Bypassing Censorship

Reverse Engineering a Social Media Censorship Classifier to Generate Adversarial Posts

Christopher Cross
Stanford University
chrisglc@stanford.edu

Sasankh Munukutla
Stanford University
sasankh@stanford.edu

Tan Siah Yong
Stanford University
siahyong@stanford.edu

1 Abstract

Given the rise of social media censorship and recent events, we develop models to replicate censorship filters used by Weibo, training on a dataset of censored posts. We achieve the best performance using a CNN-LSTM model (0.962 test set accuracy, 0.74 true positives accuracy), which we then use to develop substitution algorithms for altering posts in order to bypass filters. Posting our transformed posts to Weibo, our substitution algorithms reduce censorship by ~30% over a 24 hour period.

2 Introduction

With the rise of the digital age and social media, voices can be amplified and change can be catalyzed. At the same time, recent events in Hong Kong have raised questions about free speech in the face of institutional opposition. This has inspired our project to explore the evasion of censorship in social media. We analyze a large database of censored and uncensored posts from Weibo, a Chinese Twitter equivalent, in order to to find trends in the way these posts have been categorized. From this analysis, we will be able to reverse engineer and draw insights into the way these censorship algorithms work. Finally, we develop methods through which these algorithms can be evaded through homophone substitutions in order to alter posts so that they evade censors while retaining their original meaning.

3 Related Work

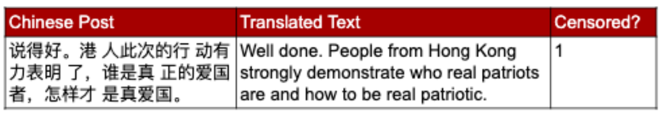
With regard to the first part of our paper, a fair amount of prior work has been done in the area of predicting censorship in Chinese social media posts. Bamman [1] found that on Sina Weibo, the presence of certain sensitive terms seem to indicate a higher probability of deletion for a post, and that the posts that contain politically sensitive content within a hot online topic are more likely to be censored. Zhu[16] further found that most censorship can occur as soon as 5-30 minutes and nearly 90% of the deletions happen within the first 24 hours of the post. Jin Li[9] took it a step further by attempting to model Weibo's censorship architecture statistically. Though Jin Li takes into account more features in addition to the ones we're considering (including user metadata like post history and followers), his models only uses traditional machine learning models with little hyperparameter tuning and simple textual features (such as Bag-of-Words). Continuing on, Yin Ng[12] focuses on the more linguistic elements of Weibo's censorship classification, exploring more complex features such as sentiment, semantic classes, and word embeddings.

With regard to the second part of our paper, much of our approach will be based upon the work done by researchers on counter-censorship substitution algorithms. Lin and Gilbert[3] discuss a method of using homophones to replace keywords known to trigger censorship, and a means of assessing their viability in avoiding censorship detecton. Primary evaluation methods include direct posting onto Sina Weibo to evaluate how long before the post gets censored, as well as the usage of Mechanical Turks to evaluate how human readable the resulting text is. However, our work will be greatly expanding off of Lin and Gilbert's, as we use the methods inspired by researchers from the first section to develop a classifier that finds more salient words that can be substituted instead than just simple keywords.

4 Dataset and Features

4.1 Data Collection

We utilized the Open Weiboscope Data Access[2] compiled from researchers from the University of Hong Kong, detailing approximately 200,000 uncensored microblog posts and 15,000 censored posts from 2012. Each microblog is assigned with a label: 1 indicating that a post was deleted due to censorship or 0 if otherwise. Roughly 8% of the microblogs have been labeled as being deleted due to censorship. Since a large imbalance exists between classes (censored/uncensored) in our dataset, a prediction bias towards uncensored posts was considered. We ultimately decided against rebalancing the dataset for two reason: 1. our analysis accounts for the precision and recall of both classes and 2. the experimental design of the second part of our project necessitates a classifier that parallels Weibo's auto censor as closely as possible, and rebalancing the dataset could introduce unnecessary variability to our results. Train/dev/test splits can be found in the experimental setup section.



Chinese Post	Translated Text	Censored?
说得好。港 人此次的行 动有力表明 了，谁是真 正的爱国者，怎样才 是真爱国。	Well done. People from Hong Kong strongly demonstrate who real patriots are and how to be real patriotic.	1

Figure 1: Example of censored post from dataset

4.2 Pre-Processing

Given that our project is NLP focused, we choose to focus entirely on the textual content of posts (i.e. ignoring user metadata). We first filter out images, retweets, and posts containing less than five words. After extracting the UTF-8 character encodings from the remaining posts, we parse out any links, numbers, and punctuation in order to reduce any noise in our data. One major challenge is the ambiguity of word segments in Chinese. Multiple characters can represent a single word and no spaces exist between each segment. Context is needed to be able to segment each word, necessitating an external word segmenter. We experimented with using the Stanford NLP Chinese Segmenter [4] and Jieba[5], but found the best results in terms of interpretability to come from the python module SnowNLP[14]. After segmentation, we eliminate stopwords (e.g. can, will) and unique words (words that appear < 5 times) in order to normalize the dataset, and output the final tokens.

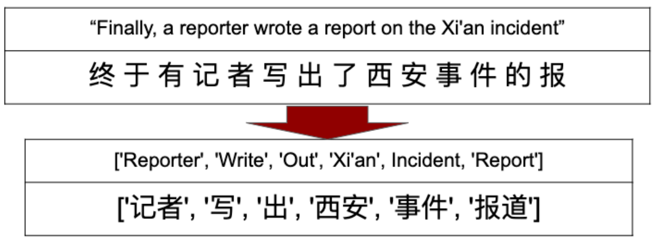

"Finally, a reporter wrote a report on the Xi'an incident"
终 于 有 记 者 写 出 了 西 安 事 件 的 报
['Reporter', 'Write', 'Out', 'Xi'an', Incident, 'Report']
['记者', '写', '出', '西安', '事件', '报道']

Figure 2: Example of preprocessed post

4.3 Feature Extraction

- **Unigram Bag-of-Words**: Our first feature involves using a "Bag of Words" (BOW) approach to count the unigram token frequency of each microblog. To generate this feature set, we first compute the entire vocabulary using every token occurring in the training set, and assign each unique token a particular index. We chose to focus on the unigram frequency over bigrams or trigrams, as unigrams tend to be more effective for classifying short texts (in this case, microblog entries).

- **TF-IDF:** One major issue with the Bag-of-Words approach is that common words such as "the" and "like" may be overly emphasized in this feature space due their relative frequencies. The term frequency-inverse document frequency (TF-IDF)[13] attempts to account for this issue by using an embedding approach that counts word frequencies for each post (similarly to BOW), but then normalizes those frequencies by their frequencies in other posts. This "inverse frequency" therefore accounts for words that aren't salient but appear more frequently in general.

- **Sequence Embeddings**: Word embeddings are a more complex family of features that seeks to map the semantic meaning of words unto a geometric space. We'll be specifically focusing on these features for the two neural models we'll be using (CNN, LSTM-CNN). We use a special type of sequence embeddings from Keras, which learns these word embeddings as a layer of a neural network so that its features are specifically tuned for the learning task (in this case, text classification). We'll also only be considering the 20,000 most common words from the dataset.

- **Pretrained W2V Embeddings:** We'll also be using a pretrained source of word embeddings trained on 230,000 Weibo posts using gensim, a python module for training word embeddings. These embeddings were collected using Word2Vec, which uses a skip-gram model and shallow two layer neural networks to train embeddings that capture more contextual information behind each word. The vocab size for the embedding is 50013 and the embedding size is 300.

5 Classification Methodology

5.1 Traditional Models

- **Multinomial Naive Bayes (MNB)**:
We have utilized MNB as they are standard models for text classification, yet fairly rigorous and robust to concept drift, which is important as the language that is censored can change over time. The likelihood for the two-class MNB takes the form:

$$\mathcal{L}\left(\phi_y, \phi_{k|y}\right) = \prod_{i=1}^{n} p\left(x^{(i)}, y^{(i)}\right)$$
$$= \prod_{i=1}^{n}\left(\prod_{j=1}^{d} p\left(x_j^{(i)} | y^{(i)}; \phi_{k|y}\right)\right) p\left(y^{(i)}; \phi_y\right)$$

where $\phi_y = j$ is the multinomial probability that example y belongs to class j and $\phi_{k|y} = j$ is the probability that feature k is present in posts of class j. $j \in \{0, 1\}$, as we have two classes: uncensored ($j = 0$) and censored ($j = 1$). Parameters are estimated using Maximum Likelihood Estimation with Laplace Smoothing, to avoid posterior probabilities dropping to zero.

- **SVM**:
Similarly to the MNB model, the SVM is a discriminative algorithm that is another standardized method for text classification that we'll be using as a baseline. We'll be using a soft-margin SVM with a "one-vs-rest" approach, optimizing for optimal parameters C and kernel parameter γ. Though literature[6] suggests that a linear kernel would be optimal for text classification, our preliminary results show the RBF kernel performs better. The SVM optimizes for a boundary of the form:

$$\min \tfrac{1}{2}\|w\|^2 \text{ s.t. } y_i\left(w \cdot x_i + b\right) \geq 1, \quad \forall x_i$$

- **Logistic Regression**:
Logistic Regression is used in a variety of text classification tasks such as spam detection. Specifically, we implemented Binary Logistic Regression, as we have two 2 possible outcomes: uncesored or censored. Logistic Regression has the form: $h_\theta(x) = g\left(\theta^T x\right) = \frac{1}{1+e^{-\theta^T x}}$ yielding the probability that a post is censored. We utilized L2 Regularization to prevent overfitting to minimize the following loss function:

$$\mathcal{L} = \underbrace{-\tfrac{1}{n}\sum_i \left[y_i \log p + (1-y_i)\log(1-p)\right]}_{\text{cross-entropy}} + \underbrace{\frac{\lambda}{2n}\sum_\theta \theta^2}_{\text{L2 reg}}$$

5.2 More Complex Models

- **LightGBM**:
  LightGBM[7][10] is Gradient Boosting Decision Tree (GBDT) algorithm that combine predictions of multiple decision trees to make predictions that generalize well with fast training speeds and high efficiency. A decision tree is a model where each non-leaf node represents a split on a particular feature, each branch represents the flow of data based on the split, and each leaf represents a classification. LightGBM was chosen over other GBDTs like XGBoost since it creates more complex trees through a leaf wise split approach rather than a level-wise approach, achieving higher accuracies. It is extremely fast to train due to exclusive feature bundling, which combines mutually exclusive features into a single feature with sub-sampling data.

- **CNN**:
  Convolutional Neural Network (CNN) is a class of deep, artificial neural networks that excel at learning the spatial structure in the input data by learning a set of filters that can be applied to data [8]. It is useful for text classification tasks as CNNs learn the spatial and temporal features of data. We first have an embedding layer, which serves as the input for the network (to represent text as vectors), then the CNN Layer that applies filters. We then use Max-Pooling after CNN layers to reduce dimension, speed up run time, and mitigate over-fitting. Given the binary nature of our classification task, we used the categorical binary cross-entropy loss as the objective function for both the CNN and LSTM-CNN models:
  $BCE = -\frac{1}{N}\sum_{i=0}^{N} y_i \cdot \log(\hat{y}_i) + (1 - y_i) \cdot \log(1 - \hat{y}_i)$

- **LSTM-CNN**:
  Long Short-Term Memory (LSTM)[15] is a type of Recurrent Neural Network (RNN) useful to process sequences of information. Unlike traditional RNNs, LSTM networks are not vulnerable to gradient explosion by using a forget gate, input gate, and output gate in each hidden unit. These gates decide how much information to let through, and therefore can connect information with a wide gap between. Given the binary nature of our classification task, we built a many-to-one LSTM model ontop of our CNN, with the following general architecture:

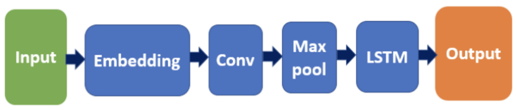


Figure 3: General architecture of LSTM-CNN pipeline

6 Classification Experiments/Results

6.1 Experimental Setup

In this section, we'll compare the training and testing accuracies of each model+feature combination (Sequence/W2V embeddings for Neural Models and BoW/TF-IDF for traditional ML models). We utilized scikit-learn's GridSearchCV to help with tuning hyperparameters for the models using a validation set, which performs an exhaustive search over combinations of parameters using 5-fold cross validation (typical value is $k = 5$) to find the best combinations of parameters. For all our experiments, we create a 80/20 split of the dataset into training and test examples. In order to perform hyperparameter tuning, we further divide our training set into 75/25 buckets for training and validation - resulting in a 60/20/20 splits across training/validation/test sets. Our two primary metrics are the general accuracy of the models as well as their F1 scores (focusing on the classification accuracy on censored posts, given the imbalance in the dataset).

6.2 Traditional Methods

We used the scikit-learn Python module as templates for our MNB, SVM and Logistic Regression models. We use the predefined CountVectorizer (Bag of Words featurizer) and TfidfTransformer (TF-IDF featurizer). We ran both approaches through the aforementioned models and used GridSearchCV to tune hyperparameters. For logistic regression, we explore C-values of [0.001, 0.01, 0.1, 1, 10, 100, 1000] for both L1 and L2 regularization. For Naive Bayes, we explore alphas of [0.001, 0.01, 0.1, 2.0, 5.0] for Laplace Smoothing. Finally, for SVM, we explore C-values of [1, 10, 100, 1000] and gamma values of [0.01, 0.001, 0.0001].

6.3 LightGBM

LightGBM was implemented using the Python Module lightgbm (GBDT as the boosting type). We ran both the CountVectorizer (BoW featurizer) and TfidfTransformer (TF-IDF featurizer) through the model. The hyperparameters tuned were the number of leaves and learning rate. These hyperparameters are the main factors for accuracy, which was optimized for[11]. This was implemented using GridSearchCV to find the ideal combination of parameters. The learning rates explored were from 0.05 to 0.20 in 0.01 increments and the number of leaves explored were [50, 100, 200, 400, 600, 800] based on LightGBM documentation[11].

6.4 Neural Models

The Python Module Keras was utilized to implement the CNN and CNN-LSTM architecture. For the Embedding Layer, we used both Keras's built-in Embedding Layer (Sequence featurizer) and Pre-trained Word Embeddings (W2V featurizer). The hyper parameters tuned were the embedding size, the number of filters, the kernel size, the pool size using GridSearchCV. For the CNN-LSTM, the LSTM output size was also tuned using GridSearchCV.

We also explored the facets of the architecture for each model. For CNNs, we experimented with two and three Convolutional Layers and Max-Pooling after each one, but ultimately a single layer was found to have the best accuracy on the test set. For CNN-LSTMs we also experimented with two and three LSTM layers, but similarly a single LSTM layer yielded the best performance.

	BoW			TF-IDF		
Model	**Train**	**Test**	**F1**	**Train**	**Test**	**F1**
Regression	0.960	0.947	0.763	0.953	0.945	0.741
Kernel SVM	0.937	0.936	0.737	0.935	0.933	0.755
Naive Bayes	0.946	0.941	0.792	0.938	0.936	0.768
Light-GBM	0.972	**0.964**	**0.793**	0.968	0.959	0.771

	Sequence Embedding			W2V Embedding		
Model	**Train**	**Test**	**F1**	**Train**	**Test**	**F1**
CNN	0.965	0.941	0.831	0.973	0.947	0.861
CNN/LSTM	0.968	0.961	0.842	0.974	**0.962**	**0.868**

Figure 4: Accuracy/F1-scores of feature+model combos

6.5 Quantitative Results

There seems to be good accuracies across the board, and especially good F1-scores for the neural models. Unsurprisingly, the more complex models seem to compute higher accuracies for both the training and testing errors. W2V-embeddings seems to work better than Sequential embeddings for neural models, and BoW seems to work better than TF-IDF for traditional models. Although the BoW+Light-GMB (0.964) model has slightly higher accuracies than the W2V+CNN/LSTM (0.962), the F1-scores of the latter are considerably better (0.793 vs. 0.868).

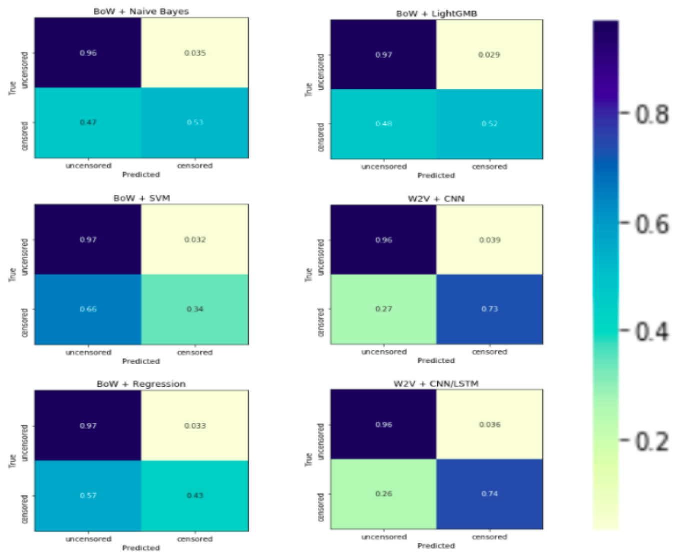


Figure 5: Confusion matrices of models w/ best features

6.6 Qualitative Analysis

Given the imbalance in the dataset, we see that the F1-scores seem to be a better metric for performance. Intuitively, it make sense that W2V would work better than the default embedding layers for the neural model, as W2V encodes contextual language information that a censor filter would consider. The higher F1-scores across the neural models compared to the tradional models imply that the neural models are better predictors for censored examples (further verified by the confusion matrices). Due to the second part of our project necessitating accurate censored predictions, this is why we'll be using the CNN-LSTM model over the LightGBM model for the substitution algorithm despite the high accuracies of the latter.

For error analysis, we also encounter a significant amount of overfitting for the CNN model compared to the other models (97.3% training accuracy versus 94.7% testing accuracy for W2V embeddings). Even with hyperparameter tuning, we weren't able to decrease that gap. This may be due to an error with regard to our model architecture (e.g. padding, convolutional layers, pooling, etc). One interesting example of an incorrectly classified text is "Glad the weekend is finally here"[translated], which was classified as uncensored when it was actually censored in the dataset. This doesn't seem to make sense since the statement is neutral at first glance. However, the Weibo censor could have also considered variables outside textual content like user-metadata (maybe the user has had a history of censored posts?). This underlines a limitation in the scope of our experiment, and a potential area of future work.

7 Generating Adversarial Posts

7.1 Homophone Substitution

Chinese uses a pictographic script where terms are a combination of one or more monosyllabic words. Due to the limited number of sounds possible, many words in Chinese share the same or similar sounds, with native speakers inferring the correct word through context. Hence, we considered substituting each word with a homonym as it would confuse the computer algorithm while still being capable of conveying its intended message to a native speaker. The end goal would be to reduce the amount of automation possible in censorship, forcing an increase in manual censorship and making such posts last longer on Weibo.

7.2 Adversarial Generation Methodology

We seek to generate adversarial posts using this homonym substitution algorithm in order bypass censorship. By using our classifier, we can determine the optimal combination of replaced words by minimizing the censorship prediction of a given post. As a baseline, we use an existing list of 9,054 blacklisted Chinese keywords compiled by The Citizen Lab. The baseline algorithm would then replace every instance of a keyword in a given censored post. Our general intuition is that our approach should be better than the baseline approach of blacklisted keyword substitution since our model presumably captures more context-specific nuances:

	Chinese	Translated	Transformed	Prediction*
Original	我恨政府腐败	I hate government corruption	我恨政府腐败	.72852
Baseline	我恨政府腐败	I hate government corruption	我恨政府[illegible]	.56332
Approach	我恨政府腐败	I hate government corruption	我痕政府[illegible]	.34616

Predicted w/ W2V+LSTM-CNN*

Figure 6: Example of model prediction of original text compared to baseline/approach transformed texts

In the example above, we see that the baseline would only replace "corruption" in the text as a blacklisted keyword, but leaves the negative sentiment expressed by the post untouched. However, our approach would also replace "hate" which is not blacklisted, indicating that our approach accounts for this negative sentiment. Note the differences in censorship prediction across all texts by our classifier.

Our algorithms are only applied to data that have been labelled as removed for censorship reasons, as our algorithm has no need to modify posts that are already considered 'safe'. We begin by changing one word within the post to an identical homonym with identical root sound and tone to prevent loss of meaning due to excessive changing. Iterating over all possible singular substitutions, our adversarial algorithm selects the substitution that minimises the probability provided by our classifier. This represents the simplest of our proposed substitution algorithms. We also consider a multi-word substitution approach here, ordering each individual replacement by the differences in probabilities between the original and replaced text and taking the top-K, where K is hand-tuned according to our own judgement based on qualitative analysis.

7.3 Generation Results

We can see an example of single-word substitution in the phrase: 秩序夜之间会帮恢复民族素质质疑群众运动恐惧民主政治绝望 to 秩序夜之间会帮恢复民族素质质疑群众运动恐惧暋主政治绝望. Observe the change of 民 to 暋 in the sixth character from the end. Despite both sentences still being read the same way, the first sentence is predicted to be censored with likelihood 0.60265 while the second is 0.52897. Overall, our single-word substitution algorithm lowered the estimated probability predicted for the modified phrase by 15.9%. A limitation is that this algorithm is unable to improve a fair proportion of test sentences. Our modified algorithm allowing multiple replacements led to an average reduction of 37.2%, and despite having several words changed, the phrases sound identical to the original when read aloud.

8 Preliminary Testing

8.1 Experimental Setup

In order to test the robustness of our transformation algorithm, we design an real-world experiment whereby we evaluate whether our transformed posts successfully resist censorship. We first draw 40 censored posts from the dataset. Then, we apply both our substitution algorithm and the aforementioned baseline to generate 2 transformed versions of the 50 posts. We use 3 newly-created accounts to manually post each of these sets of posts, and then track what proportion of each set of posts survives overtime using the official Sino Weibo API (Posts with "Permission Denied" indicates that it was removed by filters).

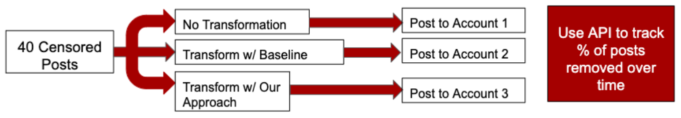


Figure 7: Outline of preliminary testing w/ Weibo accounts

8.2 Results and Discussion

Due to time constraints, we limit our timeframe to 24 hours from the time that the posts were posted. From our experimental setup, we clearly see that the transformed posts survive much longer than the untransformed posts. All of the original posts eventually get censored after around 14 hours (in line with the estimate provided by Zhu et al). We also see that our approach seems to fare slightly better than the baseline, with both a lower rate of censorship and more surviving posts in general (with 30% surviving over 22.5%). Our preliminary results therefore indicate that our approach successfully bypasses censorship to some degree.

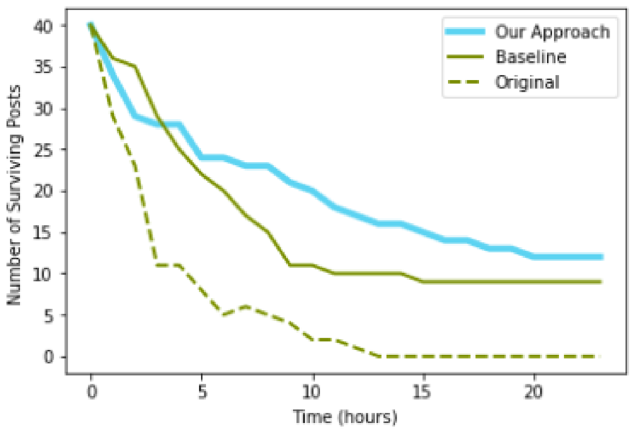


Figure 8: Proportion of posts surviving over a 24 hour timeframe across each of the 3 accounts

9 Conclusion / Future Work

From our results, we see that the LSTM-CNN trained on our pretrained Word2Vec embeddings had the best combination of classification accuracy and censored post accuracy. We believe this is due to the robustness of the model and the contextual information encompassed by the Word2Vec featurizer. We were then able to use our trained classifier to determine optimal word substitutions. Finally, we test posts developed using this algorithm in a real-world setting and find that they resist censorship more than their untransformed and baseline transformed counterparts.

If we had more time, there are definitely many fronts where we could expand our project. We could improve our classification model, working with other complex models (XGBoost) and embeddings (BERT,GloVe, etc) and features (user metadata, post history, etc) to attain higher accuracy on censored posts. We could also greatly expand our experimental setup for testing our transformed posts (e.g. using more accounts, testing exponentially more posts, distributing posts across different time frames, etc)

10 Acknowledgements

We would like to express our gratitude for the open-source Open Weiboscope Data Access, scikit-learn, Keras and LightGBM, which enabled us easily to collect the data for our project and implement our models. We would like to give special thanks the TAs of CS229 for their unwavering commitment and support. We would especially like to thank Ethan Steinberg and Leo Mehr, for their helpful feedback and support throughout the project.

11 Contributions

All members of the team contributed equally to this project. Chris gathered the dataset, implemented the feature extractors, the baseline models, and error analysis. Sasankh implemented the Logistic Regression and Complex Models. Siah Yong implemented the Homophone Substitution and Methodology. All members contributed equally to the creation of the poster and writing of the report.

References

[1] David Bamman, Brendan O'Connor, and Noah Smith. 2012. Censorship and deletion practices in Chinese social media. *First Monday* 17, 3 (2012). `https://firstmonday.org/article/view/3943/3169`.

[2] King Wa Fu. 2017. *Weiboscope Open Data.* `https://hub.hku.hk/cris/dataset/dataset107483`.

[3] Chaya Hiruncharoenvate, Zhiyuan Lin, and Eric Gilbert. 2015. Algorithmically bypassing censorship on sina weibo with nondeterministic homophone substitutions. In *Ninth International AAAI Conference on Web and Social Media.* `http://comp.social.gatech.edu/papers/icwsm15.algorithmically.hiruncharoenvate.pdf`.

[4] Galen Andrew Daniel Jurafsky Huihsin Tseng, Pichuan Chang and Christopher Manning. *Chinese Word Segmenter - Stanford NLP Group.* `https://nlp.stanford.edu/software/segmenter.shtml`.

[5] Jieba. *Jieba - Chinese Word Segmenter.* `https://github.com/fxsjy/jieba`.

[6] Thorsten Joachims. 1998. Text categorization with support vector machines: Learning with many relevant features. In *European conference on machine learning.* Springer, 137–142. `hhttp://www.cs.cornell.edu/people/tj/publications/joachims_98a.pdf`.

[7] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. Lightgbm: A highly efficient gradient boosting decision tree. In *Advances in Neural Information Processing Systems.* 3146–3154. `http://papers.nips.cc/paper/6907-lightgbm-a-highly-efficient-gradi`.

[8] Siwei Lai, Liheng Xu, Kang Liu, and Jun Zhao. 2015. Recurrent convolutional neural networks for text classification. In *Twenty-ninth AAAI conference on artificial intelligence.* `https://www.aaai.org/ocs/index.php/AAAI/AAAI15/paper/viewPaper/9745`.

[9] Jin Li. 2015. *Predicting Large-Scale Internet Censorship - A Machine Learning Approach.* `https://libra2.lib.virginia.edu/downloads/d504rk52c?filename=Jin_Li_MS_Thesis.pdf`.

[10] LightGBM. *LightGBM's documentation.* `https://lightgbm.readthedocs.io/en/latest/`.

[11] LightGBMParam. *LightGBM Parameters Tuning.* `https://lightgbm.readthedocs.io/en/latest/Parameters-Tuning.html`.

[12] Kei Yin Ng, Anna Feldman, Jing Peng, and Chris Leberknight. 2018. Linguistic Characteristics of Censorable Language on SinaWeibo. *arXiv preprint arXiv:1807.03654* (2018). `https://www.aclweb.org/anthology/W18-4202`.

[13] Juan Ramos and others. 2003. Using tf-idf to determine word relevance in document queries. In *Proceedings of the first instructional conference on machine learning*, Vol. 242. Piscataway, NJ, 133–142. `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.121.1424&rep=rep1&type=pdf`.

[14] SnowNLP. *SnowNLP - Python library for processing Chinese text.* `https://github.com/isnowfy/snownlp`.

[15] Chunting Zhou, Chonglin Sun, Zhiyuan Liu, and Francis Lau. 2015. A C-LSTM neural network for text classification. *arXiv preprint arXiv:1511.08630* (2015). `https://arxiv.org/pdf/1511.08630.pdf`.

[16] Tao Zhu, David Phipps, Adam Pridgen, Jedidiah R Crandall, and Dan S Wallach. 2013. The velocity of censorship: High-fidelity detection of microblog post deletions. In *Presented as part of the 22nd {USENIX} Security Symposium ({USENIX} Security 13).* 227–240. `https://www.usenix.org/system/files/conference/usenixsecurity13/sec13-paper_zhu.pdf`.

A sample of the resources we used and our code can be found at the following link: https://tinyurl.com/cs229chinesecensorship