Machine Learning Engineer Nanodegree

Capstone Proposal

Patrick Harley
January 22, 2018

Proposal

Domain Background

This project focuses on detecting toxic text comments. The data was gathered from an extensive set of Wikipedia comments made by editors. 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.

This project interested me for a couple of reasons. Having served as a moderator at a large forum in the past, I'm aware of the challenges in determining what type of behavior is "over-the-line". The possibility of automating such decisions is intriguing. Also, having worked as a search engine evaluator for 4 years, I've seen how textual classification algorithms work "behind the scenes" from a data-collection perspective, and was interested in seeing how to implement them myself.

Problem Statement

The problem is to take a text comment posted by a user, and classify it as toxic or non-toxic. The training set provides values for the binary label "toxic", so standard scikit packages[3] can be employed to calculate the numbers of true positives/negatives and false positives/negatives in a test set, which can be viewed in a confusion matrix. These in turn can be used to produce the useful accuracy, precision, recall, and f-score metrics.

Datasets and Inputs

The data was taken from a competition hosted by website kaggle.com[2]. 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.

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.

Solution

The classification algorithm should correctly identify a comment as toxic (1) or non-toxic (0). The labeled data will allow tallies of true positives (classified toxic when toxic), true negatives (non-toxic when non-toxic, false positives (non-toxic when toxic), and false negatives (non-toxic when toxic).

Benchmark Model and Metrics

In the dataset there are 144,277 non-toxic comments, 15,294 toxic ones. Therefore a "dumb 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. This yielded a slightly improved accuracy of 91.8%, with precision 99%. However, the f-score was only 25%, dragged down by a dismal recall of 14%.

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. These observations, and the above scores, are in line with the formulas for precision and recall:

recall = True positive / (True positive + False negative)

precision = True positive / (True positive + False positive)

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.

Project Design

The goal of the project is to build a reliable algorithm to identify toxic internet comments. The given dataset has 159,572 comments and a binary label. Each attempted and adjusted model will use accuracy, recall, precision and f-score as evaluation metrics.

Using the Multinomial Naive Bayes classifier as a benchmark, with no adjustments to the default settings, has yielded slight improvement in accuracy over the "dumb model". However it has shown a particular deficiency, appearing to be too reluctant to classify toxic comments, i.e it displays poor recall. While NB tends to fit quickly, and perform decently as a classifier, it doesn't offer much in the way of parameter adjustment. It would seem wise to consider other algorithms.

Support Vector Machines (SVM's) are another popular choice for text classification. One advantage that SVM's have over Naive Bayes models is a set of adjustable parameters. Using an SVM in combination with a grid search over some of these (parameters) may yield better performance[3], and this attempt should

be made. Like the earlier described attempt with NB, the words from the comments will be vectorized, and assigned inverse document frequency scores in a pipeline.

Logistic regression has also proven to be a good choice for binary classification tasks, including text-based ones[4]. Scikit learn has a number of parameters for hypertuning for its implementation, and this algorithm will also be tried, in conjunction with a grid search.

There will also be an attempt made at constructing a neural network, either a CNN (convolutional), or RNN (recurrent). There has been success using word embeddings[5], a vectorization technique that maps words to a multi-dimensional geometric space, grouping words with similar semantic meanings closer to one another. There exists both the possibility of constructing an embedding from scratch (e.g. using the word2vec[6] technique), or using the pre-trained GloVe[7] embedding. There are ample guidelines at different sites on the web that would facilitate training such a NN.

Notes:

(1) https://conversationai.github.io/
(2) https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge
(3) http://scikit-learn.org/stable/tutorial/text_analytics/working_with_text_data.html
(4) *Python Machine Learning*, Sebastian Raschka, Ch.8, 2016
(5) https://machinelearningmastery.com/best-practices-document-classification-deep-learning/
(6) https://www.tensorflow.org/tutorials/word2vec
(7) https://nlp.stanford.edu/projects/glove/