Patronizing and Condescending Language Detection

Authors: Jay Seabrum, Weitung Liao, Cutter Dalton

Summary:

The following is a report on the approaches utilized in the task of detecting patronizing and condescending language. Our goal for this assignment was to implement different approaches based on complexity. We had three approaches. The first (and our best approach) is our most naive approach in which we anticipated using a simple multi-layered perceptron as the baseline. The code for this approach is submitted with this document. The second approach attempted the classification task using a pre-trained model (Google's BERT). The third approach was via feature extraction, both with TF-IDF and a CRF approach. Both of these approaches availed to nothing as they didn't have any sort of predictive power behind them in this task. The code for the second and third approaches is the one web link found in this document.

Simple Approach (MLP): Troubleshooting Upsampling/Downsampling:

Our first attempt at establishing a baseline for this task was through the implementation of a multi-layered perceptron. Without adjusting the unbalanced dataset, our MLP approach, with 100 hidden layers each utilizing ReLu as an activation function, produced the following metrics:

```
In [16]: prec
Out[16]: 0.4946236559139785

In [17]: rec
Out[17]: 0.21800947867298578

In [18]: f1
Out[18]: 0.3026315789473684
```

The results seemed decent enough given the conditions that we decided to try the recommended approaches in adjusting the unbalanced data. First we only downsampled the majority class. We attempted, in each alteration in the sample, to have the ratio between the majority and minority class be roughly 80-20. In this case, we had the same number of

condescending examples (993), but only 5,476 non-condescending examples (originally about 9,000). As can be seen, this approach worked marginally better:

```
prec
```

```
0.5307692307692308
```

```
rec
```

```
0.3150684931506849
```

```
f1
```

```
0.3954154727793696
```

Upsampling proved to be a little treacherous. Our implementation of upsampling involved duplicating a certain number of times each example of condescending language (ie: the positive cases in the dataset). In this iteration we duplicated each positive sample 20 times to bring its total sample size to 19,860 and kept the total number of negative samples the same. This proved to be completely ineffective, as there was an overwhelming majority of positive samples which were direct copies. This let the system train on the same data it would test on, and led to an almost perfect classifier:

```
In [15]: prec
Out[15]: 0.9712106299212598

In [16]: rec
Out[16]: 1.0

In [17]: f1
Out[17]: 0.9853950817625765

In [ ]:
```

In response, we scaled down the duplication to just once per sample, and downsampled the negative majority class by 700 samples. This yielded our “best” results, however, there is the possibility that the results are the consequence of the same issue as above where there was overlap between training and testing samples. Given that the MLP was fairly uncomplicated,

and most feature extraction was done through only word counts, we have a hard time fully accepting its success.

We did, however, learn which data balancing approach to rely on for more reliable results.

```
prec
```

```
0.8111111111111111
```

```
rec
```

```
0.8538011695906432
```

```
f1
```

```
0.8319088319088318
```

Pre-trained model (BERT):

Link to Google Collab:

https://colab.research.google.com/drive/1amCjBH6v4xffFJFVIeZYx2YgkbfLFdD3?usp=sharing

At first, we used a pre-trained BERT model to do classification without doing any preprocessing and or dealing with down/upsampling the unbalanced data. The train-test data split is around 90-10 The result accuracy is around 91%, which is almost the same accuracy of just guessing negative for all data.

After this, we tried downsampling with a different number of minority/majority samples. The original dataset had a distribution of 91-9 for negative-positive samples. The first attempt we tried with downsampling was to make the minority.majority samples 50-50. This resulted in an accuracy of around 89%, which is even worse than just guessing negative for the entire dataset. Beside that, we found that the validation loss increases by epoch.

```
***** Running training *****
  Num examples = 1774
  Num Epochs = 8
  Instantaneous batch size per device = 8
  Total train batch size (w. parallel, distributed & accumulation) = 8
  Gradient Accumulation steps = 1
  Total optimization steps = 1776
```

[1776/1776 17:55, Epoch 8/8]

Step	Training Loss	Validation Loss	Accuracy
100	0.662500	0.571123	0.703916
200	0.537200	0.607142	0.692455
300	0.451500	0.585578	0.718243
400	0.433300	0.301387	0.896848
500	0.266600	0.666201	0.831901
600	0.277000	0.646034	0.828080
700	0.163600	0.506575	0.890162
800	0.141300	0.701108	0.847182
900	0.132800	1.171003	0.765043
1000	0.056500	0.825644	0.833811
1100	0.060900	1.309641	0.762178
1200	0.041900	1.021398	0.811843
1300	0.025500	1.137661	0.804202
1400	0.005900	1.430225	0.778415
1500	0.008100	1.244591	0.797517
1600	0.000400	1.303504	0.793696
1700	0.000400	1.367824	0.793696

When we increased the majority-minority class distribution to 75-25, the results got slightly better, of which an accuracy of 91% was achieved.

```
***** Running training *****
  Num examples = 2661
  Num Epochs = 6
  Instantaneous batch size per device = 8
  Total train batch size (w. parallel, distributed & accumulation) = 8
  Gradient Accumulation steps = 1
  Total optimization steps = 1998
```

[1998/1998 15:36, Epoch 6/6]

Step	Training Loss	Validation Loss	Accuracy
100	0.602600	0.461181	0.729704
200	0.479200	0.415253	0.817574
300	0.567300	0.243079	0.910220
400	0.455900	0.793722	0.748806
500	0.327300	0.522468	0.812798
600	0.341000	0.305181	0.894938
700	0.239500	0.916804	0.806113
800	0.127900	0.760115	0.840497
900	0.162100	0.486278	0.888252
1000	0.105000	0.518528	0.889207
1100	0.026300	0.580162	0.896848
1200	0.089800	0.531385	0.894938
1300	0.045500	0.529098	0.894938
1400	0.026800	0.732504	0.871060
1500	0.011000	0.682782	0.880611
1600	0.009000	0.713683	0.882521
1700	0.009200	0.707816	0.886342
1800	0.007200	0.710487	0.887297
1900	0.000300	0.719688	0.887297

```
***** Running Evaluation *****
```

Feature extraction, two attempts:

We guess that, in the same category, high PCL data may have some shared words/features that commonly appear in the high PCL data text. So we try to use TF-IDF to find out if PCL data has common words that occur with the keyword given that the sentence is labelled as PCL. When looking at the frequency of the distribution of the words with regards to the keywords in both positive and negative samples, we found that there is no strong correlation of certain words appearing between non-PCL and PCL sentences. We also attempted to do a CRF approach

with these data to confirm if what we found in the TF-IDF approach was true. The CRF approach confirmed that the words that appear near the keyword didn't matter and therefore did not improve any sort of predictive power in this binary task. We also found that there was no correlation between the country of origin of the sentence and the rate at which PCL occurs in the sentences:

```
Country: AllData Pos: 0.0949     Neg: 0.9051
Country: hk      Pos: 0.0592     Neg: 0.9408
Country: nz      Pos: 0.0907     Neg: 0.9093
Country: ca      Pos: 0.0868     Neg: 0.9132
Country: sg      Pos: 0.071      Neg: 0.929
Country: jm      Pos: 0.1163     Neg: 0.8837
Country: au      Pos: 0.0684     Neg: 0.9316
Country: za      Pos: 0.1056     Neg: 0.8944
Country: lk      Pos: 0.0972     Neg: 0.9028
Country: ke      Pos: 0.0835     Neg: 0.9165
Country: pk      Pos: 0.0972     Neg: 0.9028
Country: ph      Pos: 0.1229     Neg: 0.8771
Country: gh      Pos: 0.1434     Neg: 0.8566
Country: my      Pos: 0.0788     Neg: 0.9212
Country: us      Pos: 0.081      Neg: 0.919
Country: ng      Pos: 0.1341     Neg: 0.8659
Country: in      Pos: 0.0736     Neg: 0.9264
Country: gb      Pos: 0.1056     Neg: 0.8944
Country: ie      Pos: 0.0968     Neg: 0.9032
Country: tz      Pos: 0.0988     Neg: 0.9012
Country: bd      Pos: 0.0859     Neg: 0.9141
```