Capstone Project

Machine Learning Nanodegree

Patrick Harley
March 13, 2018

Project Overview

The internet provides an abundance of forums and other platforms for communication. Unfortunately, the exchange of ideas and opinions can be negatively impacted by the presence of harassing, abusive, or even threatening language.

This project focuses on detecting toxic text comments. The data was gathered from an extensive set of Wikipedia comments made by editors. While the vast majority of the posts were not offensive, a number of them were, and will be put to use in helping to build a toxicity detector.

This and related data are being used by the Conversation AI project[1], to help clean up online textual interactions - removing threats, profanity, insults and other unwanted language. The CAI team has assembled the Perspective API[2] as a framework for rooting out toxicity in online interactions. The dataset for this project comes from a competition which sought categorical classification of toxic posts.

To give weight to the significance of this effort, the group's site[3] notes: "This is a challenging and important problem for the Wikipedia community: a harassment survey conducted in 2015 found that 38% of Wikipedia editors surveyed have personally experienced harassment. Of those who've witnessed harassment, 44% reduced their involvement on Wikipedia as a result."

On a personal note, the author of this paper was for a time a moderator at a large sports forum, and has some experience with the challenges of determining what type of behavior is "over-the-line". The possibility of automating such decisions is intriguing.

The Problem

The objective is to build a classifier, one which will take a text comment as an input, and categorize it as toxic or non-toxic. The dataset provides encoded values for the binary label "toxic", for each comment. The strategy will be to convert the data to numerical form, then try a number of different algorithms from the Sklearn package, and a Neural Network as well. The binary classifier with the best metrics will be chosen as the final model.

---

[1] Conversation AI Project, https://conversationai.github.io/
[2] Perspective API Documentation, https://github.com/conversationai/perspectiveapi/blob/master/README.md
[3] *The False Positive*, https://medium.com/the-false-positive/better-discussions-with-imperfect-models-91558235d442

Metrics

Due to the definition of the dependent ("toxic") variable, a testing classification of toxic will be positive, non-toxic negative.

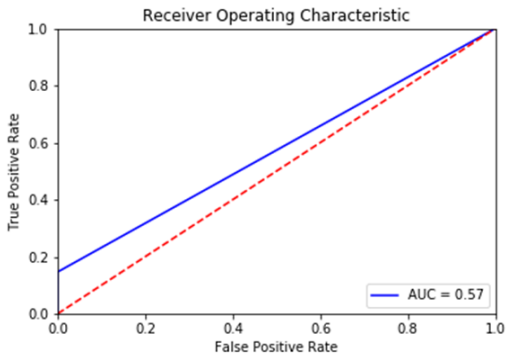
For this type of classification, standard Python Scikit-Learn packages[4] can be employed to calculate the numbers of true positives/negatives and false positives/negatives in a test set. These can be used to calculate the following useful metrics (note TP = true positive, TN = true negative, FP = False Positive, FN = False Negative):

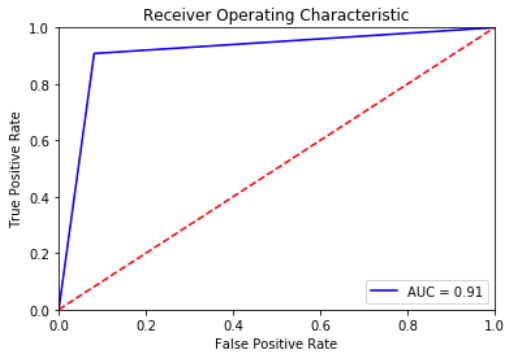
- $accuracy = \frac{TP + TN}{number\ of\ samples}$, the overall rate at which the model predicts correctly on the test set.
- $precision = \frac{TP}{TP + FP}$, which for this project would give the useful percentage of comments the model has classified as toxic, that actually are toxic. If this gauge is too low, the number of false positives (labeled toxic when non-toxic) is high relative to true positives (labeled toxic correctly).
- $recall = \frac{TP}{TP + FN}$, which in this domain would be the proportion of all toxic comments in the set that the model has classified as such. In this study, this tended to be the most volatile of the metrics, as the number of False Negatives varied with different approaches, inducing big swings in the denominator.

In addition, ROC/AUC curves will help visualize the relationships between TPR (True Positive Rate, same as recall) and False Positive Rate,

- $FPR = \frac{FP}{FP + TN}$, also known as *fallout*

A full discussion of the construction of ROC curves is out-of-scope, as it involves the intersection of indefinite integrals. Put briefly, AUC stands for "Area Under Curve", and models relying on this statistic seek to get AUC close to 1, visually assessed as its closeness to the upper left corner. Below, the right curve is better. Here, the ROC curve would represent a model's ability to correctly identify toxic comments vs. its false identifications at different thresholds on the FPR axis.





[4] *Working With Text Data,* http://scikit-learn.org/stable/tutorial/text_analytics/working_with_text_data.html

Dataset

The data was taken from a competition hosted by website kaggle.com[5]. The original competition sought a multi-class classifier which would assign comments to as many as 6 categories: *toxic, severe_toxic, obscene, threat, insult, identity_hate*. For this project, only the 'toxic' arch-category will be used as a label. Thus the two crucial columns from the set will be the comment itself, "text_comment", and the "toxic" label, already encoded as 0/1. The data had been cleaned beforehand, and did not require preprocessing.

Although the site also offered a test set, due to the competition, this was unlabeled. Thus the set provided in the file "train.csv" will have to be split and used for both training and testing. That being said, the file contains 159,572 comments, and the amount of available data should be sufficient for the task at-hand. Generally, a 90/10 percent split will be employed.

There are 144,277 non-toxic comments, and 15,294, or 9.6% toxic ones.

Examples of non-toxic comments:

> *You, sir, are my hero. Any chance you remember what page that's on?*

> *I can't make any real suggestions on improvement - I wondered if the section statistics should be later on, or a subsection of ""types of accidents"" -I think the references may need tidying so that they are all in the exact same format ie date format etc. I can do that later on, if no-one else does first - if you have any preferences for formatting style on references or want to do it yourself please let me know.*
>
> *There appears to be a backlog on articles for review so I guess there may be a delay until a reviewer turns up. It's listed in the relevant form eg Wikipedia:Good_article_nominations#Transport "*

Examples of Toxic Comments (censored):

> ***** YOUR FILTHY MOTHER IN THE ***, DRY!*

> *Why can't you believe how fat Artie is? Did you see him on his recent appearance on the Tonight Show with Jay Leno? He looks absolutely AWFUL! If I had to put money on it, I'd say that Artie Lange is a can't miss candidate for the 2007 Dead pool!*
>
> *Kindly keep your malicious fingers off of my above comment, . Everytime you remove it, I will repost it!!!*

[5] *Toxic Comment Classification Challenge*, https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge

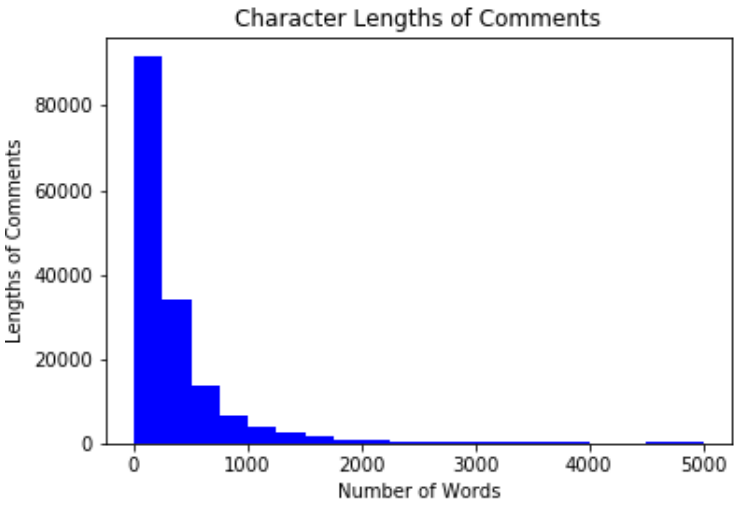
Note that the last toxic comment above doesn't use profanity, which might make it harder to detect.

Using the word cloud[6] library for Python, some visualization of common words is possible, weighted by number of appearances. Because of the large number of samples, the clouds were constructed on samples. First, taken straight from the set unfiltered:

some of these are generic, like "think", "see" or "will", some more typical to the task-at-hand like "article", "page", "section" and (of course) "Wikipedia". A word cloud was also constructed for a sample of toxic comments:

Which suggests a profane environment with threats, insults, with some paranoia mixed in for good measure. Looking at some summary statistics for the character lengths:

[6] *amueller/word_cloud*, https://github.com/amueller/word_cloud



Character length statistics:
Mean length of comments: 394.073221325
Standard Deviation of length of comments: 590.718430938
Median length of comments: 205.0
Minimum length of comments: 6
Maximum length of comments: 5000

Character lengths statistics, toxic comments:
Mean length of toxic comments: 295.2460442
Standard Deviation of length of toxic comments: 617.358841359
Median length of toxic comments: 123.0
Minimum length of toxic comments: 8
Maximum length of toxic comments: 5000

While the toxic comments tend to be shorter on average, they still range from 8-5000 (max) characters, and the standard deviation is *higher* than the full set, signaling greater variability. The difference in the statistics doesn't seem great enough to consider taking (longer) samples out of the training set.

Algorithms And Techniques

Notes on Special Techniques:

• The dataset is imbalanced; one class (non-toxic) makes up 90% of the samples. This can present problems for conventional algorithms, as they are often "biased towards the majority class because their loss functions attempt to optimize quantities such as error rate, not taking the data distribution into consideration"[7]. There are various techniques to attempt to ameliorate this setup, such as oversampling the minority class, or undersampling the majority one. The latter was applied to several of the chosen candidate algorithms.

[7] *Learning From Imbalanced Classes*, Tom Fawcett, https://www.svds.com/learning-imbalanced-classes/#fn2

• As the inputs are words in the text comments, the feature space is very high-dimensional. Some common techniques for dealing with this situation were employed:
(a) De-emphasizing a corpus of common and generally non-predictive inputs as "stop words".
(b) Vectorizing the words as sparse matrices so numerical techniques can be applied.
(c) Using inverse document frequency (IDF) to tease out less common, but more predictive words.

Algorithm: Naive Bayes

Naive Bayes is a fast-to-train and simple model that generally performs well on classification tasks. A crucial assumption the algorithm makes is the independence of the features. This project will use sklearn's MultinomialNB as a benchmark model.

Algorithm: Support Vector Machine

Support Vector Machines are a popular choice for classification, and thus are a good candidate for this binary labeling task. Some observed strengths:

- Ability to work with high-dimensional feature spaces (the case here).
- Works well on small, clean datasets
- Works efficiently on a subset of the training data (support vectors).

That being said, the algorithm can be slow to train, and can struggle with situations where there are overlapping classes, or imbalanced datasets[8].

Sklearn offers different options for implementing SVM's. Following suggestions from documentation[9] at their site, it was decided to fit an SGDClassifier, which employs Stochastic Gradient Descent for updating weights. As with the baseline previously, the comments were vectorized, and assigned inverse document frequencies. After some experimentation, particularly with the regularization parameter ('alpha'), the metrics were calculated on the test set.

Algorithm: Logistic Regression

Logistic Regression is another widely used algorithm for classification, especially with binary labels, as in this project. Using the logistic "sigmoid" function, probabilities between 0 and 1 are output, which can then be interpreted as classes. LR can also be regularized with a penalty term. A vanilla LR was constructed, with no adjustments to the defaults, and the results were promising. So a grid search was performed on several parameters, and the adjustments yielded improved scores in all categories

LSTM Network

Most of the contestants in the Kaggle competition that the dataset came from were constructing Neural Networks to attack the larger multi-class problem. After some research on text classification, and the special challenges posed in Natural Language Processing, I decided to construct a form of Recurrent Neural Network known as an LSTM (Long Short Term Memory). LSTM can be used as a classifier with

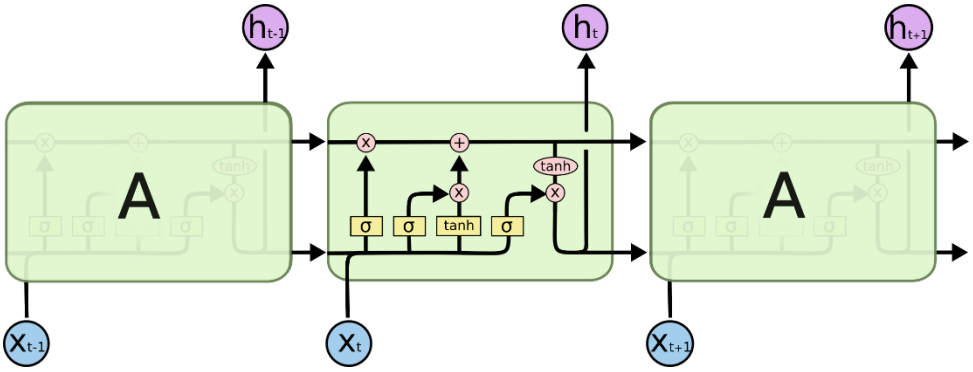
---

[8] *Applying Support Vector Machines to Imbalanced Data Sets,* Rehan Akbani https://www.researchgate.net/publication/221112311_Applying_Support_Vector_Machines_to_Imbalanced_Data_Sets

[9] *Working With Text Data,* http://scikit-learn.org/stable/tutorial/text_analytics/working_with_text_data.html

several different loss and decision functions, and has proven to be a top performer for text classification, making it an excellent candidate for the task at hand.

An RNN consists of repeated layers of the same type of block[10]. The replication of identically structured layers allows the network to feedback information through itself (recursively). The difference between a conventional RNN and LSTM layer is in the composition of the layer[11] itself.



The LSTM block has special "gates" that allow updating of the "remembered" information, via matrix operations. This structure allows the model to both "forget" old information, and "add" other information, with new inputs. In this way, the LSTM network is able to hold context for longer periods than a traditional RNN, and have traditionally outperformed them as well. This is especially helpful for detecting relationships between words that are "far apart" in a sentence, but whose connections are important for understanding said sentence.

Benchmark Model and Metrics

In the dataset there are 144,277 non-toxic comments, 15,294 toxic ones. Therefore a "naive model" which predicted every comment was non-toxic would have accuracy 90.4%. To get a feel for the problem, a Naive Bayes model was fit and tested, after vectorizing the set of words, and applying scikit's inverse document frequency package. Note that "naive" means different things in context. In the first instance, the model is naive in that it always picks the majority class. In the latter case, "Naive" refers to an assumption of independence among the predictors, proper to the Bayesian model.

The Multinomial Naive Bayes model yielded a slightly improved accuracy of 92.2%, with precision 99%.

```
Accuracy score:  0.9217320466223837
Precision score:  0.9914893617021276
Recall score:  0.15743243243243243
F1 score:  0.2717201166180758
```

However, the f-score was only 25%, dragged down by a dismal recall of 14%.

[10] *Long Short Term Memory*, https://en.wikipedia.org/wiki/Long_short-term_memory
[11] *Understanding LSTM Networks*, http://colah.github.io/posts/2015-08-Understanding-LSTMs/

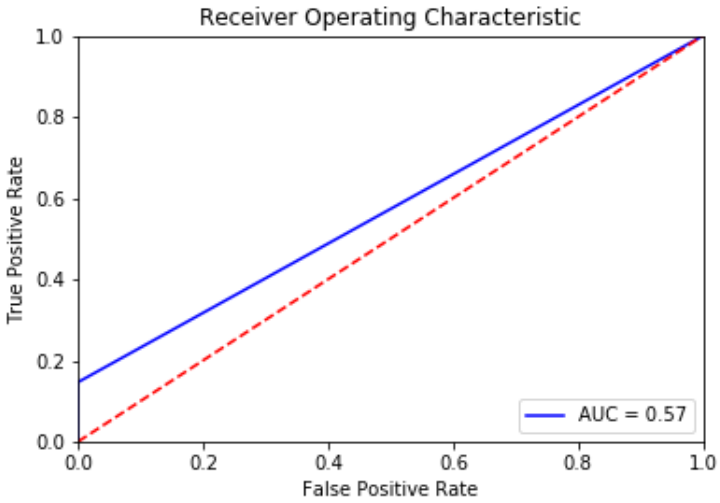
Looking at the confusion matrix gives some insight. Note that 'negative' is non-toxic, 'positive' is toxic:

	Predicted: Negative	Predicted: Positive
Actual: Negative	36073	5
Actual: Positive	3256	559

The NB algorithm has only predicted 5 comments to be toxic, when said comments were non-toxic (false positive). However, it has misclassified 3256 toxic comments as non-toxic, compared to only 559 toxic ones correctly. Overall, the model has predicted only 564 of the 3820 toxic comments in the test set to be toxic.

Further modeling attempts need to be more effective at classifying toxic comments as such, as the Naive Bayes algorithm seems to err to the side of over-classifying inputs as non-toxic.

The ROC curve is also telling in this regard:



Ideally, the blue line tracking AUC (area-under-curve) would come as close to the top left corner as possible. But because the True Positive rate is so stunted in this model (few toxics classified as such), the ROC is actually linear.

Methodology

Data Preprocessing

- This dataset was previously constructed and cleaned for a competition, and there are no missing values that need to be imputed, or issues in format.
- The set has an over-arching "toxic" category, with sub-categories pertaining to classes of toxicity (*i.e.* each row can have multiple 1's). As previously stated, only the "toxic" class will be used as a label, the others discarded.
- Kaggle's test set is unlabeled, so the central "train.csv" will be split for training, validation, and testing. All reported metrics below are for the test set.

Implementation : A Tour of Algorithms

It is good to remind ourselves of the results of the benchmark phase. The *Naive Model* which simply picks non-toxic for every sample will be correct 90.4% of the time. The Naive Bayes model didn't improve much on this, perhaps a signal of the difficulty earlier noted – that conventional algorithms may struggle with imbalanced classes. At this point, we consider a modification of the baseline, and then look at other models.

Naive Bayes with Undersampling

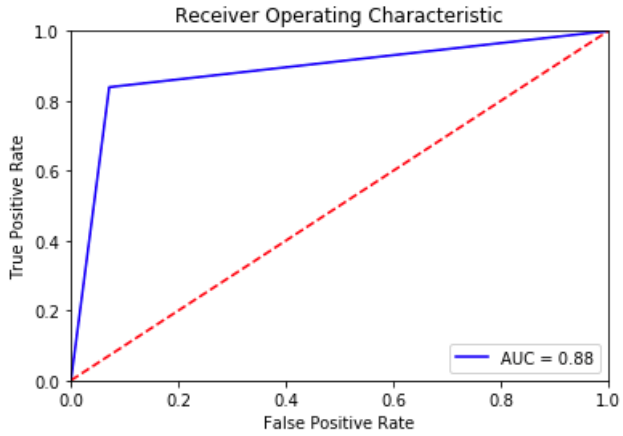
The python package[12] Imbalanced-learn was imported, and the RandomUnderSampler object was used in a pipeline with a TFIDFVectorizer and MultinomialNB. The results are interesting.

```
Accuracy score:  0.9204787567364331
Precision score:  0.5464144302683678
Recall score:  0.8391891891891892
F1 score:  0.6618705035971223
```

The accuracy has actually *decreased*, though the f-score has improved. Most notable here is the inversion in the comparative values of Recall vs Precision, that was so profound in the baseline. The Recall is now significantly greater.

	Predicted: Negative	Predicted: Positive
Actual: Negative	13447	1031
Actual: Positive	238	1242

As can be seen, the number of false negatives has decreased, from 3256 to 238, while the number of false positives has risen equally dramatically. Describing this qualitatively, undersampling seems to produce a model which is less conservative in assigning the "toxic" label. As Recall is the y-value for the ROC Curve, there's a corresponding improvement in area-under-curve (AUC):



[12] Imbalanced-learn, 0.3.0, http://contrib.scikit-learn.org/imbalanced-learn/stable/index.html

SVM

Again, a pipeline was employed, with words-to-numbers by the CountVectorizer class, TFIDFTransformer for the document frequencies, and the SGDclassifier.

```
Accuracy score:  0.9632159418473493
Precision score:  0.8906386701662292
Recall score:  0.6878378378378378
F1 score:  0.7762104460541365
```

This is a notable improvement from the baseline model across all categories except precision. The modest decline is counter-balanced by the much higher recall, producing an f-score of 77.6%. The AUC was 0.84 for this SVM.

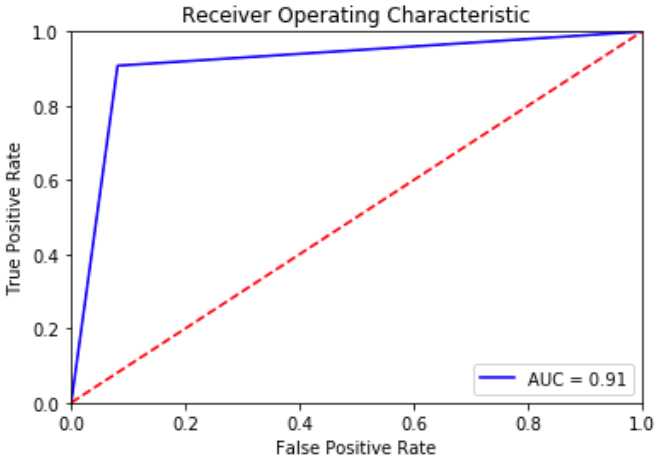
	Predicted: Negative	Predicted: Positive
Actual: Negative	14353	125
Actual: Positive	462	1018

SVM With Undersampling

Given the previous experiment with undersampling, it seemed wise to apply it to this model as well. Again, the pipeline of vectorization, IDF, RandomUnderSampler, and the classifier was assembled.

```
Accuracy score:  0.9177841834816393
Precision score:  0.5333333333333333
Recall score:  0.9081081081081082
F1 score:  0.672
```

There's an even more precipitous drop in accuracy, compared to the NB case. But again, we see the sharp increase in Recall, with Precision falling. The AUC score is 91%



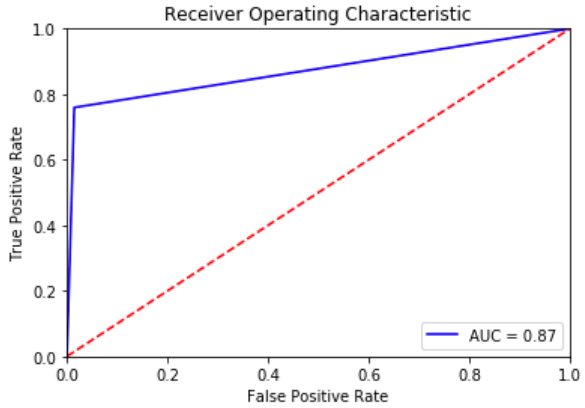
Logistic Regression

First, a vanilla model was constructed, a pipeline with CountVectorizer, TFIDF, and a LR classifier with no adjustments to the defaults. The results were promising, with accuracy 95.9% and F1 score 74.3%. So a grid search was performed on several parameters, and the adjustments yielded improved scores in all categories.

```
Accuracy score:  0.9644065672390024
Precision score:  0.8413173652694611
Recall score:  0.7594594594594595
F1 score:  0.7982954545454546
```

The F-score in particular, was the best of any models tried to that point, and there's better balance between the types of errors.

	Predicted: Negative	Predicted: Positive
Actual: Negative	14226	212
Actual: Positive	356	1124



Logistic Regression with Undersampling

Introducing the RandomUnderSampler to the pipeline, the same pattern that was evident with the other models repeated itself. Again, the accuracy and precision decreased, the Recall increased to surpass precision.

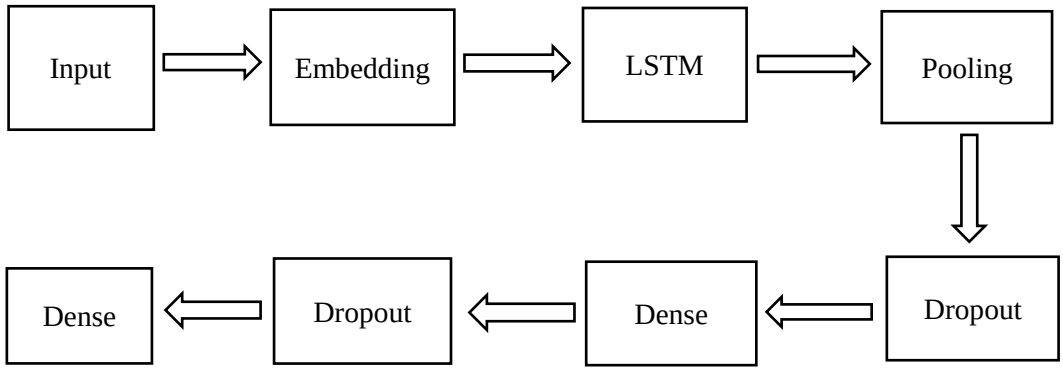
```
Accuracy score:  0.9241132974056899
Precision score:  0.5563468789275241
Recall score:  0.8972972972972973
F1 score:  0.686837341608482
```

The AUC was the same as for the SVM, 0.91.

LSTM

Architecture

With any Neural Network, there are questions about different types of layers, number of layers, number of nodes at each layer. Relying on best practices and guidance by example is prudent. There were several helpful kernels[13] available on Kaggle, that helped me determine a useful architecture.



The data was tokenized, vectorized, and padded with zeros to uniform length, to be fed into the input layer. Next comes an embedding layer, where "words are represented by dense vectors where a vector represents the projection of the word into a continuous vector space."[14] Essentially, a word embedding graphs words into an n-dimensional geometric space where the distance between the word vectors corresponds to the similarity in their meanings. After that, comes the LSTM layer, a pooling layer for sub-sampling, then a dropout layer before a Dense one (using the "relu" loss function), then another dropout before the final fully connected layer with 1 node for the classification. Using the summary() function:

```
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 200)               0
_________________________________________________________________
embedding_1 (Embedding)      (None, 200, 128)          2560000
_________________________________________________________________
lstm_layer (LSTM)            (None, 200, 60)           45360
_________________________________________________________________
global_max_pooling1d_1 (Glob (None, 60)                0
_________________________________________________________________
dropout_1 (Dropout)          (None, 60)                0
_________________________________________________________________
dense_1 (Dense)              (None, 50)                3050
_________________________________________________________________
dropout_2 (Dropout)          (None, 50)                0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 51
=================================================================
```

[13] e.g. https://www.kaggle.com/sbongo/for-beginners-tackling-toxic-using-keras, https://www.kaggle.com/jhoward/improved-lstm-baseline-glove-dropout/code

[14] *How to Use Word Embedding Layers for Deep Learning with Keras*, Jason Brownlee, https://machinelearningmastery.com/use-word-embedding-layers-deep-learning-keras/

cont'd
Total params: 2,608,461
Trainable params: 2,608,461, Non-trainable params: 0

Fortunately, using the Keras library, with Tensorflow as a backend, allowed the author to leverage a GPU for faster training. I originally trained the network for 4 epochs.

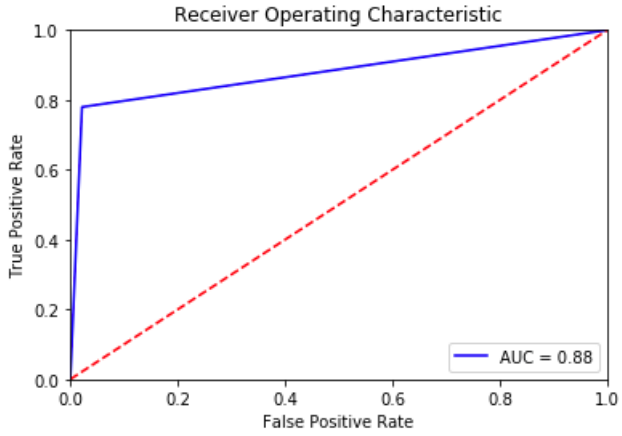
It was necessary to manually map the returned probabilities to 0's and 1's, to calculate the standard metrics.

```
Accuracy score:  0.9598320591552826
Precision score:  0.7855684138869979
Recall score:  0.7797297297297298
F1 score:  0.7826381824347236
```

The accuracy and F1 score aren't quite as high as for the previous Logistic Regression model with Grid Search. But there is good balance between precision and recall. The confusion matrix:

	Predicted: Negative	Predicted: Positive
Actual: Negative	14163	315
Actual: Positive	326	1154

The number of False Positives and False Negatives are very close. The AUC is also best among the non-undersampled models:



Taking a closer look at the output revealed signs of quick overfitting:

After the second epoch, the loss begins to increase, as the training loss continues to fall. I re-fit on just two epochs.

Accuracy score: 0.9639052512846221
Precision score: 0.8889845094664371
Recall score: 0.697972972972973
F1 score: 0.7819833459500378

These are in the same neighborhood, but the LSTM is still outscored by Logistic Regression.

Challenges

• A lot of research on the ins-and-outs of the functioning of algorithms, how different choices are coded, was time-consuming.
• Structuring a Neural Network is something of a dark art, with so many different choices for layers, nodes, and loss functions. I tried a number of suggestions I saw at Kaggle and other webpages addressing text classification.
• There were also a wide range of possibilities for the GridSearch Cross-validation (values listed below), and this and training the LSTM also took a good deal of time.

Refinement

The first run of each algorithm was essentially to establish the best candidates. After it was decided to use Logistic Regression, a Grid Search CV was performed, and some parameters were adjusted that benefitted performance:
• The inverse regularization strength ("C") of the LR model was set to 4, from default 1. The search was on 1, 4, 10, and 100.
• The penalty term was set to the "l1" norm (as opposed to default l2). Note that the l1 tends to eliminate features.
• The vectorization split the text into "bigrams", (two-word chunks), with ngrams_range = (1, 2) (default is (1,1))

As discussed above, a number of experimental adjustments were made to the LSTM network. In particular, Dropout levels between 0.1 and 0.5 were tried, with 0.1 being the choice at the end. The original number of nodes in the first Dense layer was 60, this was reduced to 10 for the last training run.

Results

Model Evaluation and Validation

The chosen model was a pipeline. It first used a TFIDFVectorizer from sci-kit to convert the words into vectors, then assign inverse document frequencies. A grid search cross-validation was run to find suitable parameters. Then a logistic regression model was fitted to the data. Then metrics were produced by comparing its predictions with the test data.

To test the robustness of the model, I predicted on a group of comments taken from a political forum, which I felt could be clearly classified (see notebook: 3 toxic, 6 non-toxic). The algorithm correctly predicted all 9 posts.

Example, correctly predicted non-toxic:

> "Probably the only thing from turning his approval rating to 85%."

Predicted toxic:

> "Bottom line is that Trump is smarter than you... which automatically makes you the dumber idiot."

Justification

Comparing the metrics from the baseline model, Naive Bayes:

```
Accuracy score:  0.9217320466223837
Precision score:  0.9914893617021276
Recall score:  0.15743243243243243
F1 score:  0.2717201166180758
```

To those of the final model, LR pipeline:

```
Accuracy score:  0.9644065672390024
Precision score:  0.8413173652694611
Recall score:  0.7594594594594595
F1 score:  0.7982954545454546
```

There has been marked improvement. The accuracy has improved by 4.26%, the recall by over 60%. While the precision is not as good (76% vs. 99%), that's because the NB model was so close to the *naive model* in being too conservative about classifying as toxic. The F1 score has improved by more than 62%.

Note that the SVM, LR and LSTM models were all very close in their predictions. It was hard to produce much progress after a certain point, for the chosen metrics. Some of this may be owing to the restricted purpose of this version of the classification, i.e. removing the labels for the other classes has *also removed information*.

It's worth looking at a couple of comments which have been mis-classified by LR.

```
False Negative Comment:  please do not write me more!! Silly

Perhaps no one has told you but you're obviously frustrated and I would
like that please do not write me more. For my part I have decided that
I will not bring anything more to wikipedia, because wikipedia is a mon
opoly of information manipulated by a lot of abnormals like you.
```

"Abnormals" is not an everyday insult, and this post has a fair amount of reasonable language as well. Contrast this with:

```
False Negative Comment:  Do not help the jew Schnider. Final warning
```

which is clearly inflammatory, should have been picked up by the algorithm.

```
False Positive Comment:  Ram it up ur a** very hard
till ur eyes water
```

I checked the file manually for this (slightly censored) comment, and it was labeled 'obscene' and not 'toxic', one of very few comments that are. But whether a mistake or categorical decision, this is a case where some information was not available to a classifier working on the full task.

In summation, I would say that the classifier trained in this project is a *good* solution, but not a perfect one. Some of that may be remedied by further effort, some inherent limitations may prevent progress after a certain point.

Conclusion

Reflection

It seems like a daunting problem, having an algorithm comb though textual comments to figure out ones which are toxic. Gratefully, there are well thought-out frameworks for converting the words into numerical vectors, and standardizing approaches to figure the importance of their occurrences in the documents of the corpus. Once these processes have taken place, the standard sklearn algorithms or Keras Neural Networks can operate on them more-or-less cleanly. I felt it was important to try a number of these models, being sufficiently thorough and applying the same metrics to each. At the end, the Logistic Regression model I chose did best on said metrics.

There was an aspect of the study that particularly interested me. The reader might note that I returned repeatedly to testing the algorithms with undersampling, despite the poor overall metrics. This was origainally motivated by the highly imbalanced classes. I felt it was important to see if the drastic increases in recall would form a pattern (they did). My reasons:

(a) It may be that further experimentation with the settings of the imported library might modulate the decrease in precision, while keeping recall high, and yielding a better model.

(b) The *qualitative* significance of recall vs. precision, particular to the aims of this project. A model with high recall tends to classify more comments as toxic, resulting in less false negatives. A model with

high precision tends to have less not-toxic comments flagged (false positives). In fact, if such a model was employed in practice, this type of trade-off might be of great importance, depending on whether the primary goal was to err on the side of flagging anything remotely toxic, or being more resistant to doing so (for example, from fear of being accused of censorship).

Improvement Potential

• More data always helps. In this case, some had to be split off for validation and testing (though the Grid search used Cross-validation). If, at the conclusion of the competition, the labeled test set is released, the algorithm could be re-trained on the entire original training set. There may also be the possibility of augmenting with external data in the future.

• Apply the same strategy, with the same metrics, to the larger problem, and enter the competition. As noted above, the clipping of the original label set may have resulted in some "information loss" inherent to the construction of the dataset.

• More rigorous and in-depth application of undersampling. As mentioned before, the possibility of improving recall, *while holding the precision high*, would give a better model.

• Experimentation with other Neural Network architectures.