Classifying An Imbalanced Dataset with CNN, RNN, and LSTM[1]

Catherine Xin Xin Yu
Matricola: 917114

Abstract

With the outbreak of the COVID-19 pandemic, anti-Asian speech is more frequently seen on social media platforms, and attempts have been made to detect and classify this type of hateful or offensive speech using machine learning methods. The purpose of the current study is to train and optimise CNN, RNN, and LSTM models to classify the East-Asian Prejudice Dataset (Vidgen et al., 2020), compare their performance, and practice using essential NLP and ML resources and libraries, including Numpy, Pandas, NLTK, SpaCy, ScikitLearn, Gensim, GloVe, FastText, Keras, Optuna, and Imbalanced-Learn. This study finds that the best performance can be achieved by using a CNN model with GloVe Twitter 200D embeddings, and the most effective way to address class imbalance in this dataset is to oversample minority classes during training.

1. Introduction and Literature Overview

1.1 Hate Speech and Offensive Language Detection in NLP

Hate speech and offensive language detection is a supervised classification task that has gained considerable attention in the field of natural language processing (NLP) in recent years, as shown by the large number of relevant publications and conference tasks (see MacAvaney et al., 2019). The correct classification of hateful or offensive speech is important for detecting, monitoring and reducing cyber-aggression, without unnecessarily or mistakenly infringing upon online freedom of speech. It can also reduce the workload for human moderators and increase coverage and efficiency, especially on social media platforms like Facebook, Twitter and forums where the amount of user-generated content is tremendous.

There are various existing approaches to hateful and offensive language detection/classification. The Keyword-based method (such as using the Hatebase lexicon) is a rudimentary approach that tends to result in a large amount of misclassification, sinch it cannot take context into account (e.g. predicting false positives due to non-offensive uses of swear words in colloquial language, or false negatives due to hostile remarks that do not use offensive words, such as praising a genocide). A more sophisticated method is to use machine learning models to classify texts, either non-neural architectures like Naive Bayes, logistic regression, support vector machines, etc., or state-of-the-art neural networks that are increasingly popular, like convolutional neural network (CNN), recurrent neural networks (RNN), long-short term memory networks (LSTM), bidirectional encoder representations from transformers (BERT) and neural ensembles. Data representation methods have also become more sophisticated through the years, evolving from bag-of-word (BOW) approaches and TF-IDF representation to contextual embeddings that can capture meaning and context. Indeed, in recent hateful and offensive language classification tasks such as HatEval 2019 (Basile et al., 2019) and OffensEval 2019 (Zampieri et al., 2019b), the majority of participants used deep neural networks with contextual embeddings.

There is quiet a number of existing datasets that can be used to train hateful and offensive language classifiers, including Hatebase Twitter (Davidson et al., 2017), Waseem and Hovy (2016), Waseem (Waseem 2016), Stormfront (de Gibert et al., 2018), TRAC (Kumar et al., 2018), HatEval (Basile et al., 2019), GermanTwitter (Ross et al., 2016), and Offensive Language Identification Dataset (OLID) (Zampieri et al., 2019). However, they generally focus on racist, sexist, and homophobic language,

[1] All codes used for this project can be found here (exam_submission branch): https://github.com/CathxxYU/CoLi-project/tree/exam_submission/codes_v2

targeting groups like people of colour, immigrants and refugees, women, and members of the LGBTQ community (MacAvaney et al., 2019).

1.2 COVID-19 and Anti-Asian Hate on Social Media

Hate speech and offensive language associated with the COVID-19 pandemic involve a new lexicon (e.g. medical terms, place names, Anti-Asian slurs) and a new set of targets (e.g. East Asian governments, organisations, and individuals) that are not covered by previous datasets and classifiers, which is why it might be useful to develop new resources and models. There are already some papers published in mid-2020 that examine online hateful or offensive speech related to COVID-19. Schild et al. (2020) compared Sinophobic cyber behaviour on different platforms by training contextual embeddings using two dataset, one composed of tweets and the other composed of posts from 4chan's Politically Incorrect Board. The authors found that new racist slurs emerged after the outbreak of COVID-19 and the meaning of some existing words shifted (e.g. an increase in cosine similarity between 'chinese' and 'virus'), and users of different online platforms used racist slurs differently. Unfortunately, since the authors did not share the actual embeddings, they cannot be used to train classifiers. Ziems et al. (2020) compiled the COVID-HATE dataset, composed of 2400 English tweets divided into three classes: hate, counterhate, and neutral. However, this dataset is rather small and the annotation not very fine-grained, and it is mainly used to train a classifier to automatically label a much larger dataset to allow the authors to analyse the role of user-interaction in the creation of hateful content ('social contagion'). Last but not least, Vidgen et al. (2020) compiled a dataset that is much larger and more finely annotated. This is the dataset used in the present study.

2. Dataset Description

The dataset presented in Vidgen et al. (2020) contains 20,000 English tweets collected between 1 Jan, 2020 and 17 March, 2020. The tweets are divided into the following five classes, listed in descending order according to the number of samples per class.

- **neutral** (67.6%, or 13,528 tweets): tweets that not fall into any of the other categories
- **hostility against an East Asian entity** (19.5%, or 3,898 tweets): 'Tweets which express abuse or intense negativity against an East Asian entity, primarily by derogating or attacking them' (Vidgen et al., 2020). It is distinguished from the 'criticism' class by the presence of 1) *direct attacks* against EA people, or 2) critical statements that are '*framed normatively*' (Vidgen et al., 2020).
- **criticism of an East Asian entity** (7.2%, or 1,433 tweets): 'Tweets which make a negative judgment about an East Asian entity, without crossing the line into abuse.' (Vidgen et al., 2020) As argued by the authors of the dataset, distinguishing between 'hostility' and criticism' is particularly important for online freedom of speech, since the former should be censored while the latter should be allowed. However, the difference between the definitions of these two classes is very subtle, and is indeed a major source of disagreement in annotation and error in classification. As the results below will show, this is the most difficult class to classify.
- **discussion of East Asian prejudice** (5.1%, or 1,029 tweets) & **counter speech** (0.6%, or 116 tweets): these two classes are combined during the classification task due to the similarity of the content and the small number of samples in both classes.
    - **discussion of EA prejudice**: 'Tweets that discuss prejudice related to East Asians but do not engage in, or counter, that prejudice' (Vidgen et al., 2020)
    - **counter speech**: 'Tweets which explicitly challenge or condemn abuse against an East Asian entity' (Vidgen et al., 2020)

There are no duplicate tweets in the dataset. There is no indication of any tweet-retweet-response relationships between tweets, nor the precise date of each tweet. The annotation was done manually by two trained annotators and one expert to adjudicate in case of disagreement. They also annotated secondary categories and additional flags, but these are not relevant for the present classification task.

Vidgen et al. experimented with seven models: LSTM, AlBERT(xlarge), BART(large), BERT(large), DistilBERT(base), RoBERTa(large), XLNet(large), and evaluated them using F1 score (best: RoBERTa, 0.83), Recall (best: RoBERTa: 0.83), and Precision (best: LSTM, 0.88). They preprocessed the text by removing URLs and usernames, lowercasing, and using hashtag replacements (Vidgen et al. 2020: 'with either a generic hashtag-token or with the appropriate thematic hashtag-token'). The representation methods used were one-hot vectors for the baseline LSTM model, and contextual embeddings for the rest of the models. They did not use CNN or RNN, which are among the models tested in this project, and they did not use contextual embeddings with their LSTM model.

3. Experiments

3.1 Preprocessing & vectorisation

Target label preprocessing: Following Vidgen et al.'s practice, the two smallest minority classes ('counter speech' and 'discussion of EA prejudice') have been combined, producing a dataset with 4 classes only. The labels are then one-hot encoded using Scikit Learn's LabelBinarizer.

Preprocessing: In keeping with Vidgen et al.'s preprocessing, I also performed minimal preprocessing in this project. URLs, user IDs, and specific tweet-related words such as 'RT' were removed. Lowercasing was performed in order to reduce the size of the vocabulary, but lemmatization and stop-word removal were not performed, in the hope that the neural models can learn from the syntax of tweets and the inflected forms of words. Hashtags and the hash sign were not modified or removed either, since FastText subword embeddings should be able to compute meaningful embeddings for tokens with sub-components, including hashtags composed of multiple words.

Tokenization: **SpaCy** (English Core Web Large model), an up-to-date NLP python library that uses deep-learning models to tokenize texts, was used to tokenize tweets for this project. Before settling on SpaCy, I experimented with **NLTK TweetTokenizer** and **NLTK TreeBankTokenizer**, but both were problematic. The former was not able to tokenize contracted words like negatives and possessives, and 14% of the tokens became out-of-vocabulary (OOV) tokens in the subsequent vectorization step using GloVe embeddings, most of them being common words. With TreeBankTokenizer, 18% of the tokens were OOV, perhaps because this tokenizer recognizes limited vocabulary and is not suitable for dealing with Twitter language. With SpaCy and GloVe, OOV tokens mostly consist of neologisms, hashtags, misspelled words, numbers, and special characters (see Appendix IV. SpaCy + GloVe Out-of-vocabulary Words).

Representation: Two types of contextual embeddings were used: pretrained **GloVe Twitter 200-dimension** embeddings, and pretrained **FastText subword 300-dimension** embeddings. The former was chosen because it was trained on a corpus of tweets, while the latter was chosen because it can cope with inflections and rare words by breaking tokens down to subwords. In fact, there are no OOV tokens when samples are represented with FastText subword embeddings.

Splitting the dataset: the dataset was trained and tested using a **stratified 80/10/10** train/validation/test split (following Vidgen et al., 2020).

3.2 Models

CNN (implemented using Keras) was chosen due to its recent popularity in similar NLP tasks, and the fact that Vidgen et al. did not experiment with this type of model.

Bidirectional RNN (implemented using Keras) was chosen for its ability to learn long-distance dependencies and remember sample-wide context. Vidgen et al. also did not experiment with this model.

LSTM (implemented using Keras) was chosen for its ability to select relevant long-distance dependencies and sample-wide context. Its gated architecture also offers an advantage over RNN since it is less prone to run into the problem of vanishing or exploding weights in long data samples.

Table 1. Model Overview

	CNN	BiRNN	LSTM
Architecture	● Conv1D (padding='same') ● GlobalMaxPooling1D ● hidden Dense layer ● Dropout layer ● output Dense layer	● Bidirectional Simple RNN ● Dropout layer ● Flatten layer ● output Dense layer	● LSTM ● Dropout layer ● Flatten layer ● output Dense layer
Training	● loss: categorical crossenthropy ● optimizer: Adam ● metric: precision, recall		

3.3 Addressing Class Imbalance

Since the dataset was significantly imbalanced, I experimented with four ways to address class imbalance, following the methods discussed in Brownlee (2020).

I. **Baseline**: not address the class imbalance (no alteration to the training set, no class-weight used during training).

II. **Weighted neural network**: use the class_weight argument in Keras's model.fit(). The weight of the majority class is set to 1, while the weight of each minority class is the inverse of the imbalance of that class with respect to the majority class (e.g. if that class has ⅓ of the number of samples in the majority class, then the weight of this class is set to 3)

III. **Oversampling** (applied to the training set only): randomly duplicate all minority classes to match the number of samples in the majority class using the RandomOverSampler() method of the Imbalanced-Learn library. This was chosen instead of a generative method (such as SMOTE) and a selective method (such as varieties of borderline-SMOTE) to avoid generating data that are not 'real tweets' or skewing towards samples located in specific areas of the class distribution.

IV. **Combination of oversampling and undersampling** (applied to the training set only): randomly delete majority class samples by half, using the RandomUnderSampler() method of the Imbalanced-Learn library, then randomly oversample all minority classes to match the number of samples in the halved majority class in the same way as described in method III. Random undersampling was used instead of a selective method in order to avoid skewing towards samples located in specific areas of the class distribution (such as deleting samples that are in the centre of the majority class distribution using Condensed Nearest Neighbour undersampling, or deleting samples that are close to the decision boundary between classes using Tomek Link cross-class nearest neighbours or Edited nearest-neighbour rule undersampling). This methods is chosen because it allows samples in minority classes to be duplicated fewer times, therefore less likely to cause the model to overfit the minority classes.

3.4 Hyperparameter optimisation

Hyperparameters are optimised by taking the best of 100 trials for each type of model, representation, and maxlen. It was performed using **Optuna**, a library for Bayesian optimisation published in 2019 (Akiba et al., 2019). A popular (and older) alternative is HyperOpt, but Optuna was chosen because the implementation is more intuitive and the API and tutorials more user-friendly. The

hyperparameters and the values searched are indicated in Table 2 below (see also Appendices I, II, and III for all models, their hyperparameters, and performance scores).

Table 2. Hyperparameters Searched Using Optuna

CNN	BiRNN	LSTM
■ **filters**: 100 – 1000 ■ **kernel_size**: 3, 4, 5 ■ **strides**: 1, 2 ■ **hidden_dims**: 20 –1000 ■ **dropout**: between 0.2 – 0.5 ■ **lr**: between 1e-4 – 1 ■ **batch_size**: 32, 64,128, 256, 512	■ **num_neurons**: 20 – 1000 (RNN or LSTM layer) ■ **dropout**: 0.2 – 0.5 ■ **lr**: between 1e-4 – 1 ■ **batch_size**: 32, 64, 128, 256, 512	

Two other hyperparameters were not optimised using Optuna. The number of training **epochs** is set at 200 with an **early stopping** regime (patience=1, restore_best_weights=True). In practice, CNN models took only 3-9 epochs to train, and BiRNN and LSTM models took only 2-10 epochs to train.

The **maxlen** hyperparameter cannot be optimised in Optuna since it is part of the preprocessing. I experimented with the mean, max, and the number of tokens in the majority of tweets (called 'majority' maxlen below; see Figures 1 & 2, or here for HD images). When represented with **GloVe** embeddings, samples have an average of **30** tokens/tweet, maximum **86** tokens/tweet, and the majority has **≤61** tokens/tweet. When represented with **FastText** subword embeddings, samples have an average of **36** tokens/tweet, maximum **113** tokens/tweet, and the majority has **≤72** tokens/tweet. Data represented with GloVe embeddings are generally shorter because OOV tokens are dropped. (N.B. the maximum length per tweet had been extended to 280 characters when this dataset was compiled, and the fact that most samples have ≤72 tokens fits the average estimate that there are about 4 characters/word in the English language. Particularly long samples tend to contain many punctuation marks or emojis, which are considered individual tokens by SpaCy.)

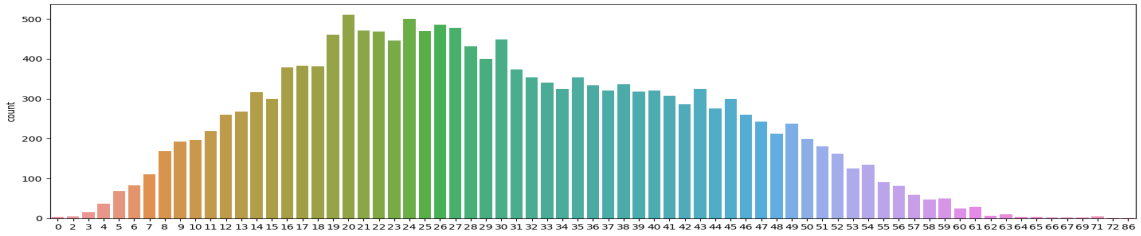

Figure 1. Count Plot for Samples Tokenized and Vectorized with SpaCy and GloVe

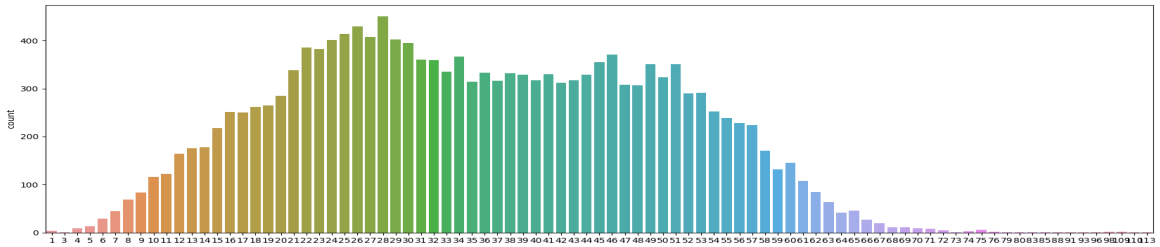

Figure 2. Count Plot for Samples Tokenized and Vectorized with SpaCy and FastText

3.5 Reproducibility

In order to make all the experiments repeatable, the random seed was generally set as 9 when randomness was involved (numpy random seed, tensorflow random seed, sklearn train_test_split random_state, Imblearn RandomOverSampler and RandomUnderSampler random_state, Optuna TPESampler), except for Optuna trials for RNN models (in this case TRESampler=99 was used; see section 4.1 for explanations).

4. Results & Analysis

Table 3. Summary of Best Models

Model Representation	**Model Name** method of addressing imbalance (**red**: best model)	**Performance on test set**						
		F1		Precision		Recall		Loss
		macro average	weighted average	macro average	weighted average	macro average	weighted average	
		note: **underline** = best score in its type of model; **red** = best score overall						
CNN GloVe	**CNN_gv_model2a** none	0.61	0.79	**0.73**	**0.8**	0.58	**0.81**	**0.4892329276**
	CNN_gv_model2a_II class-weighted training	0.59	0.75	0.56	0.77	0.64	0.73	0.7216849923
	CNN_gv_model2a_III oversampling	**0.66**	**0.8**	0.66	**0.8**	0.66	**0.81**	0.5153403878
	CNN_gv_model2a_IV under & oversampling	0.62	0.78	0.62	**0.8**	0.65	0.78	0.5650983453
CNN FastText	**CNN_ft_model2a** none	0.56	0.78	0.67	0.79	0.56	**0.81**	0.5148883462
	CNN_ft_model2a_II class-weighted training	0.54	0.73	0.54	0.77	0.61	0.72	0.7382657528
	CNN_ft_model2a_III oversampling	0.62	0.77	0.59	**0.8**	0.66	0.76	0.6265294552
	CNN_ft_model2a_IV under & oversampling	0.63	0.79	0.61	**0.8**	**0.67**	0.78	0.5739517212
RNN GloVe	**RNN_gv_model1b** none	0.5	**0.74**	0.58	**0.73**	0.49	**0.77**	0.6164960861
	RNN_gv_model1b_II class-weighted training	0.43	0.66	0.4	0.72	0.54	0.66	2.359926224
	RNN_gv_model1b_III oversampling	0.45	0.62	0.45	0.71	0.53	0.57	3.349912167
	RNN_gv_model1b_IV under & oversampling	0.27	0.33	0.38	0.71	0.44	0.28	6.478302956
	RNN_gv_model1[2] none	**0.52**	**0.74**	**0.59**	**0.73**	0.51	**0.77**	**0.61362046**
RNN FastText	**RNN_ft_model1** none	0.51	0.72	0.55	0.72	0.49	0.75	0.6331651211
	RNN_ft_model1_II class-weighted training	0.38	0.6	0.39	0.67	0.46	0.57	1.060679436
	RNN_ft_model1_III oversampling	0.48	0.68	0.47	0.72	0.54	0.66	0.8945197463
	RNN_ft_model1_IV under & oversampling	0.47	0.62	0.46	**0.73**	**0.55**	0.58	1.033476353
LSTM GloVe	**LSTM_gv_model1a** none	0.57	0.77	**0.64**	0.77	0.56	**0.79**	0.5483127236
	LSTM_gv_model1a_II class-weighted training	0.57	0.74	0.54	0.78	0.63	0.72	0.7817038298
	LSTM_gv_model1a_III oversampling	0.62	0.78	0.59	**0.79**	0.65	0.77	0.6496658325
	LSTM_gv_model1a_IV under & oversampling	0.59	0.75	0.55	**0.79**	**0.66**	0.73	0.7324165702
LSTM FastText	**LSTM_ft_model1b** none	0.58	**0.79**	**0.69**	**0.79**	0.57	**0.81**	**0.5275803804**
	LSTM_ft_model1b_II class-weighted training	0.62	0.78	0.6	**0.79**	0.65	0.77	0.6376052499
	LSTM_ft_model1b_III oversampling	**0.63**	0.78	0.61	**0.79**	**0.67**	0.77	0.6786770225
	LSTM_ft_model1b_III under & oversampling	0.57	0.72	0.54	**0.79**	0.66	0.69	0.8296226263

[2] I did not use this model as a baseline model for experimentations with ways to address imbalance, since it is not the RNN-GloVe model with the lowest validation loss, but it turned out to perform the best on the test set.

Table 4. Per-Class Performance of the Best Model: CNN_gv_model2a_III

	F1	Precision	Recall
Criticism	0.40	0.46	0.36
Discussion_Counter	0.67	0.63	0.71
Hostility	0.67	0.65	0.70
Neutral	0.90	0.90	0.89

4.1 Analysis of Models

Various experiments were conducted to find the best representations and hyperparameters, and all models were evaluated on the test set using F1-score, precision, and recall (both macro-averaged and class-weighted average). A complete list of all models can be found in Appendices I (CNN), II (RNN), and III (LSTM), while a summary of the best models of each type can be found above in Table 3. I selected the best model (indicated in red in Table 3) according to macro F1-score, since it indicates the best balance between precision and recall, and better ability to predict minority class labels (rather than achieving a high precision but low recall by skewing most predictions towards the majority class).

CNN models perform the **best** out of the three architectures tested, with the best scores for validation loss, test loss, and all metrics. They are also the **fastest** to train and optimise. Well-performing models can be trained with a tremendous variety and combination of hyperparameters, and even the randomly chosen set of hyperparameters perform quite well (see the models with 'tnr' in the model name in Appendix I). A consistent and significant improvement can be observed for models trained with the same set of hyperparameters, but using a **longer maxlen** (see Appendix I for details). Generally, the best-performing maxlen is not the mean or max sample length, but the number of tokens in the majority of samples (61 for samples represented using GloVe, 72 for samples represented using FastText). As for the embeddings, models represented with **GloVe** consistently perform better than those using FastText subword embeddings. CNN models represented with FastText also tend to skew more severely towards the majority class 'neutral', while predicting very few or even no sample as 'criticism'.

BiRNN models perform the **worst** out of the three architectures tested, with worst scores for loss and all metrics (low precision, even lower recall; see Appendix II). They don't seem to be able to learn the data well, skewing heavily towards the majority class and often ignoring the 'criticism' class altogether, predicting very few or no samples as 'criticism' while predicting real 'criticism' samples as 'hostility' or 'neutral'. BiRNN models are also **time-consuming and difficult to optimise** using Optuna, and it was not possible to optimise models when the samples were prepared with longer maxlen (61, 72, 86, and 113) due to constant InvalidArgument() error. Models trained with shorter samples (maxlen 30 and 36) can be optimised, but they are also prone to run into the same error, and I had to change the TRESampler seed multiple times to find one that allows Optuna to complete all 100 trials for each type of model (TRESampler=99 is the one that works). The best set of hyperparameters seems to use a small number of neurons in the BiRNN layer (20 or 27) and a small batch size (32 or 64). Randomly chosen hyperparameters (see models with 'tnr' in the model name) performed very poorly (see Appendix II). No relationship can be observed between **sample length and performance** of the BiRNN models. Model represented with **GloVe** performed slightly better, with the added advantage of being much faster to train and optimise.

LSTM models are able to learn and predict labels for all classes; they perform significantly **better than BiRNN** models and are **almost comparable to CNN models** in terms of performance scores (see Appendix III), though they are **much slower and more problematic to optimise.** They have the same problem as BiRNN models in hyperparameter optimisation: models trained with maxlen 61, 72, 86,

and 113 cannot be optimised due to InvalidArgument() error. Unlike CNN models, sample **maxlen** and the type of **embedding** do not seem have any significant effect on the performance of LSTM models. The best way to improve LSTM models was to optimise hyperparameters and address class imbalance.

4.2 Analysis of Methods to Address Class Imbalance

Models with the lowest validation loss in each category (CNN-GloVe, CNN-FastText, BiRNN-GloVe, BiRNN-FastText, LSTM-GloVe, LSTM-FastText) were chosen as baseline models, and new models were trained using the same hyperparameters as baseline models but applying various methods to address imbalance as described above in section 3.3, and named accordingly by adding the suffixes II, III, and IV to the name of the baseline model. The performance of baseline models and improved models can be found in Table 3 above.

For **CNN models**, all methods to address imbalance **improved recall**. As the confusion matrices in Figure 3 show, the baseline model performs very poorly on the 'criticism' class, while methods II, III, and IV all brought improvements. Method II (class-weighted training) tends to underperform in the majority class, leading to a significant drop in precision. Method IV (combination of random undersampling and oversampling) also tends to underperform in the majority class, given that the number of training samples in this class is halved. Contrary to my concern stated in section 3.3, Method **III** (random oversampling) did not seem to overfit minority classes. Methods **III and IV** tend to strike a good balance between precision and recall and almost always improve the F1 score of the model (see table 3).

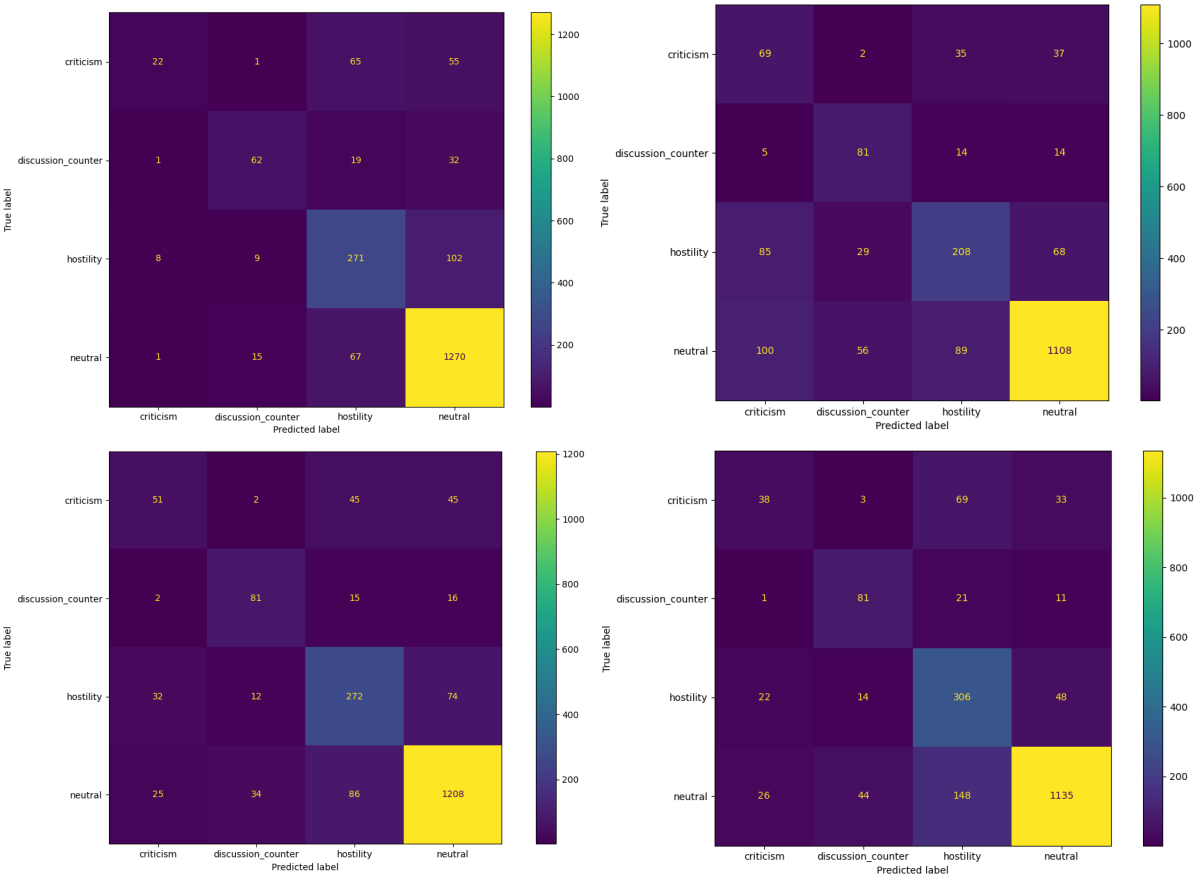

Figure 3. Confusion matrices of models *CNN_gv_model2a* (top left), *CNN_gv_model2a_II* (top right), *CNN_gv_model2a_III* (bottom left) and *CNN_gv_model2a_IV* (bottom right) (HD images can be found in this folder)

For **BiRNN models**, none of the methods to address imbalance brought any improvement to the baseline models.

For **LSTM models**, like for CNN models, all methods to address imbalance **improved recall** and **performance in the 'criticism' class** (see Figure 4). Method **III** (random oversampling) tends to perform the best, balancing precision and recall and therefore improving the F1 score.

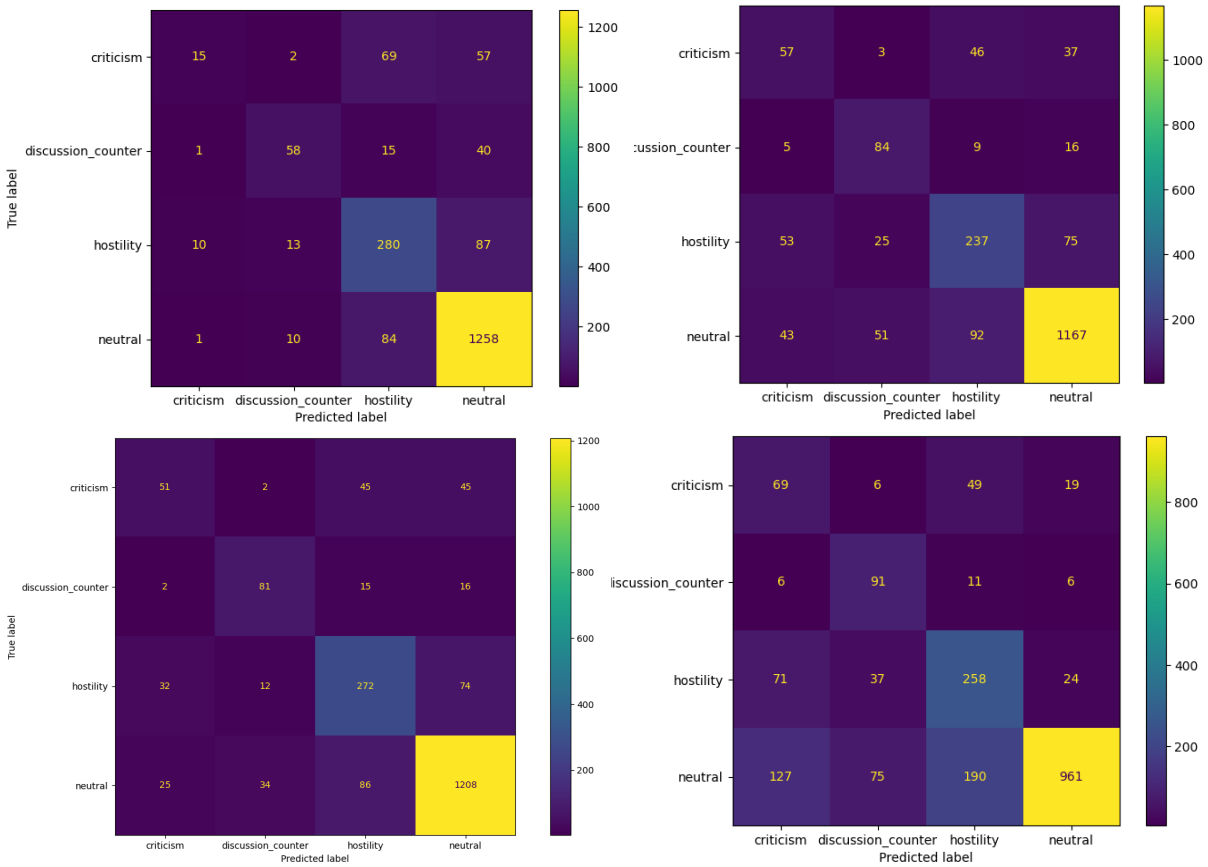

Figure 4. Confusion matrices of models *LSTM_ft_model1b* (top left) , *LSTM_ft_model1b_II* (top right), *LSTM_ft_model1b_III* (bottom left) and *LSTM_ft_model1b_IV* (bottom right). (HD images can also be found in this folder)

5. Conclusions and Future Work

This study shows that CNN is an effective model for the classification of the East-Asian Prejudice dataset, as can be demonstrated by a comparison between the confusion matrix of my best model (see Figure 3, bottom left) and that of Vidgen et al.'s best (see Figure 1 in Vidgen et al., 2020). Unfortunately, it is difficult to compare the performance scores of my models and theirs, since it is not clear how their scores are averaged between classes.

This study finds that GloVe embeddings perform better than FastText subword embeddings, on top of the advantage that they have fewer dimensions (200D rather than 300D) and are therefore much faster. This shows that using embeddings trained on a similar text type (tweets) is very important, even at the expense of dropping hashtags, neologisms, numbers, and special characters that are OOV for GloVe. It would be interesting to try substituting hashtags (like Vidgen et al. did) and rare words with more generic words (such as using 'virus' or 'ebola' for 'coronavirus') in the future, to see whether performance can be boosted. As for ways to address class imbalance, the best out of all the methods tested is to randomly oversample all minority classes.

This study also finds that the most difficult class to classify is 'criticism' (see Table 4 above), not because it is small (the smallest class, 'discussion_counter', is less difficult to predict), but perhaps because it is semantically ambivalent, since it is often misclassified as 'hostility' or 'neutral' even after

measures have been taken to address class imbalance in the dataset. In order to further improve this classifier in the future, one could examine these data samples in more detail and focus more on distinguishing between 'criticism', 'hostility', and 'neutral'.

Another interesting aspect that can be explored in the future is to see whether an effective multilingual classifier can be trained from this dataset, either by using multilingual embeddings (such as multilingual BERT, see Devlin et al., 2018), or by training a model on translated samples and testing it on a manually annotated test set in that language.

Appendices

References

Basile, V., Bosco, C., Fersini, E., Debora, N., Patti, V., Pardo, F.M.R., Rosso, P. and Sanguinetti, M., 2019. Semeval-2019 task 5: Multilingual detection of hate speech against immigrants and women in twitter. In *13th International Workshop on Semantic Evaluation* (pp. 54-63). Association for Computational Linguistics.

Brownlee, J., 2020. *Imbalanced classification with Python: better metrics, balance skewed classes, cost-sensitive learning*. Machine Learning Mastery.

Chen, E., Lerman, K. and Ferrara, E., 2020. Tracking Social Media Discourse About the COVID-19 Pandemic: Development of a Public Coronavirus Twitter Data Set. *JMIR Public Health and Surveillance*, 6(2), p.e19273.

Davidson, T., Warmsley, D., Macy, M. and Weber, I., 2017. Automated hate speech detection and the problem of offensive language. *arXiv preprint arXiv*:1703.04009.

de Gibert, Ona, Naiara Perez, Aitor García-Pablos, and Montse Cuadros, 2018. Hate speech dataset from a white supremacy forum. *arXiv preprint arXiv*:1809.04444.

Devlin, J., Chang, M.W., Lee, K. and Toutanova, K., 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv*:1810.04805.

Hapke, H.M., Lane, H. and Howard, C., 2019. *Natural language processing in action*. New York: Manning.

Kulkarni, A. and Shivananda, A., 2019. Natural language processing recipes. New York: Apress.

Kumar, R., Ojha, A.K., Malmasi, S. and Zampieri, M., 2018, August. Benchmarking aggression identification in social media. In Proceedings of the First Workshop on Trolling, Aggression and Cyberbullying (TRAC-2018) (pp. 1-11).

MacAvaney, S., Yao, H.R., Yang, E., Russell, K., Goharian, N. and Frieder, O., 2019. Hate speech detection: Challenges and solutions. PloS one, 14(8), p.e0221152.

Ross, B., Rist, M., Carbonell, G., Cabrera, B., Kurowsky, N. and Wojatzki, M., 2017. Measuring the reliability of hate speech annotations: The case of the european refugee crisis. *arXiv preprint arXiv*:1701.08118.

Schild, L., Ling, C., Blackburn, J., Stringhini, G., Zhang, Y. and Zannettou, S., 2020. "Go eat a bat, chang!": An early look on the emergence of sinophobic behavior on web communities in the face of covid-19. *arXiv preprint arXiv:*2004.04046.

Vidgen, B., Botelho, A., Broniatowski, D., Guest, E., Hall, M., Margetts, H., Tromble, R., Waseem, Z. and Hale, S., 2020. Detecting East Asian Prejudice on Social Media. *arXiv preprint arXiv*:2005.03909.

Waseem, Z., 2016, November. Are you a racist or am i seeing things? annotator influence on hate speech detection on twitter. In *Proceedings of the first workshop on NLP and computational social science* (pp. 138-142).

Waseem, Z. and Hovy, D., 2016, June. Hateful symbols or hateful people? predictive features for hate speech detection on twitter. In *Proceedings of the NAACL student research workshop* (pp. 88-93).

Zampieri, M., Malmasi, S., Nakov, P., Rosenthal, S., Farra, N. and Kumar, R., 2019. Semeval-2019 task 6: Identifying and categorizing offensive language in social media (offenseval). *arXiv preprint arXiv*:1903.08983.

Zampieri, M., Malmasi, S., Nakov, P., Rosenthal, S., Farra, N. and Kumar, R., 2019. Predicting the type and target of offensive posts in social media. *arXiv preprint arXiv*:1902.09666.

Ziems, C., He, B., Soni, S. and Kumar, S., 2020. Racism is a Virus: Anti-Asian Hate and Counterhate in Social Media during the COVID-19 Crisis. *arXiv preprint arXiv*:2005.12423.