COMP5046 Assignment 2 Report

Xing Xing[1] and Shuai Liu[2]

[1] University of Sydney, NSW 2006, Australia
xxin7882@uni.sydney.edu.au
sliu7343@uni.sydney.edu.au

1 Data preprocessing

This assignment is aimed to identify potential toxicity content in the in-game chat. The dataset in this assignment is part of the CONDA dataset from the University of Sydney NLP Group. This dataset consists of 26078 training data and labels, 8705 validation data and labels, and 500 testing data without labels. Model evaluation and performance testing will use the validation dataset and the testing dataset will only be used for Kaggle leaderboard and ranking. Some instances in the dataset are shown below:

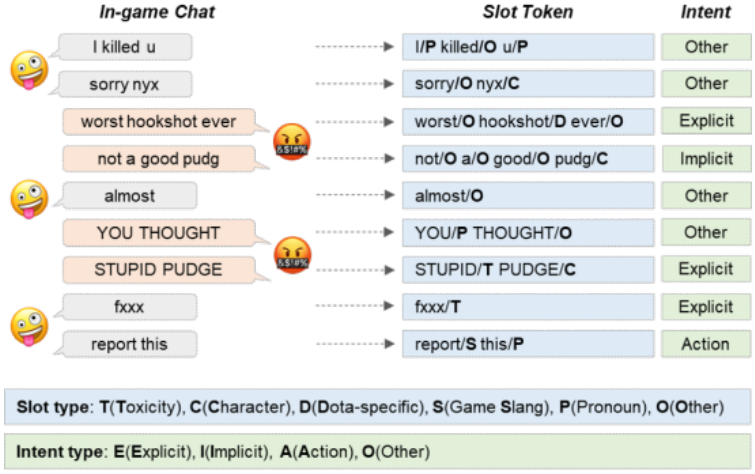

Figure 1. Some instances in the dataset

This assignment was required to be able to load data from google drive, however, 2 methods were provided for loading the data which are 'from GitHub' and 'from google drive' for potential preferences and convenience. If no modification applied, the load data from google drive will be set as default method which will fetch data from 3 documents:

- train.csv - the training data
- val.csv - the validation data
- test_without_labels.csv - testing dataset

1.1 Data example

The annotation corresponding to each sentence contains the annotation information of each word. 3 sample data and corresponding labels are shown below:

sents	labels
wow	O
cant believe it	O O P
dayuuuuuum [SEPA] lool	O SEPA O

Table 1. Data and label example

As shown in the above table, the label corresponding to each word of the sentence is a specific character, and the number of words in the input sentence is the same as the number of characters contained in the label. The distribution of labels is observed in the development duration as well, as shown in the histogram below. Meaningful annotation labels are divided into D, T, C, S, SEPA and P. Each class means that each slot class - T (Toxicity), S (game Slang), C (Character), D (Dota-specific), P (Pronoun), O (Other). The distribution is shown below:

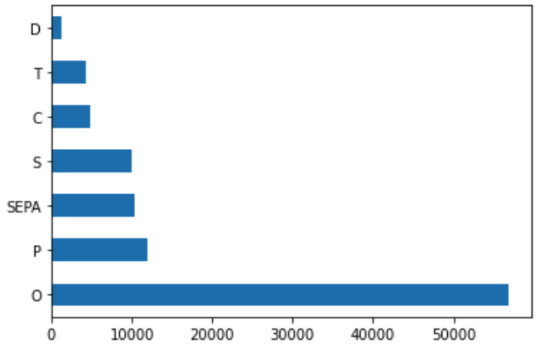

Figure 2. Label distribution

1.2 Data Preprocessing

In this assignment, a simple lower-cases tokenization method has been applied to the dataset. Since some special words were observed in the dataset such as '[SEPA]', other tokenization methods might not appropriately tokenize the data. For example, some default tokenizers will separate '[SEPA]' into '[', 'SEPA', ']' which will influence further experiment and further performance. Hence, in this assignment, sentences were simply divided into single words of lower-case using python default functions 'split()' and lower()'.

2 Input Embedding

2.1 Syntactic textual feature embedding

Since humans communicate complex ideas by combining words into larger units to convey complex meanings, we need to understand sentence structure to interpret language correctly. In this assignment, 2 syntactic textual aspect feature embedding methods were implemented namely PoS tagging and Dependency. PoS tagging refers to Part-of-speech tagging which is a basic intersection between linguistics and automatic text analysis (Pykes, 2020). PoS tagging aimed to extract the word attributes (Noun, Adjective, Verb, …) from the sentence which could demonstrate the syntactic function of a word and the possible words around it. PoS in this assignment was implemented using the "en_core_web_sm" model from spacy which is an efficient python NLP package. Like PoS, dependency path is another embedding method from another view of linguistic structure. Dependency path can explain the relationship between different units of a sentence, the same sentence may have a dependency structure, and different dependency structures may have significant semantic differences. In this assignment, the dependency path of words is also extracted using the Spacy package.

2.2 Semantic textual feature embedding

In this assignment, Word2Idx was implemented as the semantic textual feature embedding method. Word2Idx converts all unique words into a dictionary as the word pool for the model. We did not use common pre-trained word embedding models such as Glove and Word2Vec, because these models might not be able to deal with the words from in-game chat. For example, 'gg', 'AFK', 'FFS' are common abbreviations in an in-game conversation, but they are not common words outside the gaming environment. However, using word2idx could embed the words using our existing corpus which is more targeting to our task. Hence, instead of using pretrained models, we used 'word to index' to implement the semantic dimension for the data.

2.3 Domain feature embedding

In this assignment, word length has been considered as the domain feature embedding technique. In this dataset, there are a massive number of abbreviations that players used to communicate and express the information. Inspired by COMP5046 lecture 9 page 24 which introduced several embedding ideas and methods, we reckon that the word length might also be correlated with the toxicity and word abuse. Hence, we implemented a word length dictionary and assembled another dimensions in the dataset.

3 Slot Filling/Tagging model

We use the modified Bi-LSTM, which is an extension of the traditional LSTM and can improve model performance for sequence classification problems, as the training model to conduct ablation experiments. The new model is changed based on the baseline model, mainly changing the parameters and the processing methods corresponding to these parameters, which enables the new model to perform different processing on different stacked layers, attention methods, and whether CRF is used to obtain different results according to the input parameters.

3.1 Stacked Seq2Seq model

The sequence-to-sequence model is a method that can generate another sequence of non-fixed length by a specific method based on a given sequence of non-fixed length. The basic idea is to use two RNNs to process sequence information, one RNN as the encoder and the other RNN as the decoder.

LSTM is an improvement on RNN. It introduces a memory unit into each neuron in the hidden layer, and uses three gate control units: forget gate, input gate, and output gate to control the state of the memory unit, to solve the problem that ordinary RNN cannot learn long-distance timing dependence questions.

The model architecture of stacked-Bi-LSTM is to stack hidden layers, which can make the model work with higher dimensional data and be more accurate. For the determination of the number of stacked layers of the Bi-LSTM model, according to the research of Sun et al. in 2014, the number of stacked layers is not as good as possible, when the number of layers is small, the generalization of the model is poor, but for data with a small amount of data or low feature dimension, when the number of layers is large, under-fitting will occur. According to the experience of Wu et al. in 2016, in general, the number of stacked layers is not recommended to exceed 4 layers, and based on our dataset, the amount of data is relatively small and the complexity is relatively low. Therefore, we choose stacking layers of 1, 2, and 3 to conduct ablation experiments.

3.2 Attention

The LSTM mechanism has the problem of gradient disappearance. For long sentences, it is difficult to convert the input sequence into a fixed-length vector to save all valid information. As the length of the sentence increases, the effect of LSTM will decrease significantly. In order to solve the bottleneck of information loss, the Attention mechanism is introduced.

According to the calculation method of different attention weights mentioned by Caren in Lecture 10, we chose three of them, namely 'Dot-product', 'Scaled Dot-product' and 'General'. The 2017 article by Vaswaniden et al. pointed out that the dot product model can make better use of the matrix product algorithm in implementation, resulting in higher computational efficiency. However, when the hidden dimension of the input vector is relatively high, the value of the dot product model usually has a relatively large variance, resulting in a relatively small gradient of the softmax function. The scaled dot product model can solve this problem. The advantage of dot product or scaled dot product is that the calculation is simple, and the dot product calculation does not introduce additional parameters. The disadvantage is that the two matrices for calculating the attention score must be equal in size. The general calculation method solves this shortcoming by introducing an intermediate matrix Wa (trainable weight matrix). But it increases the cost of calculating the attention score.

Attention Name	Attention score function	Reference
Content-base	$score(s_t, h_i) = cosine[s_t, h_i]$	Graves 2014
Dot-product	$score(s_t, h_i) = s_t^\top h_i$	Luong 2015
Scaled Dot-product	$score(s_t, h_i) = \frac{s_t^\top h_i}{\sqrt{n}}$ *NOTE: very similar to the dot-product attention except for a scaling factor; where n is the dimension of the source hidden state.	Vaswani 2017
Additive	$score(s_t, h_i) = v_a^\top \tanh(W_a[s_t; h_i])$	Vaswani 2017
General	$score(s_t, h_i) = s_t^\top W_a h_i$ *NOTE: where W_a is a trainable weight matrix in the attention layer.	Luong 2015
Location-based	$a_{t,i} = softmax(W_a s_t)$ *Note: This simplifies the softmax alignment to only depend on the target position.	Luong 2015

Table 2: Different attention score (From Usyd NLP group, 2022)

4 Evaluation

4.1 Evaluation setup

As indicated by the assignment specification, the same optimizer, epoch, learning rate, and weight decay are used for all model training for more efficient performance comparison. Related parameters standard is shown below:

Parameter	**Value**
Optimizer	Optim.SGD
Learning rate	0.01
Weight decay	0.0001
Epoch	2

Table 3: Evaluation environment setup details

For the model evaluation result, a token F1 score is selected for evaluation. Token F1 score focuses on the prediction performance for slot tokens and calculates an F1 for each class and the token-based micro-averaged F1 score over all classes.

4.2 Evaluation between baseline and modified models

For the experiment at this stage, the performance of Baseline Bi-LSTM + CRF and Modified Bi-LSTM + CRF were generated using default parameters within 2 epochs. The default parameters used SGD with 0.01 learning rate on 0.0001 weight decay. The hidden dimension was set to 50 and the embedding dimension was set to 100 with 1 layer. Related performance comparison matrix is shown below:

	Metrics						
Model	T-F1	T-F1(T)	T-F1(S)	T-F1(C)	T-F1(D)	T-F1(P)	T-F1(O)
Baseline	0.9858	0.9319	0.9822	0.9476	0.7965	0.9934	0.9936
Modified	0.9868	0.9326	0.9822	0.9464	0.8015	0.9959	0.9947

Table 4: Baseline model comparison

In this section's experiment, baseline and modified models provided very similar overall results at 98.6% approximately. A potential explanation for this result could be that modified and baseline models have very similar structure, the only difference is that the modified Bi-LSTM + CRF support PoS, dependency, domain dimension extraction for sentences. However, in this experiment, the optional dimensions were not turned on for variables controlling which leads to very similar results on baseline and modified models with same default parameters.

4.3 Evaluation different input embedding

In this assignment, we implemented 4 types of input embedding from 3 perspectives which are Word2Idx, PoS, Dependency, and Word length. This section's experiment has tested 10 possible embedding combinations which aimed to find out the embedding (individual / combination) with the highest performance. Related experiment result is shown below:

	Model	T-F1	T-F1(T)	T-F1(S)	T-F1(C)	T-F1(D)	T-F1(P)	T-F1(O)
0	Word2Idx	0.986298	0.929204	0.983143	0.944546	0.793970	0.993394	0.994838
1	PosTag	0.759579	0.000000	0.162854	0.001828	0.000000	0.729167	0.966974
2	Dependency	0.707232	0.000000	0.034919	0.000000	0.000000	0.657774	0.931788
3	Word2Idx+PosTag	0.983150	0.920354	0.974413	0.939671	0.783920	0.990600	0.992731
4	Word2Idx+Dependency	0.982071	0.913547	0.972908	0.920780	0.728643	0.988313	0.994891
5	Word2Idx+Domain	0.985129	0.933288	0.983745	0.950030	0.783920	0.993394	0.992099
6	Word2Idx+PosTag+Dependency	0.975355	0.867937	0.950632	0.895186	0.660804	0.988313	0.994153
7	Word2Idx+Dep+Domain	0.981891	0.922396	0.970500	0.934186	0.758794	0.986789	0.992836
8	Word2Idx+Pos+Domain	0.985129	0.930565	0.977122	0.934186	0.788945	0.990346	0.995365
9	Word2Idx+Pos+Dep+Domain	0.977724	0.913547	0.969597	0.906155	0.711055	0.988313	0.989465

Table 5: Different input embedding method evaluation

During the long period of evaluation, Word2Idx and Domain related combinations provided very decent results. Word2Idx provided the highest performance at 98.6% F1 score on validation dataset. Word2Idx + Domain and Word2Idx + Pos + Domain provided about 98.5% F1 score which is slightly lower than Word2Idx only. However, the PoS tagging and Dependency path performed a much lower accuracy on the validation data. PosTag and Dependency provided 2 lowest performance at 71% and 76%. Interestingly , the Word2Idx + PoS + Dep + Domain as an all-together combination only provided a 97% F1 score which is the 3rd lowest performance among 10 combinations. Compared with Word2Idx + Domain, additional PoS and Dependency decrease the overall performance by 2%. Hence, it is appropriate to say that Domain and Word2Idx are more suitable embedding methods for this specific dataset. According to the experiment, it is obvious to find out that Word2Idx and Word length provided very stable performance and efficiency on in-game chat dataset. The reason why Word2Idx provides a valid embedding dimension is that Word2Idx has a relatively large corpus that contains all unique words in the dataset, which makes the model robust and can adapt to this specific task. On the other hand, word length could also be considered as an effective domain dimension. That means the labels and word length were correlated to some degree. As discussed earlier, the sentences in this dataset refers to the in-game chat and conversations with significant portions of abbreviations. Hence, part of speech and dependency might not be effective enough for abbreviations and one word sentences. For example, ‘gg’ and ‘AFK’ are very common sentences in the dataset which represent ‘good game’ and ‘away from keyboard’ respectively. In this kind of data, PoS and dependency paths might provide poor performance.

4.4 Evaluate different attention strategy

	Attention type	T-F1	T-F1(T)	T-F1(S)	T-F1(C)	T-F1(D)	T-F1(P)	T-F1(O)
0	Dot-product	0.982971	0.925800	0.976821	0.934796	0.776382	0.991108	0.992099
1	Scaled Dot-product	0.981891	0.916950	0.971403	0.920171	0.786432	0.988821	0.993311
2	General	0.982791	0.927842	0.979229	0.942718	0.758794	0.991362	0.990782

Table 6: Different attention strategy performance comparison

It can be seen from the results obtained by the calculation methods of different attention weights that the difference between the three different calculation results is not large, and the better one is to use the calculation method of dot-product, but these results are down compared to the baseline model (Attention = None). A possible explanation for the small difference between the models is that attention is more helpful for datasets with relatively long sentences. We use the in-game dialogue, and the length of each sentence is relatively short. We checked the average length of sentences in the training set. The average length of the sentences is 3.8, which means that each sentence contains only four words on average. Therefore, compared to the baseline model, Attention has little help for the model, and will also cause the f1-score to drop and increase the amount of computation. So, we decided to set Attention to None for training this dataset.

4.5 Evaluate different stacked layers

	Number of layers	T-F1	T-F1(T)	T-F1(S)	T-F1(C)	T-F1(D)	T-F1(P)	T-F1(O)
0	1	0.986628	0.942138	0.985551	0.943937	0.806533	0.993140	0.993837
1	2	0.984350	0.923758	0.978928	0.939671	0.809045	0.992124	0.992942
2	3	0.972717	0.893805	0.937688	0.895795	0.462312	0.988821	0.993837

Table 7: Different stacked layers performance comparison

From the results table, when the number of stacked layers is 1, the model performs best, and the value of f1-score is the largest, which is 0.986628. The main reason is that the amount of our data is relatively small, and it is communication sentences in the game, there are no very long sentences, and the overall complexity of the data is not high. According to Sun et al. in 2014, for our dataset, the higher the number of stacked layers, the more prone to underfitting. To sum up, we found that for our dataset, setting the number of stacked layers to 1 is the most reasonable.

4.6 Evaluate with/without CRF model

	With CRF	T-F1	T-F1(T)	T-F1(S)	T-F1(C)	T-F1(D)	T-F1(P)	T-F1(O)
0	True	0.985729	0.929204	0.982842	0.943937	0.796482	0.993140	0.993943
1	False	0.984200	0.938734	0.983444	0.945155	0.776382	0.995681	0.990203

Table 8: Different CRF performance comparison

Comparing the results, we can find that the model with CRF performs better, with an f1-score of 0.985729, which is expected, since CRF does help to improve the accuracy of labeling (Huang, Xu and Yu, 2015), and according to the principle of the CRF, our dataset is real-life dialogue, even if there are a lot of abbreviations, game slang and game-specific nouns, but there are contextual and logical relationships. Therefore, using Bi-LSTM with CRF is a better choice.

5 Conclusion

5.1 Evaluate best model

In this assignment, after several experiment and comparison, the best parameters has determined as:

- Hidden dimension = 50,
- Embedding dimension = 100,
- Number of layers = 1,
- PoS dimension = 0,
- Dependency dimension = 0,
- Domain dimension = 0,
- Attention = None,
- WithCRF = True

The model with all best-performed parameters demonstrated 98% overall f1 score on the dataset. Related result and performance F1 score are shown below:

	Model	T-F1	T-F1(T)	T-F1(S)	T-F1(C)	T-F1(D)	T-F1(P)	T-F1(O)
0	Best Model	0.986418	0.936011	0.983143	0.948202	0.821608	0.99314	0.993679

Table 9: Best model performance

5.2 Summary and future work

Through our modification of the Bi-LSTM model, we tested the performance analysis of the model under different stacked layers, different attention strategies and whether with CRF layers. And concluded that the Bi-LSTM model with CRF, stacked layer is 1 and without Attention, training on the Dota dialogue dataset works best. According to our modified Bi-LSTM model, we can further analyze the toxicity of the

dialogue in the Dota game, that is, the Intent type of the sentence can be further determined according to the Slot token obtained by the our model.

References

1. Huang, Z., Xu, W. and Yu, K., 2015. Bidirectional LSTM-CRF models for sequence tagging. arXiv preprint arXiv:1508.01991.
2. Pykes, K., 2020. Part Of Speech Tagging for Beginners. [online] Medium. Available at: https://towardsdatascience.com/part-of-speech-tagging-for-beginners-3a0754b2ebba [Accessed 4 June 2022].
3. Ronran, C. and Lee, S., 2020, February. Effect of character and word features in bidirectional lstm-crf for ner. In 2020 IEEE International Conference on Big Data and Smart Computing (BigComp) (pp. 613-616). IEEE.
4. Sun, L., Wang, Y., He, J., Li, H., Peng, D. and Wang, Y., 2020. A stacked LSTM for atrial fibrillation prediction based on multivariate ECGs. Health information science and systems, 8(1), pp.1-7.
5. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł. and Polosukhin, I., 2017. Attention is all you need. Advances in neural information processing systems, 30.
6. Wang, K., Lu, D., Han, S.C., Long, S. and Poon, J., 2020. Detect all abuse! toward universal abusive language detection models. arXiv preprint arXiv:2010.03776.
7. Weld, H., Huang, G., Lee, J., Zhang, T., Wang, K., Guo, X., Long, S., Poon, J. and Han, S.C., 2021. CONDA: a CONtextual Dual-Annotated dataset for in-game toxicity understanding and detection. arXiv preprint arXiv:2106.06213.
8. Wu, Y., Schuster, M., Chen, Z., Le, Q.V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K. and Klingner, J., 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. arXiv preprint arXiv:1609.08144.