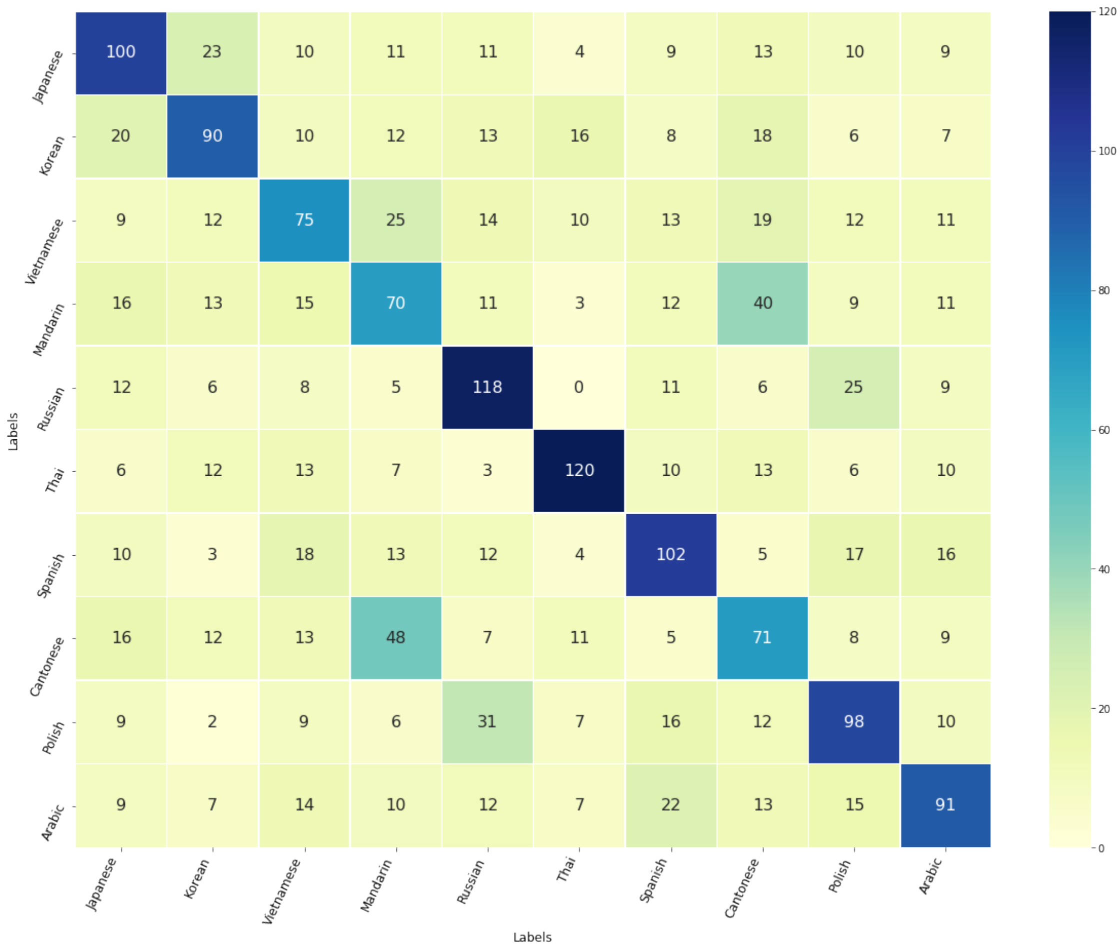
	precision	recall	f1-score	support
Japanese	0.51	0.56	0.53	200
Korean	0.50	0.47	0.48	200
Vietnamese	0.44	0.42	0.43	200
Mandarin	0.38	0.34	0.36	200
Russian	0.54	0.55	0.55	200
Thai	0.59	0.59	0.59	200
Spanish	0.52	0.58	0.55	200
Cantonese	0.37	0.40	0.38	200
Polish	0.52	0.52	0.52	200
Arabic	0.51	0.46	0.48	200
accuracy			0.49	2000
macro avg	0.49	0.49	0.49	2000
weighted avg	0.49	0.49	0.49	2000

Figure 2

test metrics:

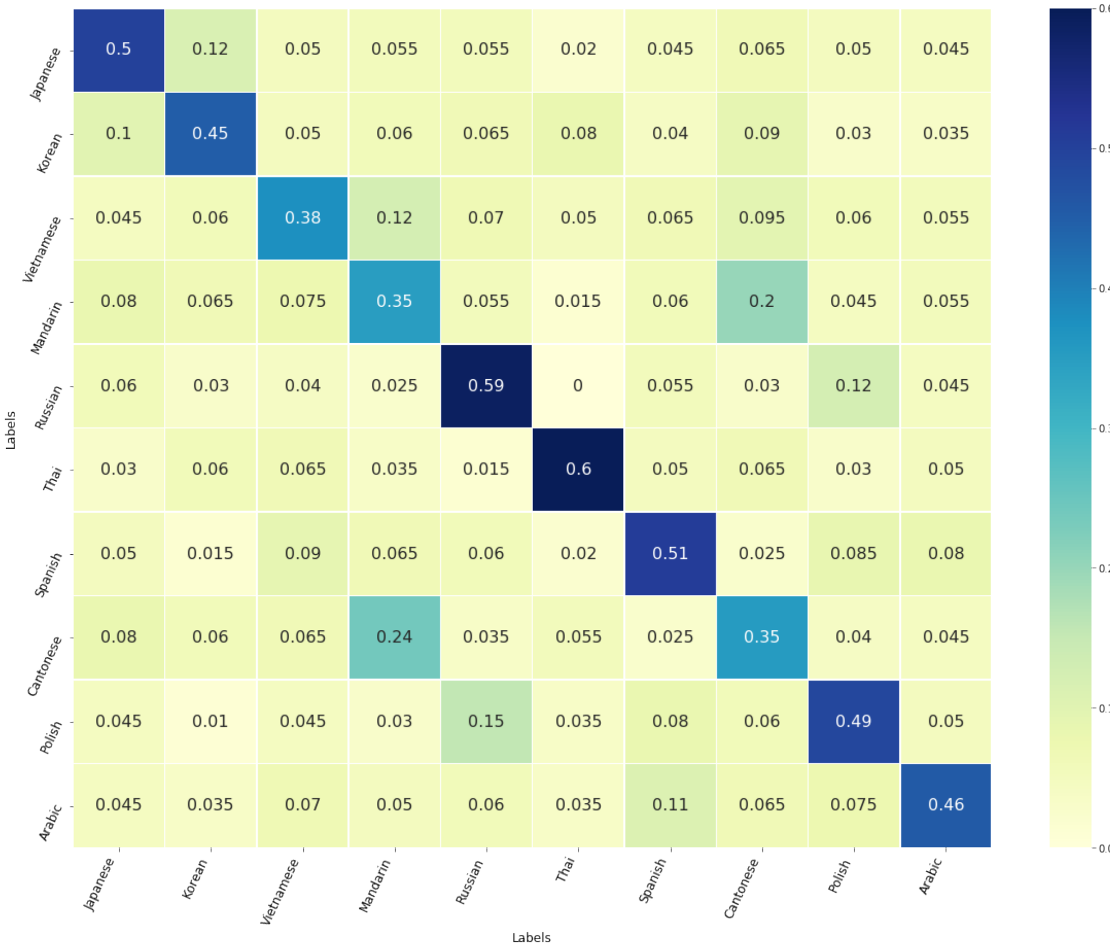
	precision	recall	f1-score	support
Japanese	0.48	0.50	0.49	200
Korean	0.50	0.45	0.47	200
Vietnamese	0.41	0.38	0.39	200
Mandarin	0.34	0.35	0.34	200
Russian	0.51	0.59	0.55	200
Thai	0.66	0.60	0.63	200
Spanish	0.49	0.51	0.50	200
Cantonese	0.34	0.35	0.35	200
Polish	0.48	0.49	0.48	200
Arabic	0.50	0.46	0.48	200
accuracy			0.47	2000
macro avg	0.47	0.47	0.47	2000
weighted avg	0.47	0.47	0.47	2000

Figure 3



Japanese
Korean
Vietnamese
Mandarin
Russian
Thai
Spanish
Cantonese
Polish
Arabic
Labels
100 23 10 11 11 4 9 13 10 9
20 90 10 12 13 16 8 18 6 7
9 12 75 25 14 10 13 19 12 11
16 13 15 70 11 3 12 40 9 11
12 6 8 5 118 0 11 6 25 9
6 12 13 7 3 120 10 13 6 10
10 3 18 13 12 4 102 5 17 16
16 12 13 48 7 11 5 71 8 9
9 2 9 6 31 7 16 12 98 10
9 7 14 10 12 7 22 13 15 91
120
100
80
60
40
20
0

Figure 4


Japanese
Korean
Vietnamese
Mandarin
Russian
Thai
Spanish
Cantonese
Polish
Arabic
Labels
0.5 0.12 0.05 0.055 0.055 0.02 0.045 0.065 0.05 0.045
0.1 0.45 0.05 0.06 0.065 0.08 0.04 0.09 0.03 0.035
0.045 0.06 0.38 0.12 0.07 0.05 0.065 0.095 0.06 0.055
0.08 0.065 0.075 0.35 0.055 0.015 0.06 0.2 0.045 0.055
0.06 0.03 0.04 0.025 0.59 0 0.055 0.03 0.12 0.045
0.03 0.06 0.065 0.035 0.015 0.6 0.05 0.065 0.03 0.05
0.05 0.015 0.09 0.065 0.06 0.02 0.51 0.025 0.085 0.08
0.08 0.06 0.065 0.24 0.035 0.055 0.025 0.35 0.04 0.045
0.045 0.01 0.045 0.03 0.15 0.035 0.08 0.06 0.49 0.05
0.045 0.035 0.07 0.05 0.06 0.035 0.11 0.065 0.075 0.46
Japanese
Korean
Vietnamese
Mandarin
Russian
Thai
Spanish
Cantonese
Polish
Arabic
Labels
0.6
0.5
0.4
0.3
0.2
0.1
0.0