COMP9417 Machine Learning Project: Toxic Comment Classification

1. Introduction

The proliferation of online forums and platforms where users can interact and share their opinions has become widespread with the introduction of the internet. As the scale of these platforms grow, it has become harder to monitor and identify abusive or toxic comments manually. Consequently, researchers have turned to machine learning to help identify these types of conversations at scale.

An opportunity to apply machine learning in this context can be found in the Toxic Comment Classification Challenge on Kaggle (Jigsaw/Conversation AI, 2018). The purpose of the challenge is to implement a set of classification models capable of detecting different subtypes of toxicity. Previous work on this topic has been limited to identifying if comments are toxic; in this competition we instead try to classify on the different subtypes of toxic comments. The data was provided by Kaggle and was derived from Wikipedia's Talk pages, with crowd-evaluated labels.

In this project we aim to build a set of models to classify whether a piece of text is labelled as one or more of the following labels: toxic, severe toxic, obscene, threat, insult, and identity hate. Note that the scope of our problem is binary classification on six separate labels, not one multiclass label. As such, a piece of text can be classified as one or more of the listed labels, and we will build a model for each label. The evaluation metric for our project will be the average of the individual ROC AUCs of each predicted label, consistent with Kaggle's evaluation metric. A sample of the text and labels are provided in the appendix.

2. Implementation

In this section we discuss the different aspects considered in building our final set of models, including feature extraction from text, a review of appropriate learning algorithms for this task, hyperparameter tuning, nested cross validation, the implementation of a model algorithm selection process to automate the model build across six labels, and finally, evaluation of our models on test data. Implementation was done using Python 3.8.

2.1. Feature extraction from text data

An important part of text classification is to convert a document (a piece of text) into a useful set of numeric features which can be ingested by a learning algorithm. This is typically done using bag of words, where words in the document are tokenised (separated into words) and represented in a vector of word counts, with each position in the vector representing a different token (Eisenstein, 2018). In our implementation, we used scikit-learn's *CountVectorizer* function, which converts documents into a sparse matrix of token counts (Scikit-Learn, n.d.). *CountVectorizer* has several parameters which we experimented on – notable ones included:

- *ngram_range*: specifies a sequence of N words to be used as a token (e.g. a bigram consists of a sequence of 2 words).
- *binary*: non-zero counts are capped at 1, i.e. this indicates if a word was present.
- *stop_words*: stop words are common words such as 'the', 'a' and 'and'. This option specifies whether to remove stop words.

In addition to this, we also tried term-frequency (TF) and inverse document frequency (IDF) term weighting, which reweights word counts by how often they appear; TF places more weight on frequent words whereas IDF places more weight on less frequent words. We use scikit-learn's *TfidfTransformer* function to implement this.

2.2. Review of suitable learning algorithms

Existing literature on text classification suggests that naïve Bayes, logistic regression and support vector machine are suitable baseline methods for text classification (Wang & Manning, 2012). Since these models are readily implemented using scikit-learn, we tried all of them in the model algorithm selection process and compared their performance. A description of each algorithm and their properties are summarised in Table 6 in the appendix.

2.3. Hyperparameter tuning

The hyperparameters we considered for each learning algorithm are summarised in Table 1. Unlike model parameters, hyperparameters are specified before model fitting, but also affect the performance of the model. As part of our project we would like to determine the optimal set of hyperparameters.

To implement hyperparameter tuning we used cross-validation with randomised search. Grid search was also considered, but was too computationally demanding since it tests every combination of hyperparameters. Prior to implementing this, we also did some simple testing on a few hyperparameters to confirm that they do make an impact on model performance – these results can be found in the appendix.

Table 1. Description of hyperparameters tested in our project.

Learning algorithm	Hyperparameter	Effect	Values tested
Multinomial Naïve Bayes	Laplace Smoothing α	This parameter is required to avoid the potential problem of the non-occurrence of a word which will lead to a frequency-based probability of zero. An α of 1 gives add-one smoothing , while 0 stands for no smoothing.	For this parameter we tested values from 0 to 1, at 0.1 increments.
Logistic regression	Class Weight	Class weights were considered due to imbalanced labels in the datasets. Balanced option adjusts the weights automatically by utilising the values of y according to the formula: n-samples/ n-classes * np.bincount(y). The default assumes all classes have weight of 1.	None or 'Balanced'.
	C	Inverse of the regularization strength. The smaller the values the stronger the regularization. Regularization reduces model complexity and lessens the risk of overfitting.	Integers from 1 to 5
Support Vector Machine (SVM)	Penalty Term	Lasso (L1) adds a penalty for non-zero coefficients and taking the sum of their absolute values. For higher regularization term, many coefficients will become zero leading to more sparse models. Ridge (L2) penalizes sum of squared coefficients instead of absolute value. Elastic Nets combines the penalty terms of both lasso and ridge.	L1 – Lasso Regression L2 – Ridge Regression Elastic Nets
	Alpha	The alpha parameter for SVM is a constant that gets multiplied to the regularization term. The lower the value, the weaker the regularization.	0.0001, 0.001, 0.01 and 0.1.
	Class Weight	As per the class weight in logistic regression.	None or 'Balanced'.

2.4. Choice of model evaluation metric

The evaluation metric for this Kaggle challenge is the ROC AUC.

A receiver operating characteristic (ROC) curve is a graph that visualises the performance of a binary classifier system at all classification thresholds. The ROC curve is drawn by plotting the true-positive rate against the false-positive rate at those threshold settings. From observing the ROC curves, we can compare models without specifying a single classification threshold.

$$True\ Positive\ Rate = \frac{True\ Positive}{True\ Positive + False\ Negative} \qquad False\ Negative\ Rate = \frac{False\ Negative}{True\ Negative + False\ Negative}$$

As the name suggests, AUC measures the entire area underneath the ROC curve, and is a single summarised measure of the performance across all classification threshold. AUC ranges from 0 to 1, where 0.5 suggests no discrimination and AUC >= 0.9 suggests outstanding discrimination (Lemeshow, 2013).

2.5. Algorithm selection using nested cross validation

Our goal was to determine the best learning algorithm for each of the six labels. In this context, we refer to a learning algorithm as a combination of:

a) A **learning method** (e.g. multinomial Naïve Bayes, logistic regression, or support vector machines), and
b) The **process for selecting the optimal hyperparameters for each learning method** (e.g. using a cross-validated randomised or grid search).

Algorithm selection can be done using nested cross-validation, which is considered best practice for generalising algorithm performance (Raschka, 2018). It is required because we want to obtain both the optimal hyperparameter set and best learning method for a given classification problem. Nested cross validation has two distinct parts – outer cross validation and inner cross validation.

Outer cross-validation is used to select the best algorithm. It gives an estimate of the generalised performance of each algorithm (including the process by which optimal hyperparameters are obtained). The algorithm with the best mean cross-validation AUC is selected. Outer cross-validation does not explicitly know about the optimal hyperparameters for each learning method (determined in inner cross-validation), but rather tests the stability of the entire process including hyperparameter selection.

For example, for a given learning method, different folds of the inner cross-validation loop may result in different sets of optimal hyperparameters given to the outer loop. If the resulting outer cross-validation performance is the same (i.e. different sets of optimal hyperparameters for the same learning method give similar AUC), then we can show that our algorithm is stable and generalises well with different data.

Inner cross-validation (along with randomised search) is used to select the optimal set of hyperparameters for each learning method. It gives an estimate of the generalised performance of the learning method using different sets of hyperparameters. The best set of hyperparameters is the one with the best mean cross-validation AUC, and this is passed on to the outer cross-validation

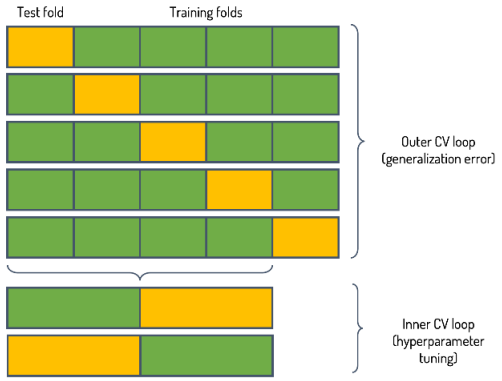


Figure 1. Diagram of how nested CV works (Schonleber, 2018). In this example, there are 5 outer CV folds and 2 inner CV folds.

process. Hyperparameters can include those from both the feature extraction steps (e.g. n_grams in CountVectorizer), and model hyperparameters (e.g. alpha in MNB).

For our implementation in Python, we set up a learning algorithm using the *pipeline* module, which also includes pre-processing steps such as word count vectorisation. For inner cross-validation we pass the created pipelines and list of hyperparameters to be tuned into a randomised search (*RandomizedSearchCV*). Grid search was initially considered as it is a more exhaustive search, but was deemed to be too computationally demanding, especially with large hyperparameter lists and the use of nested cross validation. The randomised search is then passed into the outer cross-validation loop.

Once the best learning method and its optimal set of hyperparameters were determined, we fit the selected model on the entire training set, as more data will typically improve model performance. Finally, the model's performance is evaluated against the test set.

A summary of our implementation is provided in Figure 2.

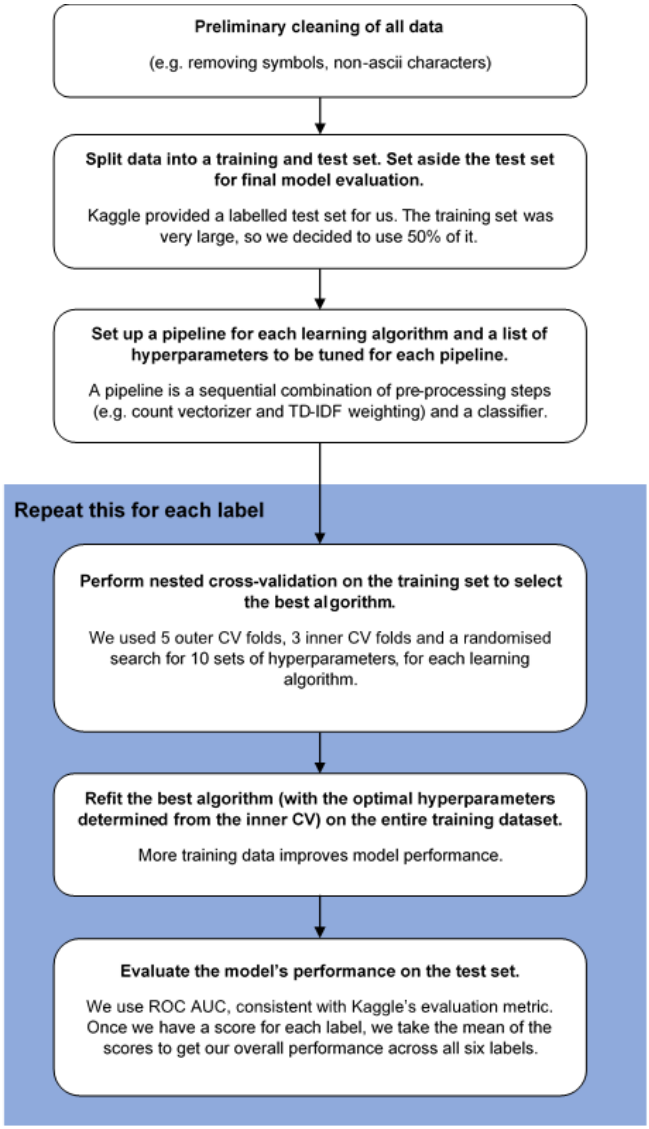


Figure 2. Summary of our implementation process.

3. Results

3.1. Exploratory data analysis and splits

Kaggle provided a labelled training and test set, containing 159,571 and 63,978[1] observations, respectively. As the training dataset was large, we decided to sample 50% of it to reduce the training time, given that we will be doing nested cross validation, which is computationally expensive. We also decided to use Kaggle's labelled test set as our own test set for comparability with leaderboard scores.

A summary of the proportion of each label in both the original training set and our sampled set is in Table 2. Note the proportions are very similar which means our sample is representative of the original set. A key observation is that the labels are very unbalanced, especially for *severe_toxic*, *threat*, and *identity_hate*. We hypothesised that models which have a hyperparameter to account for class weights (such as logistic regression or SVM) may do better given this observation.

Table 2. Proportion of positive labels in the original training set and our sampled training set.

	Number and proportion with positive label	
Label	**Original training set (n = 159,571)**	**Sampled training set (n = 79,785)**
toxic	15294 (9.58%)	7676 (9.62%)
severe_toxic	1595 (1.00%)	786 (0.99%)
obscene	8449 (5.29%)	4258 (5.34%)
threat	478 (0.30%)	243 (0.30%)
insult	7877 (4.94%)	3925 (4.92%)
identity_hate	1405 (0.88%)	674 (0.84%)

3.2. Outer cross validation results (algorithm selection)

Results from the 5-fold outer cross validation loop are summarised in Table 3. As explained in implementation, the goal of the outer loop was to select the best algorithm. The best algorithm for each label is the one which generalises well on different data and provides a consistent AUC. To determine this, we look at the mean and standard deviation of the test AUC across the 5 folds. Ideally, we want a high mean AUC and a low standard deviation, to indicate that the model fitting and hyperparameter selection process in the inner cross validation loop generalised well.

Our results indicate that logistic regression is the best algorithm for all labels. Out of the algorithms tested, it consistently has the highest mean AUC, and in most cases, the lowest standard deviation, indicating it is stable and does not overfit or respond to noisy data. However, we see that the other models still perform quite well, with slightly worse mean AUCs. SVM performs comparably to logistic regression for the *threat* and *severe_toxic* labels.

Based on our results, we proceeded to select the logistic regression learning algorithm (along with its hyperparameter selection process) for all labels.

[1] Labelled test data was released by Kaggle after the competition was finished. The original test dataset had 153,164 observations, but only 63,978 were labelled and used in the competition for evaluating models.

Table 3. Outer cross-validation results for each label. The best performing learning algorithm (pipeline) for each label is in bold. The test AUC for each fold is provided, as well as the mean and standard deviation of the 5 folds.

label	pipeline	Mean test AUC	Standard deviation	score_0	score_1	score_2	score_3	score_4
	pipe_logreg_tfidf	**0.964**	**0.005**	0.967	0.970	0.962	0.957	0.964
toxic	pipe_MNB_tfidf	0.944	0.004	0.941	0.942	0.949	0.949	0.940
	pipe_SVM_tfidf	0.919	0.044	0.874	0.956	0.947	0.858	0.960
	pipe_logreg_tfidf	**0.979**	**0.005**	0.986	0.983	0.971	0.975	0.981
severe_toxic	pipe_MNB_tfidf	0.956	0.018	0.978	0.955	0.923	0.967	0.957
	pipe_SVM_tfidf	0.977	0.01	0.981	0.957	0.981	0.983	0.983
	pipe_logreg_tfidf	**0.979**	**0.002**	0.981	0.979	0.980	0.976	0.977
obscene	pipe_MNB_tfidf	0.955	0.006	0.946	0.952	0.958	0.955	0.963
	pipe_SVM_tfidf	0.969	0.011	0.968	0.982	0.980	0.959	0.954
	pipe_logreg_tfidf	**0.966**	**0.011**	0.963	0.968	0.960	0.951	0.986
threat	pipe_MNB_tfidf	0.944	0.023	0.971	0.915	0.918	0.953	0.961
	pipe_SVM_tfidf	0.965	0.007	0.964	0.977	0.956	0.960	0.969
	pipe_logreg_tfidf	**0.969**	**0.002**	0.967	0.972	0.970	0.966	0.972
insult	pipe_MNB_tfidf	0.949	0.007	0.940	0.956	0.958	0.951	0.942
	pipe_SVM_tfidf	0.959	0.01	0.954	0.965	0.975	0.951	0.950
	pipe_logreg_tfidf	**0.966**	**0.003**	0.971	0.965	0.968	0.962	0.964
identity_hate	pipe_MNB_tfidf	0.937	0.01	0.951	0.921	0.940	0.933	0.938
	pipe_SVM_tfidf	0.94	0.028	0.941	0.886	0.957	0.957	0.960

3.3. Inner cross validation results (hyperparameter tuning for the selected learning methods)

After selecting the best learning algorithm for each label, we performed the inner 3-fold cross validation process again (on the entire training dataset) to determine the best hyperparameter set for the selected algorithm.

The results for the *toxic* label are displayed in Table 4 (results for the other labels are in the appendix). From the table, we can see the 10 sets of hyperparameters which RandomizedSearchCV selected and evaluated. Ideally, we would've liked to do more exhaustive hyperparameter tuning (which could've been achieved by using grid search, or by increasing the number of sets that RandomizedSearchCV uses), but we decided early on that time constraints would prevent us from obtaining timely results (our script to produce these results took 4 hours to run!).

For the *toxic* label, we see that the optimal hyperparameter set uses TF-IDF weighting, normalises feature vectors using the l2 norm, does not remove stop words, uses unigrams, does not cap the number of features, does not convert to lowercase, does not convert counts to a binary value, uses balanced class weights and sets C to 4.

The nature of the randomised search means it is difficult to attribute overall performance to any given hyperparameter – a grid search would have been able to give us a more definite answer to this. However, as part of developing our process for this project, we did conduct some initial experiments on a few select hyperparameters to confirm that they do make an impact on performance (as previously mentioned, these can be found in the appendix).

Table 4. Results from the inner 3-fold cross validation for the toxic label. Each row represents a hyperparameter set which was selected by RandomizedSearchCV. The corresponding hyperparameters, the mean test AUC and standard deviation are provided. The hyperparameter sets are ranked by highest mean test AUC.

TF-IDF parameters		CountVectorizer parameters					Logistic regression parameters					
use_idf	norm	stop_words	ngram_range	max_features	lowercase	binary	class_weight	C	Folds	Mean test AUC	Standard deviation	Rank
TRUE	l2		(1, 1)		FALSE	FALSE	balanced	4	3	0.966	0.0008	1
FALSE	l2		(1, 1)	50000	FALSE	FALSE		5	3	0.962	0.0005	2
FALSE	l2		(1, 2)	10000	TRUE	FALSE		5	3	0.959	0.0002	3
TRUE	l1	english	(1, 1)		FALSE	FALSE	balanced	5	3	0.958	0.0007	4
TRUE	l1		(1, 1)		FALSE	FALSE	balanced	3	3	0.949	0.0008	5
TRUE	l1		(1, 2)	10000	TRUE	FALSE	balanced	4	3	0.948	0.0005	6
TRUE	l1		(1, 1)		TRUE	FALSE	balanced	1	3	0.943	0.0006	7
FALSE	l1		(1, 1)	10000	FALSE	TRUE	balanced	5	3	0.941	0.0017	8
FALSE	l1		(1, 2)	1000	TRUE	FALSE		5	3	0.906	0.0016	9
TRUE	l1	english	(2, 2)		FALSE	FALSE	balanced	5	3	0.839	0.0029	10

Table 5. Results showing the best algorithm, optimal hyperparameters and the AUC from cross validation, training and test sets. The mean test AUC across all labels is 0.966.

Label	Best algorithm	Optimal hyperparameters	Cross validation AUC	Training AUC	Test AUC
toxic	Logistic regression	{'tf_idf__use_idf': True, 'tf_idf__norm': 'l2', 'cnt_vect__stop_words': None, 'cnt_vect__ngram_range': (1, 1), 'cnt_vect__max_features': None, 'cnt_vect__lowercase': False, 'cnt_vect__binary': False, 'LogisticRegression__class_weight': 'balanced', 'LogisticRegression__C': 4}	0.966	0.998	0.953
severe_toxic	Logistic regression	{'tf_idf__use_idf': True, 'tf_idf__norm': 'l2', 'cnt_vect__stop_words': None, 'cnt_vect__ngram_range': (1, 2), 'cnt_vect__max_features': 50000, 'cnt_vect__lowercase': True, 'cnt_vect__binary': False, 'LogisticRegression__class_weight': None, 'LogisticRegression__C': 5}	0.980	0.997	0.974
obscene	Logistic regression	{'tf_idf__use_idf': True, 'tf_idf__norm': 'l2', 'cnt_vect__stop_words': 'english', 'cnt_vect__ngram_range': (1, 2), 'cnt_vect__max_features': 10000, 'cnt_vect__lowercase': True, 'cnt_vect__binary': True, 'LogisticRegression__class_weight': None, 'LogisticRegression__C': 3}	0.977	0.994	0.965
threat	Logistic regression	{'tf_idf__use_idf': True, 'tf_idf__norm': 'l1', 'cnt_vect__stop_words': 'english', 'cnt_vect__ngram_range': (1, 2), 'cnt_vect__max_features': None, 'cnt_vect__lowercase': False, 'cnt_vect__binary': True, 'LogisticRegression__class_weight': 'balanced', 'LogisticRegression__C': 3}	0.971	0.999	0.973
insult	Logistic regression	{'tf_idf__use_idf': False, 'tf_idf__norm': 'l2', 'cnt_vect__stop_words': 'english', 'cnt_vect__ngram_range': (1, 1), 'cnt_vect__max_features': 50000, 'cnt_vect__lowercase': True, 'cnt_vect__binary': True, 'LogisticRegression__class_weight': 'balanced', 'LogisticRegression__C': 1}	0.971	0.992	0.962
identity_hate	Logistic regression	{'tf_idf__use_idf': True, 'tf_idf__norm': 'l2', 'cnt_vect__stop_words': 'english', 'cnt_vect__ngram_range': (1, 2), 'cnt_vect__max_features': None, 'cnt_vect__lowercase': False, 'cnt_vect__binary': False, 'LogisticRegression__class_weight': 'balanced', 'LogisticRegression__C': 4}	0.965	1.000	0.968

3.4. Model evaluation results

After selecting the best learning method and optimal set of hyperparameters for each label, we then fit our final models and evaluated their performance on the test set.

We fit our final models on the entire training set using the optimal set of hyperparameters found in the previous step. The reason we do is because more training data typically results in better performance (the previously trained models from the inner CV only used a fraction of the training data).

Table 5 provides a summary of the final models for each label, along with the performance of each model. Our estimate of the generalised performance of each model is provided by the test AUC, as the test dataset is unseen and has not been used to make any decisions in the modelling process.

Our overall performance as per Kaggle's evaluation metric is measured by taking the mean of the test AUC across the six labels. This gives us a mean AUC of 0.966, which is decent given that we only trained on only half of the full training dataset (for reference, the top 100 submissions scored around 0.987 or higher). We would expect that using the full training dataset may yield even better results.

An observation which is consistent across all models is that the test AUC is typically always less than the training AUC, and is about the same as the cross-validation AUC, which is to be expected.

4. Conclusions, learnings, and further work

In this project, we tested several learning algorithms and hyperparameters as part of Kaggle's Toxic Comment Classification Challenge to classify six different subtypes of toxicity.

The performance of the three learning algorithms considered were comparable, with logistic regression performing slightly better for all six labels in the algorithm selection process. The final average AUC of the six models on an independent test set was 0.966, which would have placed our team at around rank 3770 on the leader board. This is a decent score, given that we only trained on half of the original training dataset.

One key learning which we didn't realise until late in the project was that Scikit-learn's implementation of support vector machines does not actually provide probabilities directly, and we would have had to change one of the default settings to obtain these probabilities explicitly. While this did not directly affect our results (as logistic regression was chosen for every label), it would have hindered our ability to create a Kaggle submission file if SVM was chosen. To rectify this, we would have liked to turn on the option to obtain the probabilities when we ran our script (this would have been implemented using Platt scaling).

In a broader context, our models achieved their goal to detect toxic and abusive comments at scale, solving the issue of humans having to monitor and identify abusive or toxic comments manually. Additionally, our process for building all six models was largely automated, including the feature extraction, hyperparameter tuning and algorithm selection steps. The benefit of this, as opposed to bespoke models built specifically to each label, is that we can apply the process for any new labels that may come up. Furthermore, models can be retrained very easily if we see a degradation in their performance over time.

For future work, we would have liked to explore further text pre-processing techniques such as stemming and lemmatization, as these steps can play a big role in improving model performance in text classification. Further work could also be done to explore the specific effectiveness of various hyperparameters, to improve the efficiency of the randomised search.

5. References

Eisenstein, J., 2018. *Introduction to Natural Language Processing.* s.l.:MIT Press.

Jigsaw/Conversation AI, 2018. *Toxic Comment Classification Challenge.* [Online]
Available at: https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge/overview

Lemeshow, D. W. &. S., 2013. Applied Logistic Regression. In: s.l.:s.n.

Nigam, A. M. &. K., 2001. A Comparison of Event Models for Naive Bayes Text Classification. *Work Learn Text Categ,* Volume 752.

Raschka, S., 2018. *Model Evaluation, Model Selection, and Algorithm Selection in Machine Learning.* [Online]
Available at: https://sebastianraschka.com/pdf/manuscripts/model-eval.pdf
[Accessed 8 August 2020].

Schonleber, D., 2018. *A short introduction to model selection.* [Online]
Available at: https://towardsdatascience.com/a-short-introduction-to-model-selection-bb1bb9c73376
[Accessed 8 August 2020].

Scikit-learn, 2020. *Support Vector Machines.* [Online]
Available at: https://scikit-learn.org/stable/modules/svm.html#scores-probabilities
[Accessed 7 August 2020].

Scikit-Learn, n.d. *Feature Extraction.* [Online]
Available at: https://scikit-learn.org/stable/modules/feature_extraction.html#text-feature-extraction
[Accessed 7 August 2020].

Wang, S. & Manning, C., 2012. *Baselines and Bigrams: Simple, Good Sentiment and Topic Classification.* s.l., s.n.

6. Appendix

Begins on the following page.

6.1. Some examples of the text data to be classified and their labels

comment_text	toxic	severe_toxic	obscene	threat	insult	identity_hate
You are gay or antisemmitian? Archangel WHite Tiger Meow! Greetingshhh! Uh, there are two ways, why you do erased my comment about WW2, that holocaust was brutally slaying of Jews and not gays/Gypsys/Slavs/anyone... 1 - If you are anti-semitian, than shave your head bald and go to the skinhead meetings! 2 - If you doubt words of the Bible, that homosexuality is a deadly sin, make a pentagram tatoo on your forehead go to the satanistic masses with your gay pals! 3 - First and last warning, you fucking gay - I won't appreciate if any more nazi shwain would write in my page! I don't wish to talk to you anymore! Beware of the Dark Side!	1	0	1	0	1	1
FUCK YOUR FILTHY MOTHER IN THE ASS, DRY!	1	0	1	0	1	0
I'm Sorry I'm sorry I screwed around with someones talk page. It was very bad to do. I know how having the templates on their talk page helps you assert your dominance over them. I know I should bow down to the almighty administrators. But then again, I'm going to go play outside....with your mom. 76.122.79.82	1	0	0	0	0	0
I don't believe the Lisak criticism present there conforms with the NPV rule. Lisak doesn't have a neutral point of view to begin with. If an offer to polygraph or even concerned review of polygraph results shocks a complainant into thinking her lies have been uncovered, the recantation is still perfectly valid. If you know you are telling the truth, you will argue with machine or investigator. Also part of Kanin's research was a followup of the recanted story where possible to verify if any were false recantations. In all followups the recanted version of events matched what the accused said happened. Arguing that Lisak is a respected PHD is baseless if Kanin is a respected PHD. I agree that my edit wasn't as neutral as possible though, so apologize for that. Still something must be done here.	0	0	0	0	0	0
GET FUCKED UP. GET FUCKEEED UP. GOT A DRINK THAT YOU CANT PUT DOWN???/ GET FUCK UP GET FUCKED UP. I'M FUCKED UP RIGHT NOW!	1	0	1	0	0	0
Are you threatening me for disputing neutrality? I know in your country it's quite common to bully your way through a discussion and push outcomes you want. But this is not Russia.	0	0	0	0	0	0
Thanks! Undeletion was more than I'd hoped for. I'm researching the status of Texas government (including local government) copyright status, but it's slow going. Apparently works of the Florida government are usually public domain, but we don't have a similar article on Texas, so I guess I'll have to research the old-fashioned, non-lazy, actually reliable way. Or ask the copyright help desk, like you suggested. In the meantime, I'm using the fair use rationale, since it's valid while the image is used in an article. Thanks again! -	0	0	0	0	0	0
Awesome! Then I'll simply disregard your notice. Thanks!	0	0	0	0	0	0
Stupid peace of shit stop deleting my stuff asshole go die and fall in a hole go to hell!	1	1	1	0	1	0

6.2. Summary of learning algorithms considered

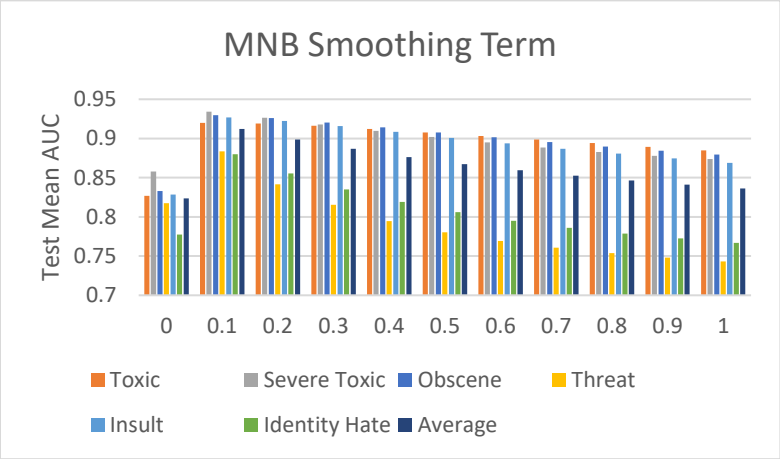
Table 6. Summary of learning algorithms considered.

Learning algorithm	Description
Multinomial Naïve Bayes	The Naïve Bayes classifier uses Bayes' theorem (along with the assumption that each feature is independent of other features) to assign class labels given a set of features. Multinomial Naïve Bayes is a variant of Naïve Bayes where features are multinomially distributed (i.e. counts of words). This algorithm is easy and fast to implement, and scales well on large datasets. While Naïve Bayes classifies well, a disadvantage is that its output probabilities are not well calibrated (Eisenstein, 2018).
Logistic regression	Logistic regression is a linear model for classification, with the assumption that features are linearly related to the log-odds of the probability of an observation belonging to a certain class. Scikit-learn's implementation of logistic regression incorporates a regularisation term, which penalises model complexity and reduces overfitting. Logistic regression gives well calibrated probabilities, which is ideal in our use case (Eisenstein, 2018).
Support Vector Machine (SVM)	Support vector machines construct a hyperplane in a multi-dimensional space which can separate points belonging to different classes. The hyperplane chosen is the one which gives the largest margin or separation between classes. SVMs are effective in high-dimensional spaces, scales well to large datasets and kernels can be used to map feature spaces efficiently. However, SVMs do not readily provide probabilities (requires Platt Scaling which uses cross validation and is thus computationally expensive) (Scikit-learn, 2020).

6.3. Results from preliminary experimentation with hyperparameters

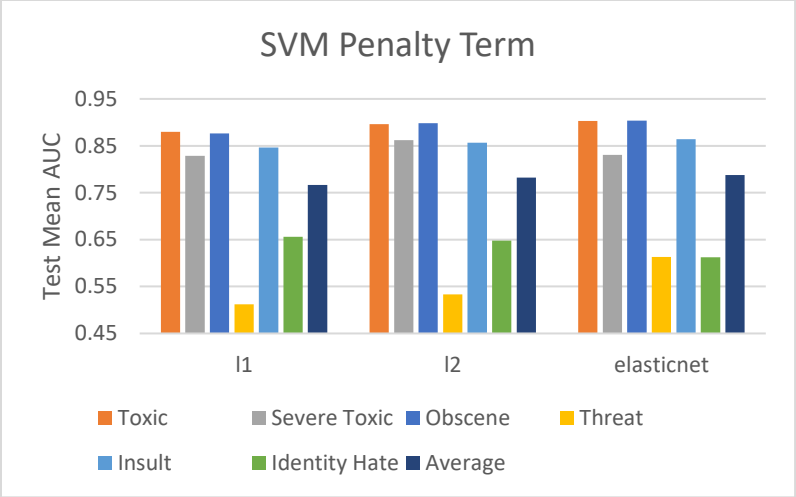
In our initial experimentation, we tested the effect of different values for a select few hyperparameters, to illustrate that there may be a clear trend. We kept all other things constant when testing different values for each hyperparameter fit on the training set and evaluated on the test set using cross fold validation.

For multinomial naïve bayes, we tested the effect of different values of the smoothing term. Keeping all things constant at their default values, we observed that a smoothing term of 0.1 is optimal for nearly all labels.

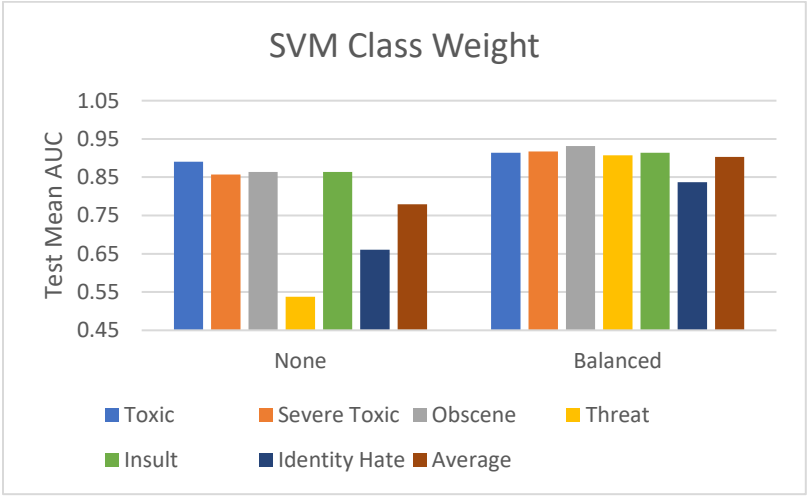


For SVM, we tested the effect of the penalty term and the class weights.

We observe that the resulting test AUC is quite similar for different penalty terms, except for elasticnet with the threat label, which performs better than l1 and l2.



For class weights, we observe that the balanced option tends to do better overall. This makes a lot of sense given that our data is highly unbalanced.



6.4. Inner 3-fold cross validation results for each label

Table 7. Inner 3-fold cross validation results for the threat label

TF-IDF parameters		CountVectorizer parameters					Logistic regression parameters					
use_idf	norm	stop_words	ngram_range	max_features	lowercase	binary	class_weight	C	Folds	Mean test AUC	Standard deviation	Rank
TRUE	l1	english	(1, 2)		FALSE	TRUE	balanced	3	3	0.971	0.0008	1
FALSE	l1	english	(1, 1)	10000	FALSE	FALSE	balanced	3	3	0.966	0.0005	2
FALSE	l1		(1, 1)		FALSE	FALSE	balanced	5	3	0.962	0.0002	3
TRUE	l1		(1, 1)	1000	TRUE	TRUE	balanced	3	3	0.944	0.0007	4
TRUE	l1		(2, 2)	50000	TRUE	FALSE		1	3	0.934	0.0008	5
FALSE	l1	english	(1, 1)	1000	TRUE	TRUE	balanced	1	3	0.917	0.0005	6
FALSE	l1		(2, 2)	10000	TRUE	FALSE	balanced	3	3	0.864	0.0006	7
FALSE	l2	english	(2, 2)	50000	TRUE	FALSE		4	3	0.789	0.0017	8
TRUE	l1	english	(2, 2)	1000	FALSE	TRUE	balanced	2	3	0.701	0.0016	9
TRUE	l2	english	(2, 2)	1000	TRUE	FALSE	balanced	3	3	0.7	0.0029	10

Table 8. Inner 3-fold cross validation results for the severe_toxic label

TF-IDF parameters		CountVectorizer parameters					Logistic regression parameters					
use_idf	norm	stop_words	ngram_range	max_features	lowercase	binary	class_weight	C	Folds	Mean test AUC	Standard deviation	Rank
TRUE	l2		(1, 2)	50000	TRUE	FALSE		5	3	0.98	0.0019	1
FALSE	l1	english	(1, 2)		TRUE	FALSE		1	3	0.978	0.0022	2
TRUE	l1	english	(1, 1)	10000	FALSE	FALSE		5	3	0.978	0.0014	3
FALSE	l2	english	(1, 2)	50000	FALSE	FALSE		3	3	0.978	0.0033	4
FALSE	l1		(1, 1)	10000	TRUE	FALSE	balanced	4	3	0.975	0.0017	5
FALSE	l2	english	(1, 1)	10000	TRUE	FALSE	balanced	4	3	0.973	0.0035	6
FALSE	l1		(1, 1)	50000	FALSE	FALSE		4	3	0.96	0.0028	7
TRUE	l2		(2, 2)		FALSE	TRUE	balanced	4	3	0.955	0.0027	8
TRUE	l1		(1, 1)	1000	FALSE	FALSE	balanced	5	3	0.95	0.0082	9
TRUE	l1		(2, 2)	10000	FALSE	FALSE	balanced	4	3	0.9	0.011	10

Table 9. Inner 3-fold cross validation results for the obscene label

TF-IDF parameters		CountVectorizer parameters					Logistic regression parameters					
use_idf	norm	stop_words	ngram_range	max_features	lowercase	binary	class_weight	C	Folds	Mean test AUC	Standard deviation	Rank
TRUE	l2	english	(1, 2)	10000	TRUE	TRUE		3	3	0.977	0.0024	1
TRUE	l1	english	(1, 1)	50000	TRUE	FALSE		1	3	0.976	0.0016	2
TRUE	l1		(1, 2)	10000	TRUE	TRUE		5	3	0.966	0.0017	3
FALSE	l1	english	(1, 1)	10000	FALSE	TRUE	balanced	2	3	0.963	0.0042	4
TRUE	l1		(1, 2)		TRUE	FALSE		5	3	0.958	0.0016	5
FALSE	l2		(2, 2)		TRUE	FALSE	balanced	5	3	0.922	0.0032	6
FALSE	l1		(1, 1)	1000	FALSE	TRUE	balanced	4	3	0.914	0.0042	7
TRUE	l1	english	(2, 2)		FALSE	TRUE		4	3	0.856	0.0075	8
FALSE	l1	english	(2, 2)	1000	FALSE	FALSE		5	3	0.683	0.0084	9
FALSE	l2	english	(2, 2)	1000	TRUE	FALSE	balanced	1	3	0.675	0.0066	10

Table 10. Inner 3-fold cross validation results for the insult label

TF-IDF parameters		CountVectorizer parameters					Logistic regression parameters					
use_idf	norm	stop_words	ngram_range	max_features	lowercase	binary	class_weight	C	Folds	Mean test AUC	Standard deviation	Rank
FALSE	l2	english	(1, 1)	50000	TRUE	TRUE	balanced	1	3	0.971	0.0009	1
TRUE	l1	english	(1, 1)	10000	TRUE	TRUE	balanced	3	3	0.966	0.001	2
TRUE	l1	english	(1, 1)		FALSE	FALSE		1	3	0.961	0.0024	3
FALSE	l1		(1, 2)	1000	TRUE	FALSE	balanced	3	3	0.923	0.006	4
TRUE	l1		(1, 1)	1000	FALSE	TRUE	balanced	4	3	0.919	0.0015	5
FALSE	l2		(1, 2)	1000	TRUE	TRUE	balanced	5	3	0.918	0.007	6
FALSE	l1		(2, 2)	10000	FALSE	TRUE	balanced	4	3	0.855	0.0024	7
TRUE	l1	english	(2, 2)		FALSE	FALSE	balanced	4	3	0.849	0.0045	8
FALSE	l2	english	(2, 2)		TRUE	FALSE		1	3	0.842	0.0052	9
FALSE	l1		(2, 2)	1000	TRUE	TRUE		3	3	0.808	0.0032	10

Table 11. Inner 3-fold cross validation results for the identity_hate label

TF-IDF parameters		CountVectorizer parameters					Logistic regression parameters					
use_idf	norm	stop_words	ngram_range	max_features	lowercase	binary	class_weight	C	Folds	Mean test AUC	Standard deviation	Rank
TRUE	l2	english	(1, 2)		FALSE	FALSE	balanced	4	3	0.965	0.003	1
FALSE	l1	english	(1, 2)		TRUE	TRUE	balanced	4	3	0.955	0.0004	2
FALSE	l1	english	(1, 1)	50000	TRUE	TRUE		5	3	0.952	0.002	3
FALSE	l1	english	(1, 1)		FALSE	FALSE		4	3	0.943	0.0072	4
FALSE	l1	english	(1, 1)	50000	FALSE	FALSE		3	3	0.942	0.0072	5
FALSE	l1		(1, 2)	10000	TRUE	TRUE	balanced	2	3	0.934	0.0088	6
FALSE	l1		(1, 1)	10000	FALSE	TRUE		5	3	0.915	0.015	7
FALSE	l1		(1, 2)		FALSE	TRUE		4	3	0.88	0.0205	8
FALSE	l1		(1, 2)	1000	TRUE	TRUE		3	3	0.863	0.0124	9
FALSE	l2		(2, 2)	1000	TRUE	TRUE	balanced	2	3	0.786	0.0173	10