Alan Kessler
MSDS 453, Section 55 Summer 2019

Assignment 4 – Final Project

Introduction and Recap from the Previous Assignment

To recap from the previous assignment, the project's goal is to take techniques from the course and apply them to a large corpus to create an accurate classifier. The first milestone for the assignment was to complete text processing and create a baseline classifier from which to compare enhancements. The next steps following involve testing alternative architectures to improve on the baseline. The code for the assignment can be found in a GitHub repository.

The data for the assignment consists of comments from Wikipedia. Each comment has six different labels related to levels of toxicity: "toxic", "severe_toxic", "obscene", "threat", "insult", and "identity_hate". A given comment can be a positive case for multiple labels. These toxic comments may be perceived as extremely offensive. There are approximately 150,000 comments in each of the training and testing data sets. The comments are processed as shown in Figure 1.

Step	Description
1	Remove punctuation
2	Remove non-alphabetic characters
3	Remove tokens under 3 characters in length
4	Remove tokens over 20 characters in length
5	Make lowercase
6	Remove stop-words
7	Lemmatize with part of speech tagging
8	Tag negations
9	Remove token "talk" if last in sentence

Figure 1 – Baseline Processing Steps

The overall modeling architecture involves reading the cleaned data to create TF-IDF features. These features are used in penalized logistic regression with separate models for each label. The steps for producing the baseline are included in Figure 2.

The baseline resulted in a public score of 0.95071 and a private leaderboard score of 0.94951. This result is a good start and could be a useful model to use in a system actually implemented but does not perform well compared to a majority of the Kaggle submissions.

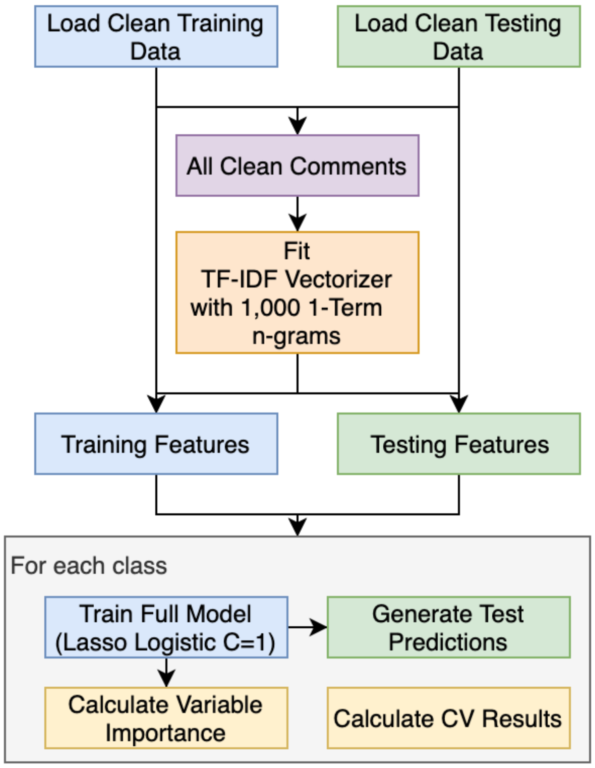
There were some significant hurdles in developing the baseline. For example, the feature importance did not indicate that negation changed the direction of certain TF-IDF features.

Areas of Potential Improvement

From the baseline, there is a question of whether the negations are actually providing value because the sign of the coefficient is the same regardless of the tag. It also makes sense to perform lemmatization prior to removing small tokens and stop words. It is possible generate a more relevant vector by lemmatizing first.

Another area for improvement is in the number of features considered. The baseline involved an informed guess of 1,000 but it is worthwhile to consider increasing this value. Logistic regression also has a small number of hyper parameters to consider like the type and strength of regularization. Tuning these could improve the model's ability to generalize. Finally, considering the labels as independent is an assumption not backed up by correlation statistics. It is possible that the model is missing out on any information that the model for one label can inform for the others.

To avoid the time-consuming task of testing these ideas over a grid, my process is to order possible enhancements or design decisions by priority and apply them one at a time.



Load Clean Training Data
Load Clean Testing Data
All Clean Comments
Fit TF-IDF Vectorizer with 1,000 1-Term n-grams
Training Features
Testing Features
For each class
Train Full Model (Lasso Logistic C=1)
Generate Test Predictions
Calculate Variable Importance
Calculate CV Results

Figure 2 – Baseline Model Architecture

Model Enhancement Process

Figure 3 shows the change in model performance by iteration. It also highlights paths that were attempted but did not improve the model. Figure 4 shows the detailed changes by iteration as well as the public and private leaderboard scores.

The first step in enhancing the model is to remove the negation tagging and move lemmatization up in the process. This change is prioritized first because of the long processing time involved. From the results, the change improves model performance but the processing time doubles to over 90 minutes. This is the result of performing part of speech tagging on more tokens rather than after filtering small tokens and removing stop words. This new processing is used in the subsequent tests as well and is shown in Figure 5.

The next enhancement is to increase the number of features to 3,000. This also improves model performance at the cost of increased processing times. Next, increasing the n-gram range from one to one to three did not improve the model. The most important features are still single tokens.

The next round of enhancements considers how best to use regularization. First, switching from L1 (Lasso) to L2 (Ridge) regularization is considered. L2 does not reduce feature weight to zero like L1, but the results indicate that using L2 improves the mean AUC. Most significantly, the processing time is improved considerably. This opens the door to testing additional features without requiring more computing resources. Following selecting a regularization method is selecting the strength of the regularization. I consider both strong and weak C-values at 0.2 and 5 respectively. Ultimately, choosing a strong regularization improved the public score slightly and is selected in part with adding additional features in mind.

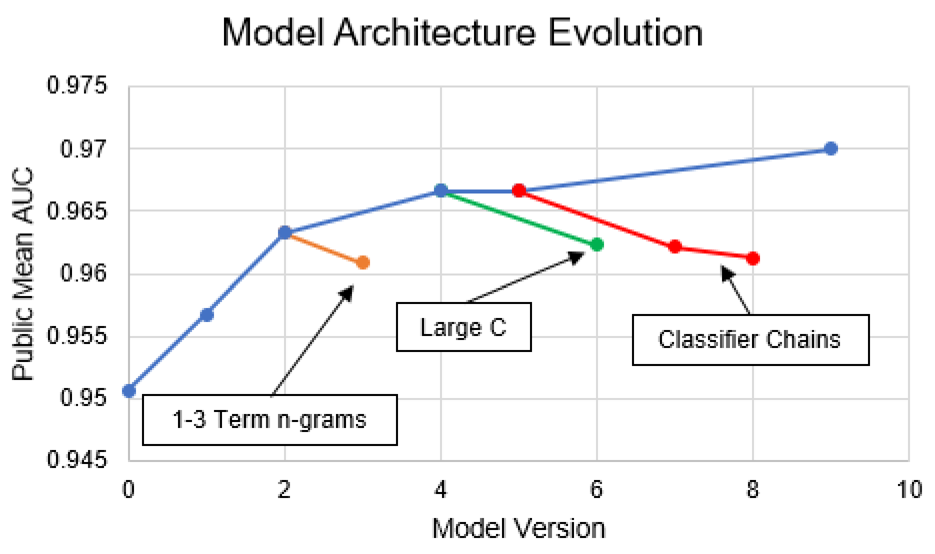


Figure 3 – Model Improvement by Iteration

Version	Change	Public Score	Private Score
0 (Baseline)	L1 logistic with 1,000 features.	0.9507	0.9495
1	Removed negation tagging and moved up lemmatization prior to filtering tokens.	0.9567	0.9563
2	Increased feature count to 3,000.	0.9633	0.9676
3	Increased n-gram range from 1 to 1-3.	0.9608	0.9650
4	Used L2 regularization.	0.9666	0.9697
5	Decreased C to 0.2.	0.9667	0.9685
6	Increased C to 5.	0.9623	0.9669
7	Implemented chain classifier.	0.9621	0.9664
8	Implemented ensemble chain classifier.	0.9612	0.9650
9	Increased feature count to 10,000.	0.9700	0.9708

Figure 4 – Model Iteration Details

Next, the assumption of independent labels is tested through the use of chain classifiers. The process is shown in Figure 6, but the idea is to use predicted probabilities by label-specific models as features in subsequent models. Each new model in the chain increases the number of features by one. Part of the rationale behind this technique is that with six labels, multiple classification could have as many as 64 diffferent classes which adds a lot of complexity (Read et al., 2009). My first step is to implement a single chain where the order of labels is based on the column order for the Kaggle submission. This technique fails to improve the scores. A possible reason is that model results will differ by the order of the chain.

To accound for the order influencing the results, it is possible to repeat the chain using randomly sampled orders. After training those models, the results are averaged to create an ensemble. This change also does not improve the model results.

Step	Description
1	Remove punctuation
2	Remove non-alphabetic characters
3	Lemmatize with part of speech tagging
4	Remove tokens under 3 characters in length
5	Remove tokens over 20 characters in length
6	Make lowercase
7	Remove stop-words
8	Remove token "talk" if last in sentence

Figure 5 – Final Text Processing Order

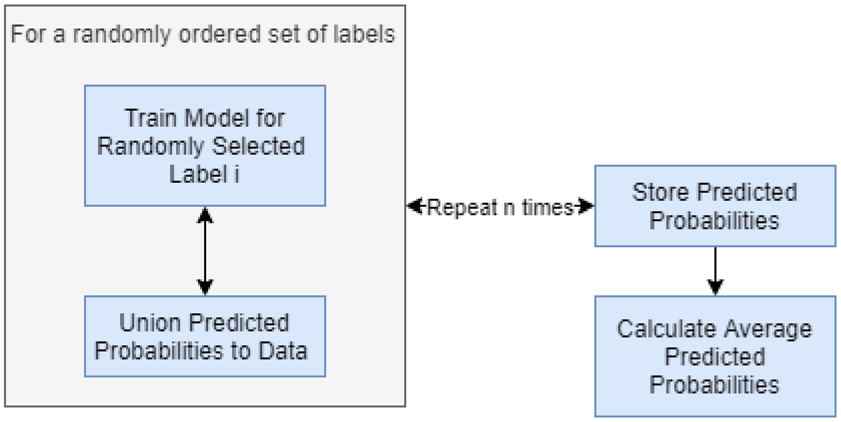

For a randomly ordered set of labels
Train Model for Randomly Selected Label i
Union Predicted Probabilities to Data
Repeat n times
Store Predicted Probabilities
Calculate Average Predicted Probabilities

Figure 6 – Ensemble Chain Classifier

The final enhancement is to increase the feature count again to 10,000. This continues to improve performance. The final model from this iteration process is shown in Figure 7.

Interpretation

The results have meaning to the overall underlying process. The first item to note is that the large number of features that continue to improve results indicates that the feature set is varied and there are not exactly perfectly formed equivalency classes to consider. However, there are some significant TF-IDF terms that standout. For example, for the threat label, both “kill” and “die” are much more important than other features. These keywords also seem to indicate why the multiple token n-grams and negations did not improve the model. Ultimately when slurs and strong words are influencial, the context near those words has less meaning. The architecture is then dependent on the use case.

Another important item from analyzing these results is the lack of improvement from using the ensemble chain classifier. It is possible that other techniques for including label information could perform better, but this technique suggests that the labels are functionally independent from one another. Looking at the feature importance, many of the same TF-IDF variables are important across labels, so this indicates that if a model responds to one of those variables, the predicted probability from another label is not adding any new information.

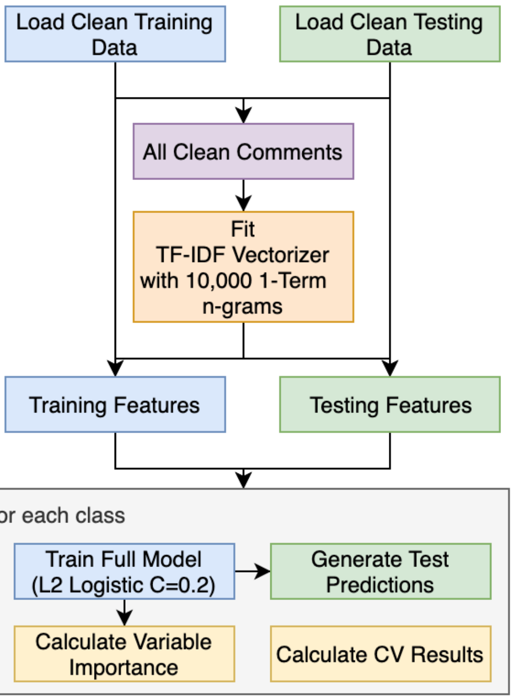

Load Clean Training Data
Load Clean Testing Data
All Clean Comments
Fit TF-IDF Vectorizer with 10,000 1-Term n-grams
Training Features
Testing Features
For each class
Train Full Model (L2 Logistic C=0.2)
Generate Test Predictions
Calculate Variable Importance
Calculate CV Results

Figure 7 – Final Model Architecture

Conclusion

This paper presents a case study for multiple label classification for natural language processing with a large corpus. It demonstrates that techniques from the course are able to scale to large data and produce classification that ranks comments well.

It highlights the importance of regularization in improving model performance and impacting training efficiency. The dimensions of the input data also show that additional information can improve the ability for the model to classify. While not of the techniques improve this particular model, they are useful tools to consider in an application like this. Finally, to the use case of Wikipedia comments, it is clear that NLP techniques can identify toxic comments well which can allow for greater automation.

Future Work

While the model does a good job ranking comments by likelihood as shown by the AUC, operational accuracy is important when building a model that would be deployed in production. Discussion participation could decrease if even small numbers of toxic comments are missed or if clean comments are tagged incorrectly. Two enhancements in particular would be interesting to try based on the discussion of the successful Kaggle submissions: pre-trained embeddings with deep learning and character n-grams. With pre-trained embeddings, the weights can serve as an input to a deep learning model. These solutions seemed to have performed very well. Character n-grams can also be generated through TF-IDF. What makes this appealing is that curse words could be embedded within some larger and possibly made up word. The downside is that these take longer to process.

References

Read, J., Pfahringer, B., Holmes, G., & Frank, E. (2009, September). Classifier chains for multi-label classification. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases* (pp. 254-269). Springer, Berlin, Heidelberg.