Toxicity Classification for online comments

Introduction

In 2017, seven out of 10 Singaporeans are active on social media platforms on mobile devices - more than double the global average[1]. With these platforms, Singaporeans are able to leave digital footprints in the form of online comments and social media posts. However, we are all too familiar with situations where people use the Internet to perform criminal deeds. For example, terrorist groups have long used Twitter to radicalise followers; some Singaporeans have also left racially discriminatory remarks on Facebook. It is then natural for us to ask, how can a person of authority monitor a digital city filled with a mix of innocent and 'unsafe' information?

A possible approach is to create Machine Learning models that are able to digitally profile an online user or a website. For example, certain features of an individual or website can be used to judge whether one should flag them for further evaluation. It is likely that such models consider features such as how many 'unsafe' comments had a person posted or what kind of facebook pages he follows. But how does one characterise comments? As such, **our group proposes a novel Machine Learning model that is able to classify whether a standalone comment is 'unsafe'.**

In particular, we hope that our model is able to serve as an intermediary classifier to a bigger and more complex model used by an authority figure. Because of the 'straightforwardness' of our model (it only looks at a piece of text on its own and does not consider factors such as the person who posted it or when was it posted), we believe it is highly scalable and modular - any process that requires classification of text 'safeness' can use our model as an intermediary step. For example, even a Google Chrome extension targeted at youngsters which filters 'unsafe' online texts can find our model beneficial.

Project Scope

We formally define the scope of our model. Given a piece of English text that is no longer than three sentences long (around 45 words), our model adopts a few sub-models to predict 6 categorical binary labels: **toxic**, **severely toxic**, **insult**, **threat**, **obscene**, and **identity hate**. Note that a piece of text can be categorised as several of the above or none at all (i.e the classes are not mutually exclusive); the reason for this is so that there is a degree of flexibility when someone adopts our model. For example, an online forum may allow **toxic** comments but choose to remove comments labelled under **identity hate**.

Data

The dataset is taken from a publicly available labelled text corpus provided by the Conversation AI team, a research initiative founded by Jigsaw and Google (both part of Alphabet), which works on tools to help improve online conversation. The dataset of comments comes from Wikipedia page edit comment sections and are labelled by admin-moderators on Wikipedia.

Methodology

Naive-Bayes (NB) and linear methods such as Logistic Regression (LR) are often used as baselines for other more state-of-the-art text classification methods. Moreover, in (Ng and Jordan 2002), discriminative learning had been shown experimentally to have lower error rates than generative learning. In this project, we explore the idea of combining both discriminative learning and generative learning and show that with sufficient feature engineering and model combination, the accuracy of such combined baseline-models can be improved tremendously. In the first section, we will adopt a modified version of the model proposed in (Wang and Manning 2012) to produce a logistic regression model built over NB log-count ratios as features values. Subsequently, we also show that different feature engineering methods yield different performances; the feature engineering methods we will explore include: word bigrams and trigrams, character n-grams, filter feature selection and traditional methods such as word stemming, case folding and emoji stemming.

In the second section, we will adopt the Long Short-Term Memory (LSTM) idea proposed in (Hochreiter and Schmidhuber 1997) to implement a Recurrent Neural Network (RNN) model as an alternative classifier for our prob-

[1] http://www.businesstimes.com.sg/consumer/7-in-10-singaporeans-use-social-media-on-mobile-double-global-average-survey

lem. RNN model with LSTM implementation is appropriate in comment classifications because of the inherent sequence of words present in comments. By considering words as a sequence of information in a comment, we consider the iteration through a piece of comment as a traversal through discrete time steps. LSTM is able to alleviate the vanishing and exploding gradient problem which occurs in normal RNN by using forget gates which selectively choose to remember the occurrences of certain inputs in previous layer. We will also see that RNN is more suitable when clients want to introduce new training data in batches.

NB-Logistic Regression Linear model

Feature Engineering We first remove common stop words from all texts. Next, we apply case-folding and word-stemming to the comments before breaking them up into unigrams and bigrams. For a piece of text, we also compute the **Term Frequency-Inverse Document Frequency** (*tf-idf*[2]) score of each unigram and bigram that occurs in it, where *tf-idf* score is as defined:

$$tf - idf_{ngram,text} = \frac{1 + \log(N_{ngram,text})}{D_{ngram}}$$

$N_{ngram,text}$ =Number of occurrences of ngram in the text.
D_{ngram} = Logarithmic of the total number of texts divides the number of texts that the ngram has appeared.

For character-ngrams models, we do not apply word-stemming to the comments because we want to preserve orderings of character sequences. Regardless of stemming, *tf-idf* gives us a convenient way to decompose a piece of text (comment) into a vector of numerical values, where each numerical values coincide with the relative importance of the ngram feature in the linear model.
Next, for each feature column, we multiply it with NB log-count value with add-1 smoothing:

$$\mathbf{r} = \log(\frac{\mathbf{p}/\left\|\mathbf{p}\right\|}{\mathbf{q}/\left\|\mathbf{q}\right\|})$$

where $\mathbf{p} = 1 + \sum f^{+}$, $\mathbf{q} = 1 + \sum f^{-}$, f^{+} = *tf-idf* score of that ngram feature for comments having positive labels, f^{-} = *tf-idf* score of that ngram feature for comments which have negative labels and $\|\mathbf{p}\|$ = 1 + number of comments with positive labels, $\|\mathbf{q}\|$ = 1 + number of comments with negative labels.

Model We formulate a separate linear model for each classification (i.e **toxic** classification warrants a different linear model from an **insult** classification). In every model, each feature value for each text has been modified using *tf-idf* value and multiplied by the NB log-count ratio (Note that the NB log-count ratio is norm-identical to the values derived from NB). The logistic regression model is similar to a sigmoid unit which outputs a prediction probability with respect to weights. In this case, the weights are the coefficients of each feature variable in the logistic regression model. The probability of a given vector of NB-log count transformed values $(x_1, x_2...x_{||V||})$ (derived from the feature engineering step) being classified as a positive label is as follows:

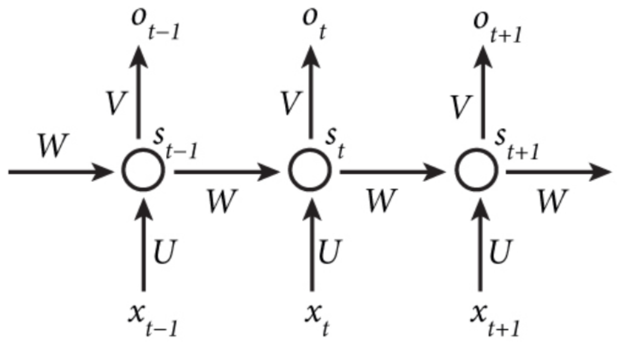
$$p(x) = \frac{1}{1 + e^{-}(\beta_0 + \beta_1 x_1 + \ldots + \beta_{||V||} x_{||V||})}$$

where $||V||$ is the number of ngrams (we take the top V ngrams occurrences depending on the model) over the collection of training texts. The steps used to obtain the coefficients β_i are similar to the Least Squares method used in linear regression models, where the misclassification error is minimised over the training texts. In our model, we find that Ridge regularization gives better performance (This may be because there are so many possible ngrams that can cause a comment to be 'bad' and thus the model should not be dominated by only a few coefficients as a result of Lasso regularization).

RNN with LSTM implementation

Feature Engineering We only perform emoji stemming, word folding and case folding on comments before tokenising them into a sequence of words.

Model RNN models assume certain dependence between inputs and outputs of each node. The following snippet of a RNN directional segment demonstrates how weights are computed:



where
W = Weights of each node
X_i = input word in sequence
O_t = output from intermediate nodes which is fed back to previous nodes
S_t = hidden states

RNN models make use of sequential information and outputs of nodes can be fed back to previous nodes. Among many other NLP tasks, RNN models are particularly suitable for the task of detecting negative comments. This is because words in a sentence depends on each other to convey an overall meaning. Hence, storing a memory of preceding words becomes crucial in terms of predicting the overall negativity of comments.

However, RNN faces both the vanishing and exploding gradient problem, which can be solved using LSTM

[2]https://en.wikipedia.org/wiki/Tf-idf

(Hochreiter and Schmidhuber 1997) with the help of activation gates taking either values 0 or 1. No matter how deep the network is, as long as the gates take on the value 1, previous outputs can be remembered for long sequences (they can be dropped as well).

There are various hyperparameters related to RNN-LSTM. In this project, we use 0.1 dropout rate (percentage of randomly selected input nodes are ignored during training, which serves as a regularization parameter) , 500 word features (number of dimensions fed at each time steps), 50 hidden layers, 32 batch-training, 2 epochs (number of passes through the full training set) and Adam Optimizer (a special type of stochastic gradient descent with a dynamically updated parameter learning rates) as proposed in (Kingma and Ba 2014). Adam Optimizer is computationally efficient and does not require much memory. These hyperparameters are tuned to minimize the performance deviation between training and validation set (to prevent overfitting).

Results and Findings

Setup

We adopted a few other feature engineering methods such as using character ngrams, varying the number of word ngrams, using binary counts instead of *tf-idf* value and others. We will summarise the performance of different methods on different types of labels in the following section. We used K-fold Cross Validation, where $K = 5$ (Our group found that for $K > 5$, the marginal returns were minimal as compared to the learning time) to generate various partitions of training and validation set while reserving an unseen test dataset. Performance indicators obtained in the validation set were used to fine tune the model and feature engineering parameters, before checking the final performance indicator on the unseen test dataset. The ratio of the dataset is approximately 17:2:1.

We also experimented with various feature selection and extraction methods, which allow each model to produce the best performance while minimising the difference in performance between training and testing dataset. Here, we present the experimental results based on the most optimal feature engineering procedure for each model.

Performance measure

We adopt the Area Under Curve (AUC) value to gauge the performance of each model. This is because in the context of classifying comments, we feel that it is equally important to capture unsafe comments as to correctly classify safe comments. The AUC value gives a good balanced measure between sensitivity and specificity. However, we will also look at the optimum threshold for translating the model outputs (a value between 0 and 1) into binary classification (since current models only produce prediction probability).

Results

	w-g LR	**w-g(0,1) NB**	**c-g NB_LR**	**w-g NB_LR**	**LSTM**
Toxic	0.948	0.889	0.964	0.96	**0.982**
Severe_T	0.962	0.954	**0.987**	0.97	0.97
Obscene	0.972	0.91	0.979	0.98	**0.984**
Threat	0.95	0.963	**0.995**	0.977	0.972
Insult	0.964	0.905	**0.977**	0.969	0.96
Identity_ Hate	0.946	0.92	0.96	0.969	**0.971**

Table 1: Comparison AUC scores for different models and different feature engineering procedures. **w-gLR**: wordgrams(1-2), *tf-idf* score, Logistic Regression (Baseline model). **w-g(0,1)NB**: wordgrams(1-2), binary count, Naive-Bayes Classifier (Baseline model). **c-gNB_LR**: charactergrams(1-6) , *tf-idf* score, NB-logcount, Logistic Regression. **w-gNB_LR**: wordgrams(1-2), *tf-idf* score, NB-logcount, Logistic Regression. **LSTM**: RNN with LSTM implementation on word sequence.

Character-Ngrams outperforms word-ngrams for linear Models

Under certain assumptions, a linear model built upon character ngrams of length 2 to 6 can be seen as more general than if built upon only on word unigrams and bigrams (e.g If a comment contains the word *'horrible'*, it will always contain the character sequence *'h-o-r-r-i'*). Hence, we can see the character-ngrams model as some kind of generalization of the word-ngram model because it also includes character sequences within words that are not considered in a mere word-ngram model. This may be the reason behind the better performance.

Furthermore, in 'unsafe' comments, users often try to obfuscate negative words with additional characters; using character n-grams can potentially detect those. This aligns with findings in (Kanaris et al. 2007), where character-ngrams are shown to outperform word-ngrams for negative text types.

Category of classification affects performance of linear models

Our group believes that 'bag-of-words/characters' models cannot classify '**identity hates**' comments well because this type of comments frequently contain innocuous words or character sequences which implies different sentiments when placed in a larger context of a longer sentence. This is also shown in (Park and Fung 2017). For example, the comment *"i love to eat black people"* (actual test example) was misclassified by the linear model as 'safe' because of the general lack of discriminatory character or word sequences. As such, linear models should be avoided when attempting to detect more subtle categories or complicated sentence implications (RNN-LSTM classifies '**identity hates**' comments better).

RNN-LSTM gives comparable AUC scores to linear models with lower optimum probability threshold

Both type of models produce prediction probabilities for the respective classes. As such, to determine the final binary classification of a comment, we have to determine the optimum probability threshold which yields satisfactory sensitivity and specificity value (remember that in some situations, both indicators are important). In the following plots of sensitivity and specificity for both models (for '**insults**' comments) at different probability thresholds, we see that the optimum probability threshold for LSTM is much lower than that of a linear model. The result is similar for classification of other classes.

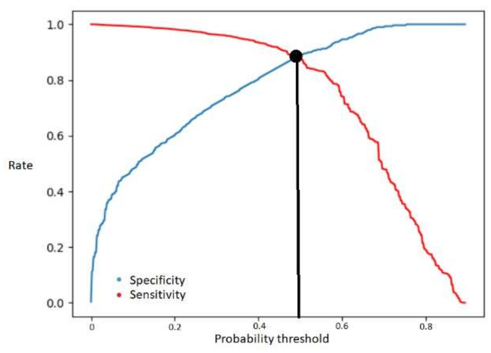


Figure 1: RNN-LSTM with $p \approx 0.5$

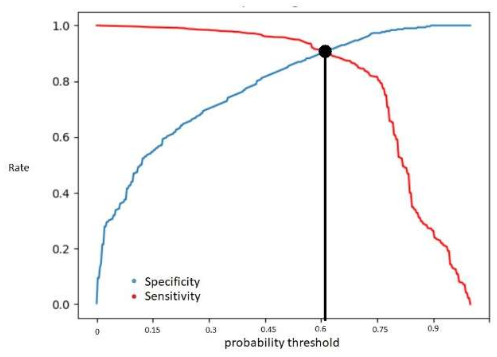


Figure 2: NB-LR with $p \approx 0.62$

In the above figures, linear models (Figure 2) give optimum classification performance only for $p \approx 0.62$ (i.e a comment is classified as '**insults**' only if model thinks that probability of comment being an insult is more than 62%) while RNN-LSTM model (Figure 1) already gives the optimum sensitivity and specificity at $p \approx 0.5$. **Note that this does not imply RNN-LSTM is more confident than NB-LR in its predictions.**

Classification-specific Feature Engineering is crucial for success

Different categories of negativity warrant careful feature selection and extraction procedure. In Figure 3 below, we see that the optimum number of character-ngrams chosen for a NB-LR model differs when classifying different categories. As such, it is important to select different number of features for models used for different categories (The procedure of selecting features based on a comparison of resulting performance is also called feature selection via wrapper methods).

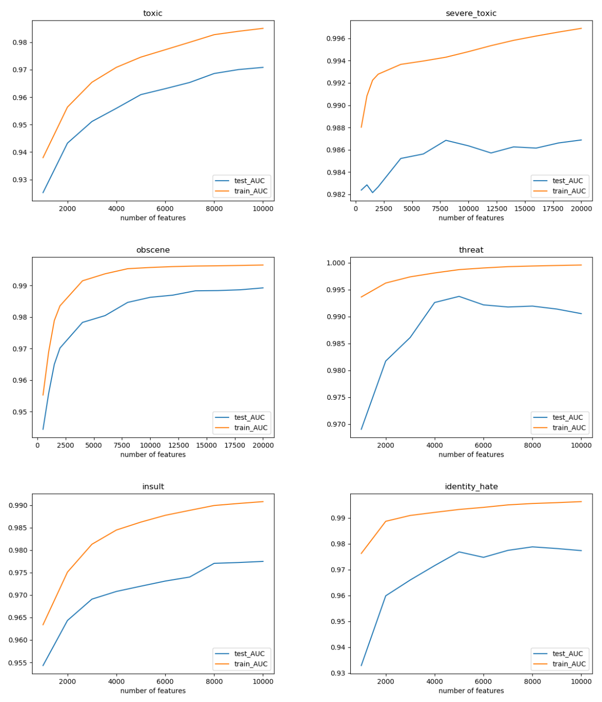


Figure 3: Variation of AUC between train and test for different number of character-ngrams in model

Feature extraction for linear models is crucial as well - in Table 2, it can be seen that transforming counts of ngrams from binary counts to *tf-idf* values singled-handedly increased performance scores across all categories for NB-LR model.

	Binary	*tf-idf*	Δ%
toxic	0.934426	0.964	+3.16
severe_toxic	0.894434	0.987	+10.3
obscene	0.943301	0.979	+3.78
threat	0.970459	0.995	+2.53
insult	0.924803	0.977	+5.64
identity_hate	0.893212	0.96	+7.48

Table 2: Variation in AUC between binary counts and *tf-idf*

Feature extraction for RNN-LSTM is equally tricky. There are many different ways to decompose a piece of comment into a sequence of objects interpretable by RNN-LSTM model. We realised that traditional vocabulary stemmers cannot tokenise comments satisfactorily because of many symbols and emojis present in online comments and thus created our custom mapping of emojis to emotional words (e.g ':)' $\mapsto$ 'smile').
Furthermore, certain obfuscation via mixing of symbols within words is hard to preprocess; this may be the reason why character-ngrams linear models outperform RNN-LSTM in certain categories.

Scalability and real-life application of Models

RNN-LSTM allows real-time scalability with new inputs of labelled comments

RNN with LSTM implementation allows new batches of comments to be trained over an existing model in *O(k)* time, where *k* is the number of new input comments. Our linear models cannot do this because it uses *tf-idf* of ngrams as features, which can all change with the introduction of new comments (for example, *tf-idf* value of a ngram changes with introduction of new comments, thus all *tf-idf* values of all ngrams will change and this takes *O((n+k)V)* time, where *n* is the number of existing training comments used and *V* is the number of ngrams features). This makes RNN-LSTM attractive for applications which want to perform continuous fine-tuning over time because it does not need to store past training comments while an application using the linear model has to store all past training comments for retraining.

This is important for applications which allow owners to continuously fine-tune the training process. For example, if a moderator on an online forum stumbles across a comment that is misclassified as a '**threat**' by our model, he could be given a quick-click option to instantly push the comment as a 'safe' comment into the existing training model for detecting '**threat**' comments, which can be retrained in a constant time.

Linear model gives satisfactory performance and can be trained much faster for one-train-fit-all situations

If an application which uses our project idea is just a novel browser extension (which attempts to hide unsafe comments on browsers), then a linear model is already good enough. Updates to the extension can be done monthly with a new batch of training data. Moreover, if a sufficiently large data set is provided in the first place, then clients may wish to adopt a one-train-fit-all method whereby no new training data will be introduced after the initialization of a model. This decreases computation costs and manpower costs in performing corrections over long time periods but may compromise on model performance.

Linear model uses concepts that are familiar to users of non-technical background

This is another advantage linear models have over RNN-based models. A linear model contains many concepts that can be intuitively explained to non-technical clients (*tf-idf* values, character n-grams, word stemming, case folding and regression).

Conclusion

Context matters

As mentioned previously, the context in which the model is used is important. For example, if the objective of our client is to aggressively remove 'unsafe comments (perhaps a google extension for very young kids to censor internet toxicity), then NB Classifier gives the best sensitivity performance but compromises specificity. On the other hand, assume that the government has a strong model to predict threat level of an individual based on his online activity, and this model uses numerous features of him, including the number of toxic comments he posts on a daily basis. Then, we will suggest to the government to use our RNN-LSTM model because it has a higher AUC score. Moreover, even though our models are not perfect, mispredictions will be passed on as noise and handled by the larger and more robust model (i.e if a user posted 20 actual toxic comments but our model classified only 18 of them as toxic and has 100% specificity, then this error difference of 2 mispredictions will be passed on as noise and, hopefully, be handled by the larger model of the government).

Future works

A clear limitation of our project is that our models were only trained on English comments. There are certainly ways to train models based on other languages but they certain require more analysis on linguistic structure (There is even an analysis on word edit distance for Korean language by (Kang 2015)) and even more complex feature extraction to decompose a piece of comment. Our models are also trained on short comments (less than three sentences long). In reality, there are many comments on social media that span longer than three sentences and our models may be inadequate in classifying these comments. As such, we could create new models in the future that cover long comments and short comments as well.

References

[Hochreiter and Schmidhuber 1997] Hochreiter, S., and Schmidhuber, J. 1997. Long short-term memory. *Neural Computation* 9(8):1735–1780.

[Kanaris et al. 2007] Kanaris, I.; Kanaris, K.; Houvardas, I.; and Stamatatos, E. 2007. Words versus character n-grams for anti-spam filtering. 16:1047–1067.

[Kang 2015] Kang, S.-S. 2015. Word similarity calculation by using the edit distance metrics with consonant normalization. *Journal of Information Processing Systems* 15(4):573–582.

[Kingma and Ba 2014] Kingma, D. P., and Ba, J. 2014. Adam: A method for stochastic optimization. *CoRR* abs/1412.6980.

[Ng and Jordan 2002] Ng, A. Y., and Jordan, M. I. 2002. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. *Proceeding of NIPS*.

[Park and Fung 2017] Park, J. H., and Fung, P. 2017. One-step and two-step classification for abusive language detection on twitter. *CoRR* abs/1706.01206.

[Wang and Manning 2012] Wang, S., and Manning, C. D. 2012. Baselines and bigrams: Simple, good sentiment and topic classification. *The Stanford Natural Language Processing Group*.

Code used for this project are available at `https://github.com/chenzhiliang94/toxic-classification`