Matthew Daniel and Cortlandt Bursey-Reece

CS 221

P-Progress

We have decided to primarily focus on detecting and classifying trolls. We will attempt to detect astroturfing if we are completely satisfied with our troll detection. If we get to astroturfing, we will use this paper as a guideline [1]. For troll classification, we have explored Naive Bayes, K-means clustering/K nearest neighbors, and LSTMs and CNNs.

Naive Bayes

As a preliminary model, we implemented a Naive Bayes classifier. We split up tweets from our Russian Troll corpus (250,000+ tweets) and Politician Tweets corpus (90,000+ tweets) into training and test sets. 70% of the tweets (~24K) from the Russian and Politician corpus were put into the training set while 30% (~10K). Before putting these tweets into feature vectors, we stripped the tweets of URLs and emojis.

Feature vectors were created from each tweet by using various methods of feature extraction on the tweets' contents: word extraction, where the tweet content was formatted as a dict mapping words to the frequency of that word's occurrence, and character sequence extraction, where the tweet content was formatted as a dict with each contiguous sequence of *n* characters is mapped to the frequency of that sequence's occurrence.

After being placed into feature vectors, we iterated 50 times while running stochastic gradient descent. On each iteration we also output the training and test error to see how our method did over time. The results can be seen below.

Feature extraction	**Training Accuracy**	**Test Accuracy**
Words	99%	74%
3-grams	95%	58%
4-grams	97%	63%

Exploration

In addition to the Naive Bayes classification we explored in our preliminary research, we also decided to explore clustering algorithms such as k-means clustering and k-nearest neighbors classification. Feature vectors were created from each tweet by using various methods of feature extraction on the tweets' contents: word extraction, where the tweet content was formatted as a dict mapping words to the frequency of that word's occurrence, and character sequence extraction, where the tweet content was formatted as a dict with each contiguous sequence of *n* characters is mapped to the frequency of that sequence's occurrence.

These dicts were treated as sparse vectors by each of the algorithms, and distances between tweets were determined by taking the euclidean distance between sparse vectors. The algorithms were trained on two data sets: a corpus of political tweets from Democratic and Republican politicians, and a corpus of political tweets from confirmed Russian trolls. We found that, while these algorithms were generally accurate when classifying tweets from politicians, both methods struggled with classifying trolls accurately. This is likely in part due to the wider variety of content in the troll tweets (which span wider topics than politics), as well as the higher frequency of emojis, quote tweets/retweets, and hashtags that would make the tweets harder to classify.

The K Nearest Neighbors algorithm was trained on 30,000 tweets, roughly split between troll and politician tweets, and evaluated on 5,000 examples. The K Means Clustering algorithm split the 30,000 tweets into two clusters, the makeup of which are listed below.

K Nearest Neighbors

Feature extraction	**Non-troll accuracy**	**Troll accuracy**	**k**
Words	97%	33%	3
3-grams	98%	23%	3
4-grams	97%	17%	3
5-grams	99%	11%	3
6-grams	98%	7%	3
Words	99%	27%	5
3-grams	98%	11%	5
Words	99%	23%	7

K-Means Clustering

Feature extraction	Cluster #1 makeup	Cluster #2 makeup
Words	83% troll	62% politician
3-grams	58% troll	54% politician
4-grams	58% troll	54% politician

Future Models: LSTMs and CNNs

For our final report, we will try and improve upon the methods we explored early. Furthermore, we will implement at least one of Long Short-Term Memory (LSTM) and Convolutional Neural Networks (CNN). LSTMs would work well for troll-classification since they are a type of recurrent neural network, and recurrent neural networks work well for text classification. The added benefit to LSTMs is their ability to remember values over an arbitrary amount of time. We can use this to remember error and appropriately "cut off" values as necessary [2]. We will implement our LSTM using either Tensorflow or Keras [3]. CNNs, while commonly used in computer vision, are also effective in text classification, as their efficient feature representation, speed, and ease of adding filters give the benefit of easily adding, testing, and tuning classification parameters. We will implement our CNN using Tensorflow [4].

Matthew Daniel and Cortlandt Bursey-Reece

CS 221

References

1. https://www.researchgate.net/publication/317290047_How_to_Manipulate_Social_Media_Analyzing_Political_Astroturfing_Using_Ground_Truth_Data_from_South_Korea
2. https://skymind.ai/wiki/lstm
3. https://rubikscode.net/2018/03/26/two-ways-to-implement-lstm-network-using-python-with-tensorflow-and-keras/
4. http://www.wildml.com/2015/12/implementing-a-cnn-for-text-classification-in-tensorflow/