Conquering Online Toxicity

Austin Adams
Patrick Canny
Liam Ormiston
Jakob Wulf-Eck

Background & Data

Problem

- Based on Kaggle's *Toxic Comment Classification* challenge
  - Part of an initiative by Alphabet's Conversation AI team to protect voices in online conversation
  - Link: https://tinyurl.com/y4afyd7v
- Goal - to programmatically identify *toxic* comments
- Toxic is here defined as **'rude, disrespectful or otherwise likely to make someone leave a discussion'**

Jigsaw, a Google Project

“[The] name, Jigsaw, acknowledges that the world is a complex puzzle of challenges, and it takes a collaborative approach to solve them.” Jigsaw attempts to solve modern global security problems.

- Password Alert
  - Alerts individuals if websites are a trying to phish their password
- Project Shield
  - Helps News Organizations be shielded from devastating DDOS

Jigsaw

Dataset and Formatting

- Our dataset contains the text of approximately 2,000,000 online comments
- Provided by Civil Comments, a defunct commenting plugin for news sites
    - Significantly, encouraged users to vote on whether other users' comments were civil or not.
- The original training set contained a toxicity score, as well as 5 other scores which rate comments on properties such as obscenity or threatening content.
    - Entrants asked only to predict the overall toxicity
- This year's competition dataset adds additional scores indicating whether the comment mentions any of a variety of racial, religious, and gender identities, along with sexual orientations and disability statuses.

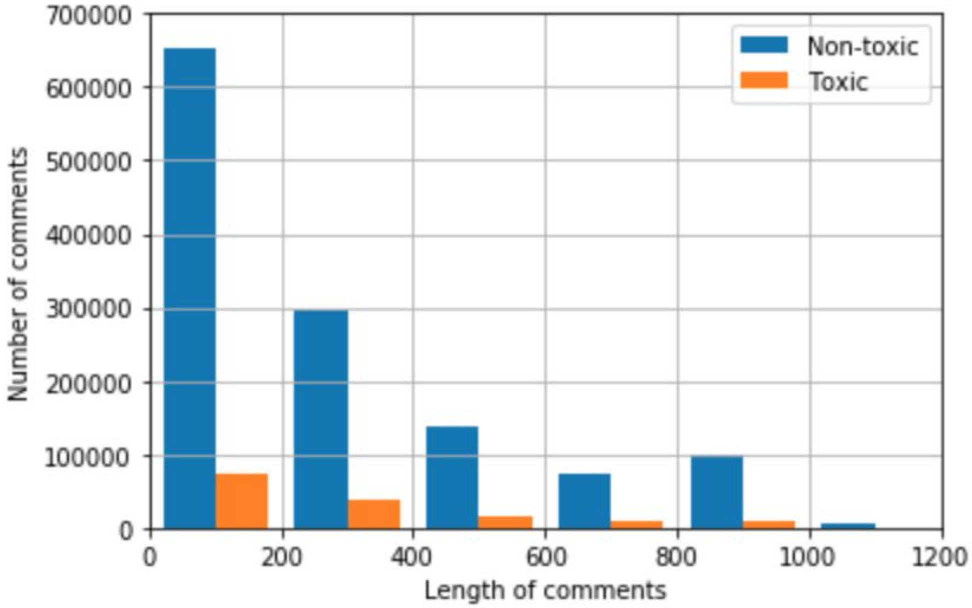
Data Pre-Processing/Exploration

Data Processing

- We sought to understand our data before taking a bigger step
- Looked at frequencies, comment lengths, and tried to extract some insights from the initial dataset
- The hope is that this will help us further develop a roadmap to success

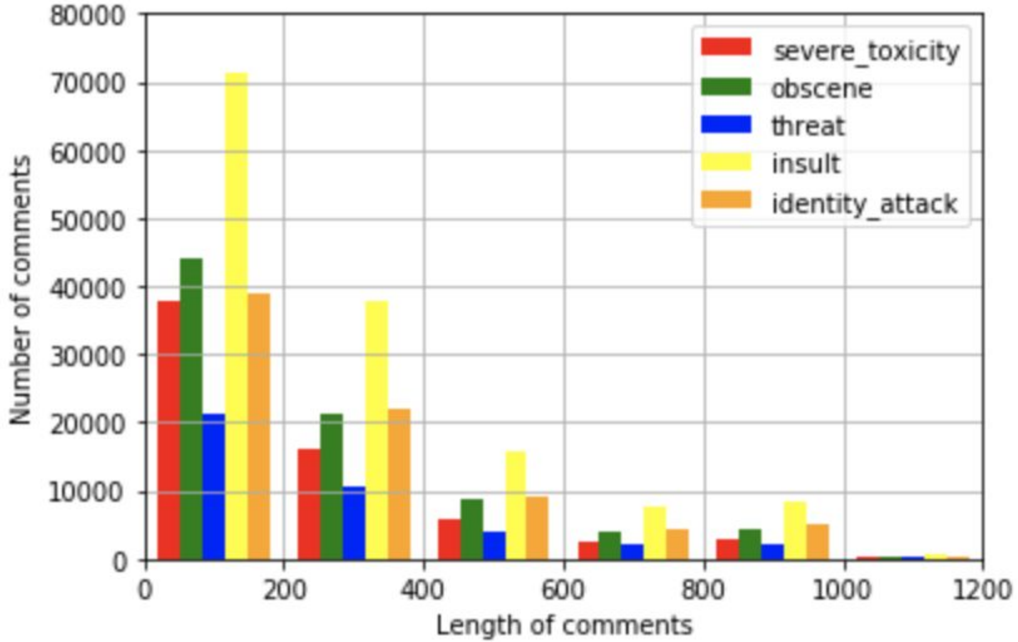
Exploration

```
average length of a toxic comment: 277.613
average length of a non-toxic comment: 287.529
```



Exploration

Comment Length vs Type



Feature Engineering

- **Stemming**
  - Words share common origins, which can be extracted by looking at their stems
  - If we can find common stems of negative words, perhaps we can say with certainty that an unknown word with a known stem is toxic/non-toxic
  - Upshot: very useful for identifying how to classify words we haven’t seen

- **Lemmatization**
  - Words share common meaning
  - Part of speech / intended meaning
  - Useful to weight words that mean that same thing similarly

Feature Engineering

- **Sentiment Analysis**
  - In the future, we will not have all these features available when evaluating a comment
  - Advantageous to extract as much info from the comment itself.
  - Two Key Metrics: Polarity and Subjectivity
  - ***Polarity*** - Is the expressed opinion positive, negative, or neutral?
  - ***Subjectivity*** - A measure of how subjective an expression is
- **Word Classification**
  - Basic process:
    - Gather list of “Stop Words” and remove them from comments
    - Gather list of words in Maximally positive comments - denote them as “Good”
    - Gather list of words in negative comments - denote them as “Bad”

Initial Models

- **Linear Regression**
  - Start off with a statistical method to see how it does
  - Only want to select a subset of features to avoid overfit
  - Aforementioned calculations plus some data about comment reactions
  - Results: Cross-Validation Score 0.03 (+/- 0.01)
    - Not very good at all
- **Random Forest Regression**
  - Tried-and-true ML model, easy to reason about
  - Used the same features as above
  - Also tried looking at every religion as the feature set
  - Results: Cross-Validation Score 0.48 (+/- 0.08)
    - Still not great: only ~50% accuracy

Initial Models

- **Could We Design Our Own Model????**
  - Develop a concept of a ‘word-score’
    - Look at # occurences of a word in toxic vs non-toxic comments
    - Use frequency to produce a naive ‘score’ for any given word: positive scores reflect a more toxic word, negative scores reflect a non-toxic word, stop-words have score 0
  - Look at a comment and use the word scores to produce a score for the whole comment
  - Generally, we could predict whether a comment was toxic or non-toxic, but with very low precision.
  - Going forward, we would try and implement concepts seen in Neural Networks to fine-tune the word scores (see following slides)

Primary Approach

Our Approach: Methodology

- Calculate initial toxicity weights for each word based on a naive metric (appearances in toxic/good comments)
- Preprocess the data, truncating/padding all comments to a standard word length and representing words by their weights
- Instead of using a tokenized sequence of words, use this sequence of word *weights* for each comment as input to a neural network
- Using the neural network, predict the toxicity value of each comment

Our Approach: Tech

- Standard Python ML/data exploration stack
  - Jupyter
  - Pandas/Numpy
  - Matplotlib
- Keras for neural network implementation
- Simple neural network design
  - 200-neuron input layer (to process comment word-weight lists padded to 200 words)
  - Two 1/10 dropout layers to improve generality, interlaced with:
  - Two 50-neuron hidden layers
  - 1 output neuron for toxicity target score
    - Could be extended to predict additional training set parameters

Our Results

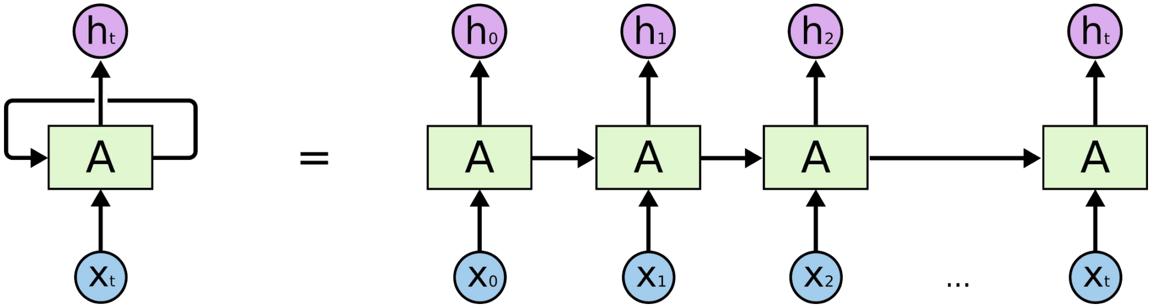
- Binary cross-entropy accuracy is good- ~70% on the test set and ~69% on the validation set
- Higher predictions tend to map to higher toxicity, but predictions don't map well to real toxicity targets
- Word-weight method could likely be improved through use of a premade word sentiment analysis dataset

```
Train on 1624386 samples, validate on 180488 samples
Epoch 1/2
1624386/1624386 [==============================] - 112s 69us/step - loss: 0.3362 - acc: 0.7022 - val_loss: 0.3448 - val_acc: 0.
6879
Epoch 2/2
1624386/1624386 [==============================] - 115s 71us/step - loss: 0.3302 - acc: 0.7022 - val_loss: 0.3449 - val_acc: 0.
6879
```

Other Approaches/Future

Other Approaches

- In place of our word-weight method, we discovered that most highly-rated Kaggle submissions use word embedding
  - Representing words as vectors representing their relative ‘closeness’ in the data’s vocabulary in order to cluster toxic and non-toxic words
- Higher-dimensional nature of an embedding allows for the use of more specialized NN designs
  - LSTM (long short-term memory) - recurrent neural network layer
- These approaches allow for kernels which obtain ~90% accuracy on the test set.
- We implemented a more standard approach in a separate Jupyter notebook



Other Approaches: Methodology

- Tokenize the comments and pad to length 200
  - Map each word to an int, then represent every word by its int equivalent
- Pass to a NN using an embedding layer and an LSTM layer
  - Neural network learns a mapping of words to float vectors which places closely-related words together in vector space
  - LSTM has ‘memory’, learns patterns among long sequences of these vectors - like our comments
- Then run through a standard NN with 2 dropout layers and 2 dense layers to produce a result
- While a baseline version of this approach still produces values that don’t look much like the training labels, it more clearly differentiates toxic and non-toxic comments.

Conclusions

Conclusions

- Classifying comments by toxicity is a challenging problem
- Our word-weighting approach likely falls prey to a problem raised by the creators of the Kaggle competition:
  - How do we avoid bias - that is, models which associate non-toxic words which appear in toxic comments with toxicity?
  - For example, if 90% of comments containing the word ‘gay’ are toxic, how do we avoid flagging comments which use the word ‘gay’ positively?
- The LSTM approach is a good start to answering these questions, but there are improvements to be made
  - Precalculated embeddings, different neural net architectures, different metrics for success...

Conclusions

GitHub: https://git.io/fjGzr

Slide Deck: https://tinyurl.com/y5g9m5c

Any questions?