Binary Classification Model:
Identification of Sexism in Text

Authors:
Nathan Oswald, Jay Whitney, Kory Smith

Input:

Provided was a set of labeled texts that fell under two complementary categories: sexist or non-sexist. The data was separated into the training and test sets. Before training the models, the text was processed in three unique ways for the model to analyze:

1. Count: Raw word count from the text while eliminating common English stop words.
2. TF-IDF: Similar to Count, but gives words weights based on the frequency across texts to value certain words more than others.
3. BERT: A complicated way of encoding text in a way that attempts to grasp the full meaning and context of the text.

Both the test and training sets were tokenized/encoded in these ways and stored for future model use.

Models:

Initially, a variety of classification/regression models were used and tested for this project. The models that were experimented with are:

1. Logistic Regression
2. SGD
3. XGB

Initial Results

P: Precision
R: Recall
0: Not Sexit
1: Sexist
W: Weighted

Classifier	Process	P (0)	R (0)	F1 (0)	P (1)	R (1)	F1 (1)	P (W)	R (W)	F1 (W)
LogReg	TF-IDF	0.8	0.95	0.87	0.75	0.36	0.49	0.79	0.79	0.766
LogReg	Count	0.83	0.91	0.87	0.67	0.49	0.57	0.78	0.79	0.783
LogReg	BERT	0.82	0.85	0.83	0.55	0.5	0.52	0.75	0.75	0.748
SGDClassifier	TF-IDF	0.81	0.93	0.87	0.7	0.41	0.52	0.78	0.79	0.772
SGDClassifier	Count	0.83	0.81	0.82	0.52	0.55	0.53	0.74	0.74	0.74

SGDClassifier	BERT	0.77	0.95	0.85	0.62	0.23	0.33	0.73	0.75	0.707
XGBClassifier	TF-IDF	0.81	0.93	0.87	0.7	0.44	0.54	0.78	0.79	0.777
XGBClassifier	Count	0.82	0.95	0.88	0.77	0.46	0.58	0.81	0.81	0.798
XGBClassifier	BERT	0.79	0.9	0.84	0.59	0.36	0.45	0.73	0.76	0.74
Average		0.81	0.91	0.86	0.65	0.42	0.5	0.77	0.77	0.759

What we can see is that, on average, without much additional work, the different models and text processors do a pretty good job at classifying non-sexist tweets with an average of 0.81 precision and 0.91 recall. However, the classification of what is sexist struggles. For the sake of this problem, we want to bring up the model's ability to classify what is actually sexist. We can be fairly confident that if the text is not sexist, but if the test is sexist, we're almost guessing 50/50.

First Adjustment:
One thing I noticed was that there were a lot more non-sexist texts in the test sets than sexist texts. LogisticRegression in Python has a parameter called "class_weight" which I set to balanced. The results were as follows:

Classifier	**Text Processing**	**Precision (not sexist)**	**Recall (not sexist)**	**F1 (not sexist)**	**Precision (sexist)**	**Recall (sexist)**	**F1 (sexist)**	**Precision (weighted)**	**Recall (weighted)**	**F1 (weighted)**
LogReg	TF-IDF	0.8	0.95	0.87	0.75	0.36	0.49	0.79	0.79	0.766
LogReg	Count	0.83	0.91	0.87	0.67	0.49	0.57	0.78	0.79	0.783
LogReg	BERT	0.82	0.85	0.83	0.55	0.5	0.52	0.75	0.75	0.748

Classifier	Text Processing	Precision (not sexist)	Recall (not sexist)	F1 (not sexist)	Precision (sexist)	Recall (sexist)	F1 (sexist)	Precision (weighted)	Recall (weighted)	F1 (weighted)
Balanced LogReg	TF-IDF	0.87	0.77	0.82	0.53	0.7	0.6	0.78	0.75	0.757
Balanced LogReg	Count	0.86	0.81	0.84	0.57	0.65	0.6	0.78	0.77	0.772
Balanced LogReg	BERT	0.85	0.74	0.79	0.49	0.67	0.56	0.76	0.72	0.73

What we can see is that the classification for sexist texts performs a little bit better with the balanced parameter, while the classification for non-sexist texts takes a bit of a hit. This is more balanced and probably better for what we want, but we can do better.

Classifier	Text Processing	Precision (not sexist)	Recall (not sexist)	F1 (not sexist)	Precision (sexist)	Recall (sexist)	F1 (sexist)	Precision (weighted)	Recall (weighted)	F1 (weighted)
LASSO (unoptimized)	Count	0.73	1	0.84	0	0	0	0.53	0.73	0.611
LASSO (optimized)	Count	0.82	0.96	0.89	0.82	0.44	0.58	0.82	0.82	0.802
LASSO (Filtered)	Count	0.82	0.97	0.89	0.84	0.45	0.59	0.83	0.83	0.808
LASSO (Filtered + < 0.373	Count	0.87	0.89	0.88	0.68	0.65	0.67	0.82	0.82	0.821

Running LASSO (unoptimized) with a default alpha of 0.5 using Count performed meh.
Optimizing the alpha showed a significant difference.
Filtering the stop words had a smaller increase.
Reducing the threshold got us to .82.