Detection of Patronizing and Condescending Language
Task4: Track 1

Matthew Dyer
December 14, 2021
University of Colorado, Boulder

Abstract

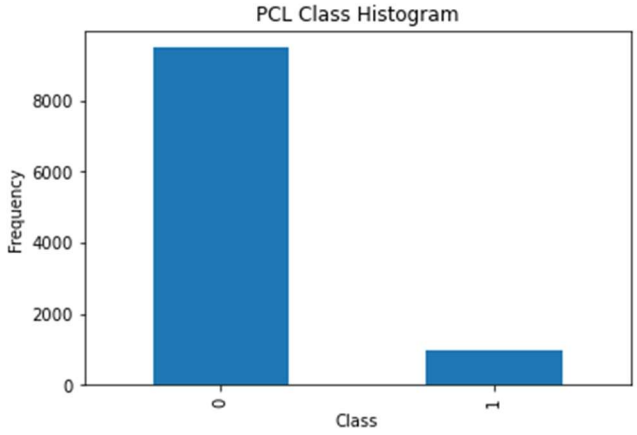
Patronizing and condescending language is difficult to classify and can harm vulnerable groups. This analysis pertains to the patronizing and condescending language (pcl) detection task (task 4). In particular, the main objective is to detect whether a paragraph contains pcl language outright (track 1). In this analysis, I used stemming, lemmatization, and extraneous word/character removal first to preprocess the data. I then created different word embeddings through tf-Idf and word2vec. I used SMOTE to balance the data before passing it through four different models. These four models included logistic regression on both word embeddings, Naïve Bayes classifiers on the Tf-Idf word embeddings, and a neural net on the word2vec word embeddings.

Exploratory Data Analysis

The dataset under analysis was provided by SemEval and inspired by the paper "Don't Patronize Me! An annotated Dataset with Patronizing and Condescending Language Towards Vulnerable Communities" (Perez-Almendros et al., 2020). It contains 10,469 paragraphs mentioning vulnerable communities taken from various media in 20 different English-speaking countries. Each paragraph contains a paragraph id, keyword, country code and a paragraph labeled from 0-4 denoting the level of pcl.

In my analysis, I considered the labels 0 and 1 to be non-pcl (denoted 0) and the labels 2-4 to be pcl (denoted 1). This is consistent with the binary classification described in the paper. I also only included the paragraph and label in my analysis while disregarding the other features for now. It is worthwhile to note, however, that pcl may differ depending on the group of vulnerable people or the culture of the country that the paragraph came from. Therefore, the keyword and country code could potentially be useful predictors that could further augment the model.

The primary difficulties of the dataset are the relatively small sample size, subtlety of the pcl, and the composition of the classes. The data are highly unbalanced. Only 9.5% of the data are pcl with 9,476 examples of non-pcl paragraphs and 993 examples of pcl paragraphs. Therefore, there is a significant risk that any NLP model will learn primarily about the dominant class when the minority class is the true class of interest. It also indicates that accuracy is not an effective evaluation metric as simply predicting 0 every time (for non-pcl) would produce an ineffective model that still is correct roughly 90% of the time. Instead, I used F1 scores to evaluate my models. The following histogram shows the distribution of the unbalanced date (0=non-pcl, 1=pcl):



Otherwise, there were almost no issues with missing data. Paragraph 8639 produced a missing text and was the lone datapoint excluded from analysis.

For exploratory data analysis, I first looked through the examples of pcl paragraphs to try to identify words that may appear more often. There were certain words that seemed to be overrepresented in the pcl data. For example, 'homeless' appeared in 19.8% of the pcl paragraphs compared to 10.3% of the non-pcl paragaphs. Also 'donate' appears 10 times more in pcl paragraphs compared to non-pcl paragraphs (2.2% vs. 0.28% respectively). This may or may not be statistically significant. It could hint that homeless people are frequent victims of pcl or that donating

typically indicates an air of superiority for the group donating. However, it could also be insignificant coincidence. It at least indicates that the presence of certain words could work as reasonable features in an NLP model.

Text Preprocessing

Before converting the text to numerical feature vectors, I first manicured the text into a cleaner format. I did not want my model to differentiate similar tokens like 'homeless' and 'homeless!' simply because of a difference with punctuation. Similarly, words that are capitalized would be treated differently than the same word with no capitals. Therefore, I first used the re and string libraries to remove punctuation, capitalization, numbers, extra white spaces, and other extraneous special characters.

Next, I removed a list of 'stopwords' defined in the nltk library. These 'stopwords' included frequent but uninteresting words that would have undue and unhelpful influence on the modelling. For example, 'a' and 'the' are frequently used articles but do not have any interesting unique meaning that could have predictive value.

In order to have the model treat slight variations of the same word as a single feature, I then used the nltk library for stemming and lemmatization. Stemming removes the affixes and suffixes of a word so that words like 'running' would be reduced to 'run'. Lemmatization reduces the word to its baseform so that words like 'am' and 'be' would be reduced to 'are'. These methods help the model group similar words that really carry similar meaning or function but have slight differences like tense or conjugation. For NLP models, however, these words should be treated the same.

After stripping special characters, removing a set of stopwords, stemming, and lemmatization, I created 'clean text'. An example transformation is illustrated in the following lines with paragraph 13.

Before pre-processing:

```
'" Ghostbusters " is a resurrection of the 1984 hit film but this time the
leads are women , not men .'
```

After pre-processing ('clean text'):

```
'ghostbusters resurrection hit film time lead woman men'
```

Word Embedding

With clean text, I then converted the words to numerical vectors so that they could act as input in machine learning models. The first form of word embedding I used was a bag-of-words model. These models, in essence, represent the counts of how many times different words appear. They are called 'bag-of-words' because they do not take the order of the words into account and only consider the frequency like all the words were haphazardly tossed into a bag. Rather than only using the counts, I vectorized the paragraphs using the term frequency-inverse document frequencies (tf-Idf). This method increases the value of a word in a paragraph's vectorization based on the number of times that word appears in that paragraph but decreases the value of that word based on its overall frequency in the entire body of text. Common words will be weighted less while rarer words will be weighted more. Therefore, uninteresting words that do not signify unique meaning will not heavily influence the model

like in the removal of stopwords. The resulting word embeddings were sparse vectors of length 24,281 with each entry corresponding to a unique word of the vocabulary of the clean text.

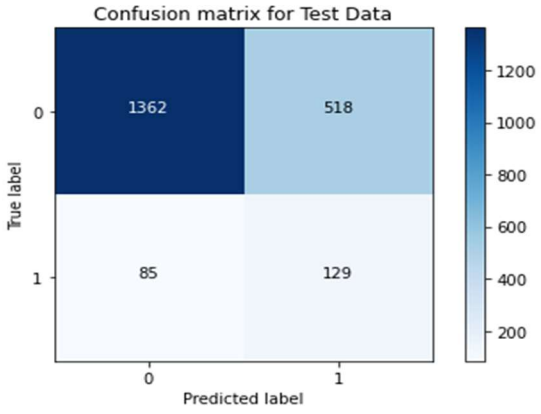
The other word embedding I used was through a word2vec algorithm using the gensim library. This method uses a neural net to learn word embeddings that capture the meaning of words in context. Therefore, they do not treat the words as individual and independent entities like in bag of words. Instead, word2vec considers the words that cooccur often and have related meaning. The resulting word embeddings have a numerical representation that are 'close in space' to similar words. Finally, word2vec reduces the dimensionality of the vectors resulting in non-sparse vectors of length 100 to reduce redundant information. Finally, I normalized these word2vec embeddings to reduce undue influence with any of the features and promote better behavior in the model fitting.

After transforming the text into word embeddings, I randomly separated 80% of the data into a train set to be used in model fitting and the remaining 20% of the data into a test set for model evaluation. Next, I balanced the training dataset by oversampling the minority class. With severely unbalanced data, machine learning algorithms tend to overanalyze the majority class and disregard the minority class. In order to balance the data, I synthesized novel pcl data for the training set using synthetic minority over-sampling technique (SMOTE). This technique randomly selects data in the minority class and its nearest neighbors in the minority class. Then, SMOTE creates a new datapoint randomly between these datapoints in high-dimensional space. Therefore, redundant samples are avoided and realistic new samples are created. Before SMOTE, there were 779 samples of pcl paragraphs and 7,595 samples of non-pcl paragraphs in the training data. After SMOTE, there were 7,595 samples of both classes in the training data. I did not use SMOTE for the test data since the test data needs to resemble the actual split of data that occur in reality.

Model 1: Logistic Regression Using Word2vec Embeddings

For the first model, I used the word2vec embeddings in a logistic regression model. Since these word embeddings contain 100 features, I used ridge regression to regularize the model to avoid issues with collinearity. This ridge regression model uses an L2 penalty which penalizes the sum of the square of coefficients and results in shrinkage of these coefficients in fitting. This helps manage collinearity and mitigates overfitting. In order to optimize my logistic regression, I used GridSearchCV to run 5-fold cross-validation on the L2 penalty hyperparameter. I evaluated integers 1-20 for hyperparameter C (the inverse of the L2 penalty) and found the optimized loss to be with C=10. Model 1 produced the following F1-values and confusion matrix:

```
F1 metric in the train dataset: 0.7367
F1 metric in the test dataset: 0.2998
```



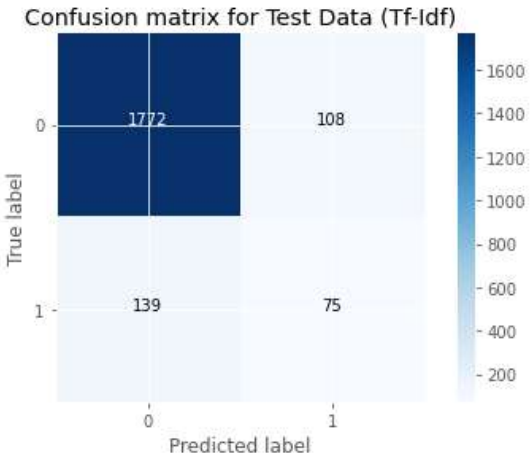
Overfitting, which persists as a problem for all the models, is clearly apparent with the high F1 value of the training set. The F1 metric for the test data was 0.3 and the model appeared to have the most trouble with false positives.

Model 2: Logistic Regression with tf-Idf

I repeated the process above using the tf-Idf vectors instead. While the context added in word2vec is desirable, the dataset under analysis is small and may not be large enough to justify

word2vec. Therefore, I wanted to compare the same model using tf-Idf. Using GridSearchCV and 5-fold cross-validation, I looked at the loss of C from 10 evenly spaced values between 1-30. In this case, the best C hyperparameter was 26.78. This model produced the following results:

```
F1 metric in the train dataset: 0.9998
F1 metric in the test dataset: 0.3778
```

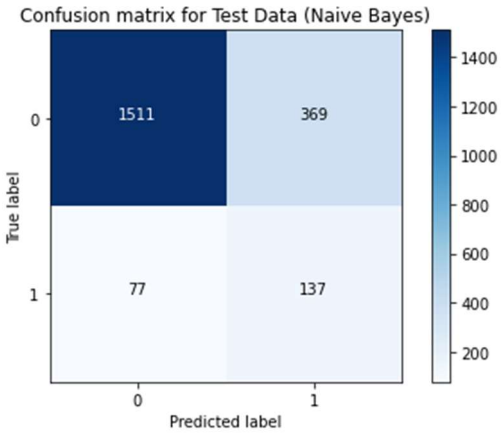


This model had extraordinary overfitting. It correctly estimated all but 3 observations in the training data for a F1 of 0.9998. The results on the test data were more promising, though, with an improved F1 of 0.38. However, it appears that the bag of words vector representation leads the model to simply memorize the training data.

Model 3: Naïve-Bayes Classifier with Tf-Idf

In the next model, I used a naïve-bayes classifier using the bag of words vectors. This classifier uses Baye's rule and independence assumptions to classify the data. This model produced the following results:

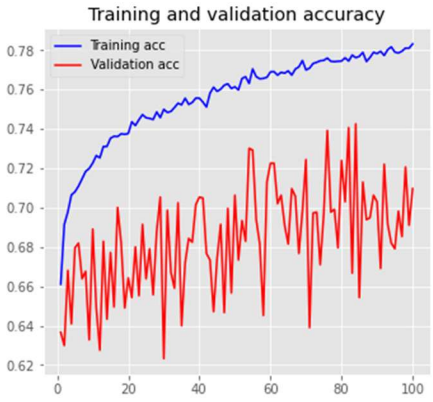
```
F1 metric in the train dataset: 0.9409
F1 metric in the test dataset: 0.3806
```

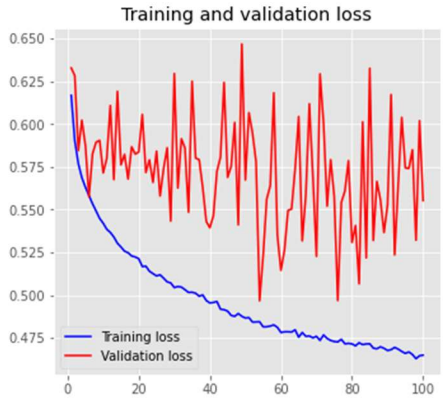


This model also suffers from severe overfitting, although not as egregiously as the logistic regression model. It seems to perform slightly better with the test data while producing a 0.38 F1.

Model 4: Feed-Forward Neural Network with word2vec Embeddings

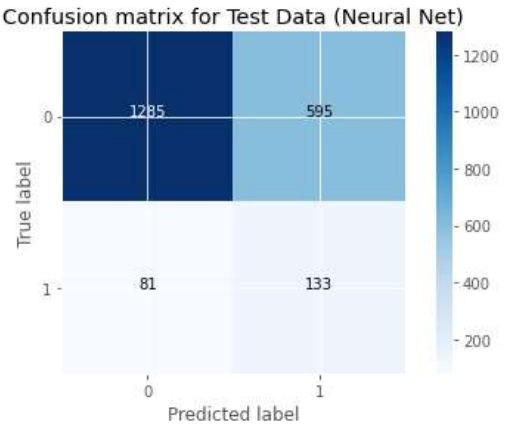
More complicated models, such as neural networks, may not be suitable for this small dataset. The immense amount of required parameters to be learned need a plethora of data and this dataset only includes around 10,000 paragraphs. Fitting neural networks could greatly benefit from finetuning a BERT model to overcome the small sample size. Nevertheless, I tried to fit a small feed-forward neural network to assess the results. This neural network had 1 hidden layer with 7 nodes using a reLu activation function for the hidden layer and using a sigmoid activation function for the output node. I used the smaller word2vec embeddings which were already learned through a neural network. This model resulted in 715 parameters needing to be estimated. I first ran the neural net through 100 epochs to assess when the model started to overfit with diminishing returns. There was no clear cutoff point but the training and validation plots seem to show diminishing returns after 60 epochs. The erratic behavior of the validation set indicates that this model may not be effective for these data.





After refitting the model with 60 epochs, the results were the following:

```
F1 metric in the train dataset: 0.8038
F1 metric in the test dataset: 0.2824
```



The neural network produced a poor F1 measure for the test data at 0.28 and seemed to have the largest struggle with false-positives. This model also suffered from overfitting albeit with a smaller F1 score than the two tf-Idf models.

Conclusion

The best model appears to be the Naïve-Bayes model using the tf-Idf word vectors. It scored the highest F1 value of 0.38 on the test data. In many ways, this model was the most simple as it assumed each paragraph was an independent probabilistic sequence of words. Pcl is complicated to detect since, by definition, the users of the language are even mainly unaware they are being patronizing or condescending. Therefore, most of the models struggled. While the NB-classifier performed the best, it seemed to overfit the training data immensely. This was a common issue in both models using Tf-Idf word embeddings.

However, the word2vec models produced lower F1 scores. This might be because the dataset is too small to produce effective word2vec word embeddings. For future research, these models could be improved by use of BERT's pre-trained models and fine-tuning. A recursive neural net could also improve on the feed-forward neural net by introducing context. However, this model would also need ample parameters to be estimated with a small dataset.

In conclusion, pcl is difficult to classify which is why it is so pervasive in our media. Even with human classifiers, there were several disagreements in the correct pcl labels between the annotators. My Naïve-Bayes classifier performed the best but still misclassified paragraphs as pcl more than twice the amount it correctly identified pcl paragraphs. This difficult task will require great innovations in natural language processing and effective feature selection. For now, the strides in fine-tuning may prove to be the most promising to tackle the intractability of patronizing and condescending language.