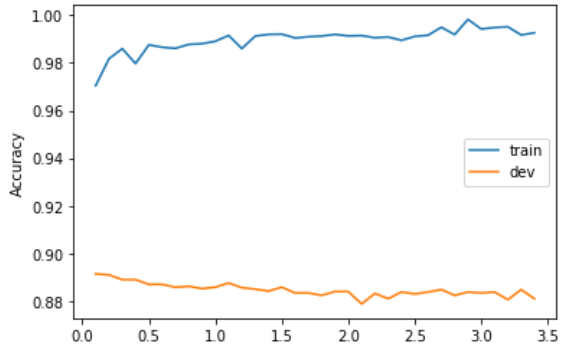
Sec B Team 8

Kaggle Team name: Sec B Team 8

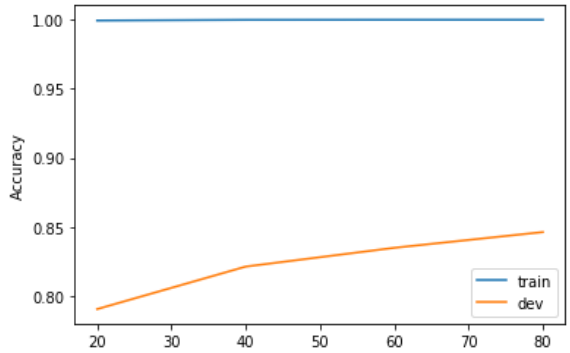
Best LR model: lr_default, accuracy = 0.880 , Best RF model: rf_default, accuracy = 0.841

I. Default features

Best for LR: cs = np.arange(0.1, 3.5, 0.1) , accuracy = 0.880



Best for RF: n_estimators = np.arange(20,100,20) , accuracy = 0.841



First, to optimize the hyper-parameters of logistic regression, I modified get_tuned_lr to several ranges, and compared the plots they generated. The final range I picked was cs = np.arange(0.1, 3.5, 0.1), because the accuracy is high, and also it offers the smallest difference between the training and testing accuracy, so we can better generalize our model for unseen data.

Second, to optimize the parameters for random forest, I also modified get_tuned_rf. N_estimators for several times, and I found as the number increased, the accuracy also increased, so the optimal range with highest accuracy I found was actually (50, 500, 50), all of its accuracy scores are 1 for the train dataset. But bigger numbers take a very long time to run, so I start decreasing the numbers and compare the plots, then the final range I got was (20,100,20) .

II. Custom features

lr_custom: accuracy = 0.823, rf_custom: accuracy = 0.837

I used 'pos_count': self.pos_count(review), and 'neg_count': self.neg_count(review) as one group, then tried different combinations with others like word_count. Then I tried CountVectorizer() with different ngram_ranges, and compared the results to TfidfVectorizer() with different ngram_ranges also. I found there wasn't a big difference between CountVectorizer() and TfidfVectorizer(), and the accuracy increases only a little bit when changing the ngrams, but runs very slow, so I just used CountVectorizer() with default ngram ranges.

III. Analysis

Classifier1 Metrics
Accuracy: 0.9546
Precision: 0.9534753810549836
Recall: 0.95584
F1: 0.9546562262794135

Classifier2 Metrics
Accuracy: 0.96924
Precision: 0.9674822666772934
Recall: 0.97112
F1: 0.9692977202858625

Classifier3 Metrics
Accuracy: 0.83808
Precision: 0.8276476973174135
Recall: 0.854
F1: 0.8406173714465706

Classifier4 Metrics
Accuracy: 0.96872
Precision: 0.9683453237410072
Recall: 0.96912
F1: 0.968732506997201
The best performance classifier is Classifier 2 in terms of both accuracy score and precision score, the classifier is rf_default.

As for the eli5 charts, the first image below is the lr model and the second one is rf. Comparing the two images, we can find, in lr model, positive features have higher weights, for instance, “excellent”, “ perfect” and “funniest” are at the top with highest wrights, while negative sentiments like “worst” and “waste” are at the bottom with lowest weights.
However, for the rf model, most of the negative sentiment features have high weights, like “bad” and “worst” are the two tops with highest weights.

```
In [23]: eli5.show_weights(classifier1, top=25)
```

Out[23]: **y=1** top features

Weight[?]	Feature
+0.782	excellent
+0.725	perfect
+0.715	funniest
+0.701	superb
+0.682	refreshing
… 34420 more positive …	
… 33966 more negative …	
-0.668	lame
-0.675	lacks
-0.684	unfortunately
-0.716	worse
-0.722	poor
-0.729	laughable
-0.730	dull
-0.733	disappointing
-0.735	ridiculous
-0.740	save
-0.742	badly
-0.757	avoid
-0.795	mess
-0.844	horrible
-0.881	boring
-0.954	poorly
-1.031	disappointment
-1.087	awful
-1.198	worst
-1.336	waste

```
In [25]: eli5.show_weights(classifier2, top=25)
```

Out[25]:

Weight	Feature
0.0105 ± 0.0297	bad
0.0093 ± 0.0233	worst
0.0071 ± 0.0169	great
0.0049 ± 0.0140	waste
0.0047 ± 0.0141	awful
0.0042 ± 0.0048	and
0.0034 ± 0.0078	no
0.0031 ± 0.0024	the
0.0030 ± 0.0073	nothing
0.0030 ± 0.0083	boring
0.0029 ± 0.0082	terrible
0.0029 ± 0.0075	acting
0.0028 ± 0.0080	excellent
0.0027 ± 0.0025	of
0.0027 ± 0.0022	is
0.0026 ± 0.0069	money
0.0026 ± 0.0057	just
0.0025 ± 0.0073	minutes
0.0025 ± 0.0058	wonderful
0.0025 ± 0.0054	plot
0.0025 ± 0.0044	best
0.0025 ± 0.0026	this
0.0025 ± 0.0027	was
0.0024 ± 0.0077	worse
0.0024 ± 0.0021	in
… 68385 more …	

As for the errors generated in the notebook, cases where lr_default and rf_defualt disagree is 5.008%, cases where rf_default and lr_custom disagree is 15.572%, cases where lr_custom and rf_custom disagree is 15.774%, and cases where lr_default and rf_custom disagree is 5.06%. Generally, there's less errors between the two default classifiers, and also less errors in the random forest classifiers. In terms of the impact of custom features, current custom features did not help a lot in reducing the errors, so future actions could better transform the data and optimize different features to reduce the errors.

IV. Statement of Collaborations

We followed the academic honesty guidelines posted on the course website. We organized to discuss the task descriptions, requirements, bugs in our code, and the relevant technical content *before* we start working on it.