Exploring Discourse on Reddit

feat. /r/TwoSentenceHorror, /r/TwoSentenceSadness, /r/OCPoetry, and /r/haiku

Introduction

Our goal is to create two classifiers which individually classify Reddit posts as belonging to one of two subreddits. One is a support vector classifier and the other is a random forest classifier. During the data collection progress, we only used non-stickied, non-edited, and non-crossposted posts. This is because stickied posts are meta-posts, edited posts provide complications to the content of the post, and cross posts contain media.

Originally, we pondered to see if the two classifiers could accurately classify between posts from two subreddits that share very similar post structure: /r/TwoSentenceHorror and /r/TwoSentenceSadness. In this context, the structures of the posts were similar in that they were each limited to two sentences (or, are supposed to follow this rule). Due to this, the only real difference between the posts were in their meanings, i.e., the semantics of the posts, where one is horror-focused and the other sad and depressing. We soon found that differentiating between the stark semantics of the two subreddits was quite a challenging task for our classifiers, where we were only able to classify with a 63% accuracy at a maximum between the development and testing data sets.

Consequently, we decided to introduce two more subreddits into our experiment: /r/OCPoetry and /r/haiku, where we would test all pairs of our four subreddits against each other in classification experiments. We instantly noticed that /r/OCPoetry featured, by far, the longest length posts, and /r/haiku featured the shortest. In the end, we found that any subreddit being classified with /r/OCPoetry was easily classified, with an accuracy of at least 96%. Similarly, any subreddit paired with /r/haiku classified well, with an accuracy of at least 86%. It did appear, however, that some of these cases involving /r/haiku seemed to exhibit some sort of learning over the training data iterations. Finally, the only real struggle our classifier seemed to have was for our original problem of classifying against /r/TwoSentenceHorror and /r/TwoSentenceSadness, where our final testing data was classified with 57% accuracy at a maximum. We believe that the variable responsible for our varying classification accuracy is the amount of difference between the lengths of each subreddit's posts.

Methods

For all data collection purposes, we used the `Praw` Reddit API to extract 500 posts (top of all time) from all four subreddits, all done in the file `data_collector.py.` To clean up the posts, we removed newlines and used a regex to get rid of non alphanumeric characters (except for spaces.) Finally, there are six different ways to pair the four subreddits together, so we created six folders with six files each. Each file contained training, development, and testing data. The total number of posts used is 500 + 500 for each folder, (500 from each subreddit,) and split up 50% / 25% / 25%, respectively. We also made sure that there were equal amounts of posts from each (of the two) subreddits in each split. The other three files in the folder contains which subreddits the posts came from, in the same order of occurrence as the posts. These will act as the target values, or labels, when using the `scikit-learn` classifier modules. Furthermore, we made sure to properly format the testing data in `data_collector.py` such that every pair of entries are from alternating subreddits. We did this to ensure that during actual training and testing, which occurs in our second and final python file, `classifiers.py`, we are able to grab 20 posts at a time (where each chunk of 20 contained an equal amount of posts from each subreddit) and test that current subset of the entire training set on the development and testing data sets.

Support Vector Classifier

Our first classifier is a support vector classifier. At a high level, support vectors are the feature vectors that lie closest to the line (a hyperplane) dividing the two datasets, i.e., posts from two distinct subreddits in our experiment. This optimal line might not exist, but the algorithm attempts to find a line of best fit throughout the training process. Before starting any training, our training data (in our case, posts separated by newlines) must be converted into feature vectors. Using `scikit-learn`‘s `CountVectorizer` object, we are able to convert each post into a feature vector of token counts for each word in the post. Given an initial data array of posts (strings), we can convert to feature vectors:

```
data       ⟵ [post1, post2, ..., postN]
vectorizer ⟵ CountVectorizer()
```

```
vectors     ⟵ vectorizer.transform(data)
```

Next, we slightly optimize the vectors such that the common words have less weight (to more accurately classify posts), by using `scikit-learn`'s `TfidfTransformer` object (meaning "term-frequency times inverse document-frequency). Using the previously computed token vectors, we obtain an optimized, transformed version of them:

```
transformer ⟵ TfidfTransformer()
vectors′     ⟵ transformer.transform(vectors)
```

Following this, we are now ready to train our data. We accomplish this by using `scikit-learn` to initialize a new support vector classifier (SVC). Note that the only non-default parameter we initialize the SVC with is the `gamma` parameter, which we set to the value `scale` (which is the default option in later builds of `scikit-learn`). The only other variable we need besides our newly produced `vectors′` are the target values, called `labels`, which are the correct corresponding subreddits to the posts from our training data arrays:

```
SVC ⟵ SVC(gamma='scale')
SVC.fit(vectors′ , labels)
```

Now, to test our SVC, we have to first transform our test set into feature vectors. We can do this the same way as before. This allows us to use the `score` function from `scikit-learn` with parameters to get a result between 0 and 1:

```
res      ⟵ SVC.score(test_vectors, test_labels)
accuracy ⟵ res × 100
```

In order to produce a learning curve to visualize the process, we made a for loop which cumulatively increased the number of training data in intervals of 20 data points and re-trained the classifier each time, where the results from each interval test was stored in a python dictionary for each of the six experiment scenarios. To produce the graphs, we used the `pyplot` interface from the `matplotlib` library.

Random Forest Classifier

Our second classifier is a random forest classifier. Random forest is one of two supported ensemble learning averaging algorithms. The idea of ensemble learning is to combine predictions from a set of hypotheses to make one prediction, using a majority vote. The idea is that if incorrect votes are unfavorable probabilistically, then for the majority of the time, correct votes are picked. All of the steps in creating and transforming the training, development, and testing data into feature vectors and labels using `scikit-learn`'s `CountVectorizer` and `TfidfTransformer` objects are exactly the same as the steps seen above. The only difference here is that we are now using a Random Forest Classifier object (RFC), initialized using `scikit-learn`. Note that the only non-default parameter we initialize the RFC with is the `n_estimators` parameter, which we set to the value 100 (which is the default option in later builds of `scikit-learn`). Thus, using the same method of data transformation to get our `vectors` and `labels` objects, we can initialize and train an RFC:

```
RFC ⟵ RandomForestClassifier(n_estimators=100)
RFC.fit(vectors′ , labels)
```

Lastly, to test our RFC with the `score` function using our transformed test data and labels:

```
res      ⟵ RFC.score(test_vectors, test_labels)
accuracy ⟵ res × 100
```

Results

Learning Curves

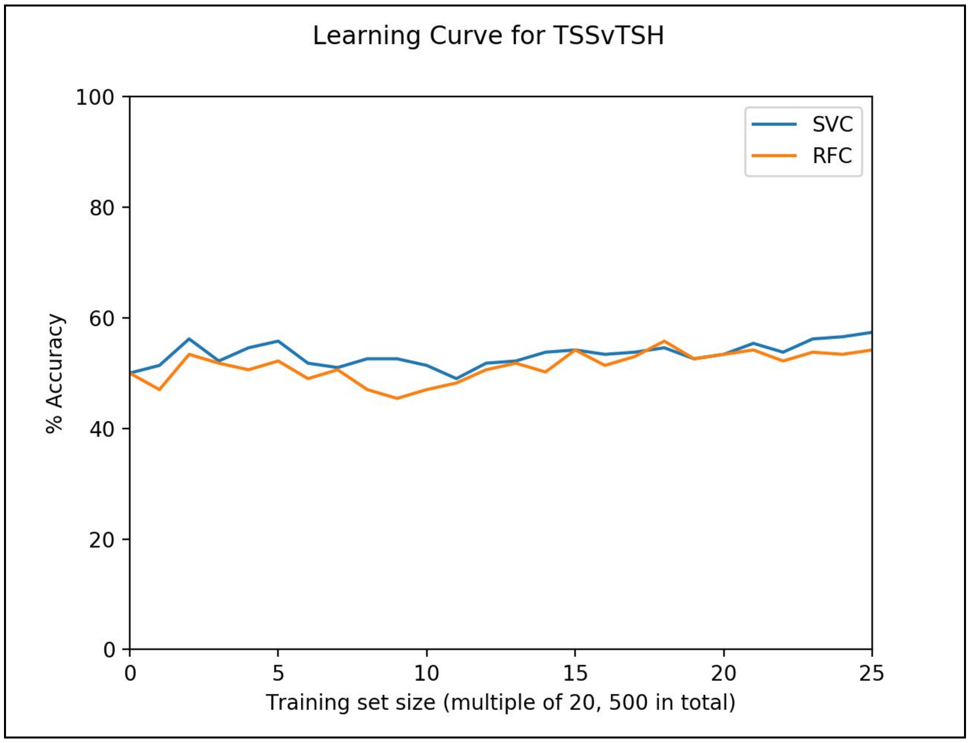

Figure 1: Classifying between /r/TwoSentenceSadness and /r/TwoSentenceHorror

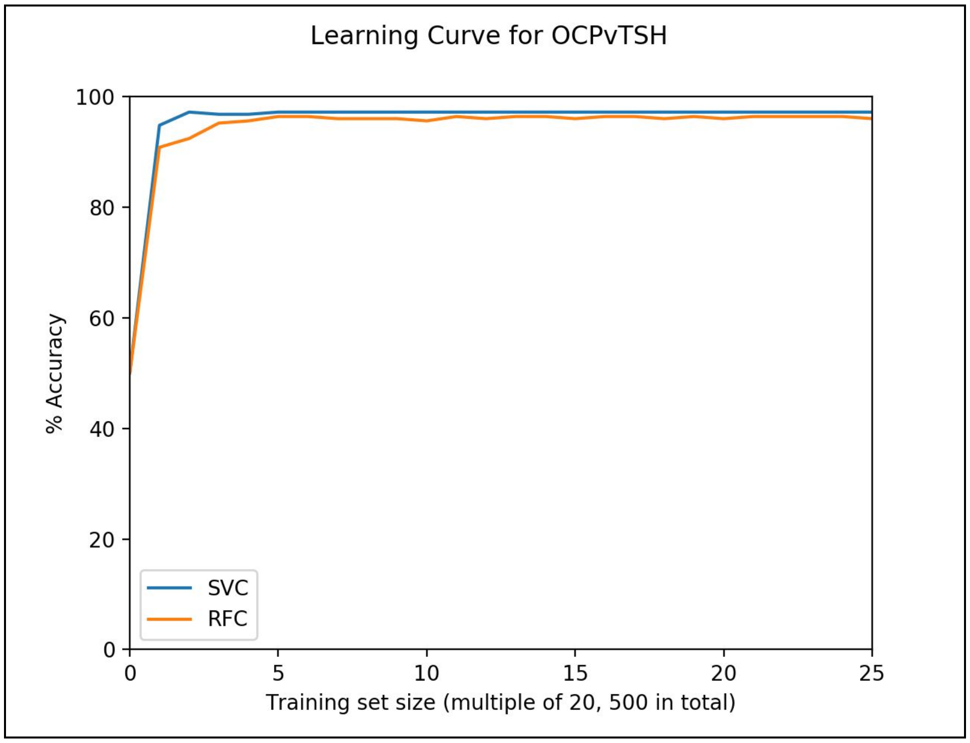

Figure 2: Classifying between /r/OCPoetry and /r/TwoSentenceHorror

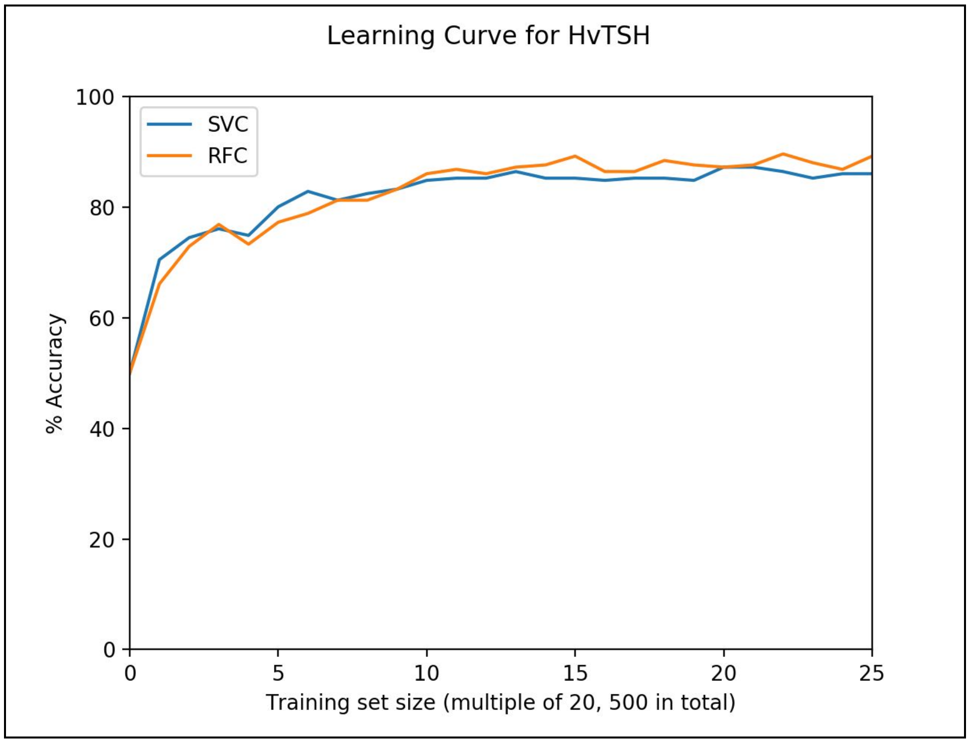

Figure 3: Classifying between /r/haiku and /r/TwoSentenceHorror

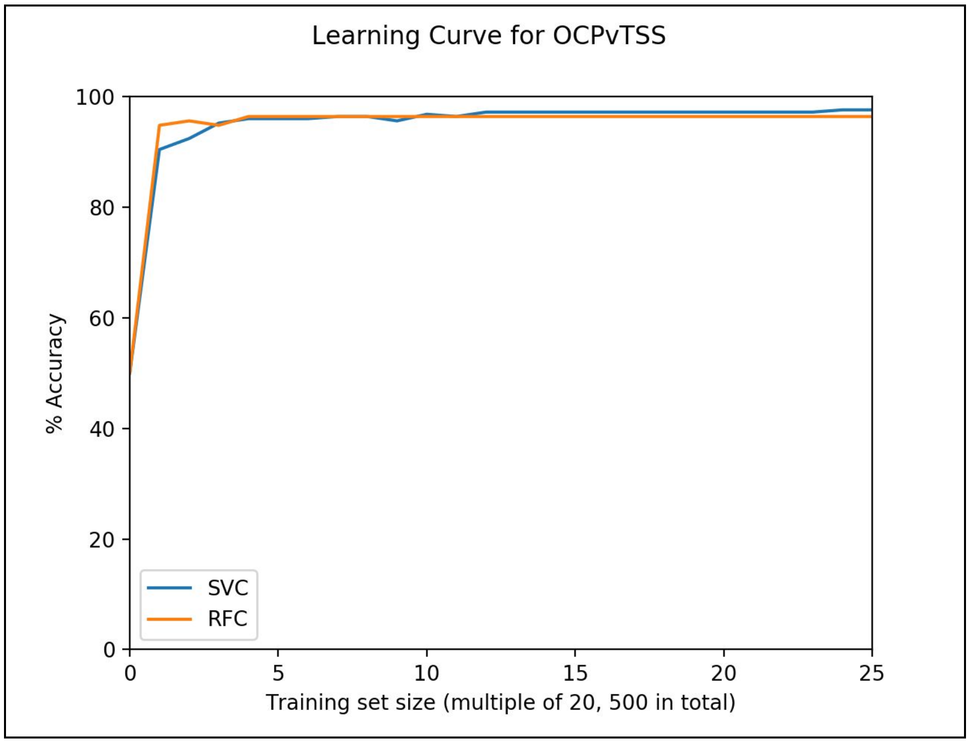

Figure 4: Classifying between /r/OCPoetry and /r/TwoSentenceSadness

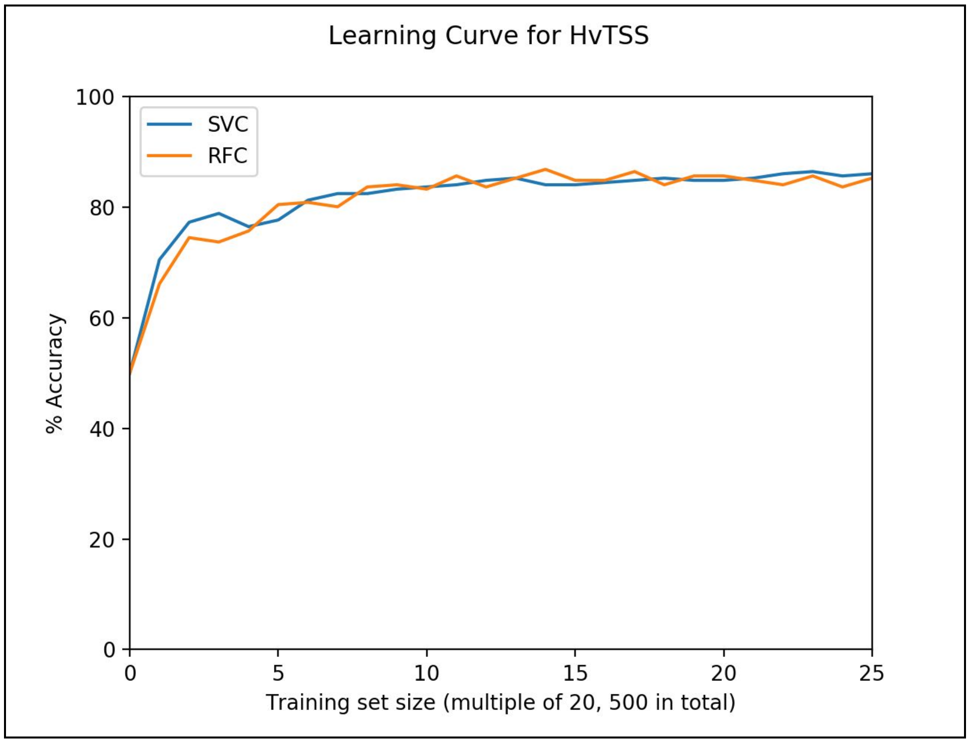

Figure 5: Classifying between /r/haiku and /r/TwoSentenceSadness

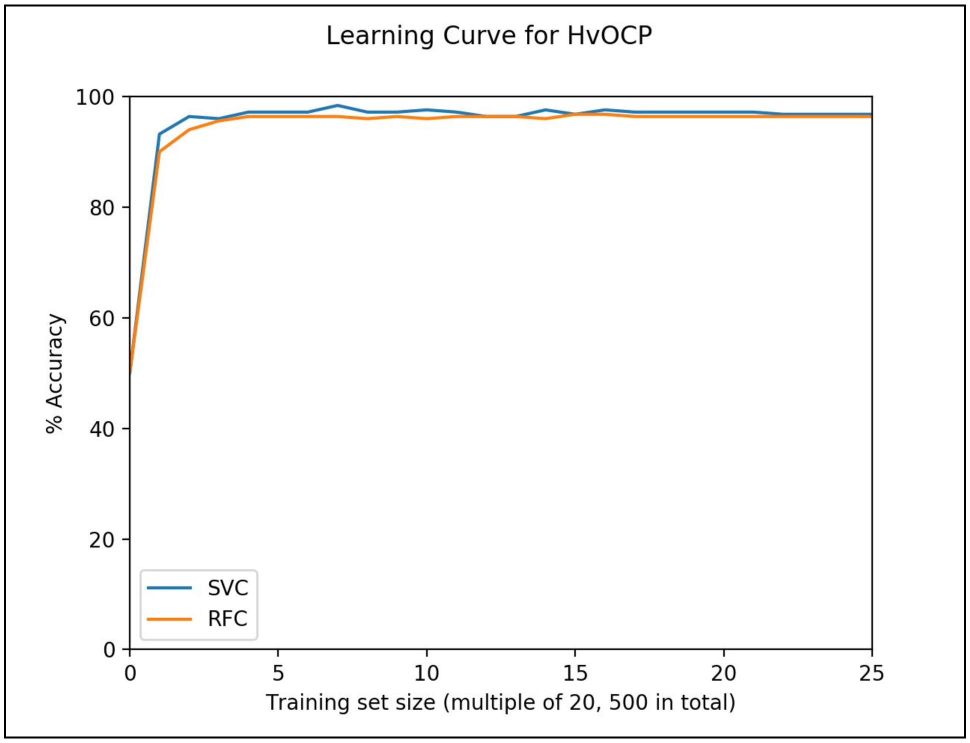

Figure 6: Classifying between /r/haiku and /r/OCPoetry

Our results were mostly expected, where we initially hypothesized that the bigger the difference between the lengths of the posts between the two subreddits, the easier the classification would be between them. We believe that this hypothesis is mostly confirmed, if not completely, by our collected data (at least in the context of testing these four subreddits). A strong example following this hypothesis is any classification involving /r/OCPoetry, where the classification accuracy was at least 96% for any of them. Again, we attribute this case of simple classification to the fact that most of the posts from /r/OCPoetry were *very* long compared to the other subreddits' posts, making it very easy for the classification algorithms to learn the difference between the posts. Additionally, we believe that little to no learning happened in the classification between /r/TwoSentenceSadness and /r/TwoSentenceHorror (figure 1), as the classification accuracy fluctuated between 50 and 60% throughout the whole testing process. We attribute this result to our classification algorithms not being able to differentiate between post semantics, i.e., sadness and horror, as it couldn't differentiate between two very similarly sized posts. Finally, we believe the highest case of learning appeared in the two classifications involving /r/haiku and /r/TwoSentenceHorror, and /r/haiku and /r/TwoSentenceSadness (figures 3 and 5). We attribute this case of actual learning to the fact that the posts from these classifications had a noticeable length difference, but not strikingly obvious one.

Conclusion

Using `Praw` to grab posts from four subreddits, we set up a training set of 500 posts, a development set of 250 posts, and a testing set of 250 posts. Next, the `scikit-learn` library was used to create feature vectors, which were subsequently used to train our classifiers. Due to the high-level nature of `scikit-learn`, training and scoring a support vector classifier and a random forest classifier required the same steps. Learning curves comparing every two sets of subreddits and the algorithms used were constructed using `matplotlib`. Visually, it can be seen that the two algorithms generally performed well and have very similar performance. It can also be hypothesized that the difference in the sizes of posts from different subreddits is easily learned.

We noticed that /r/TwoSentenceSadness and /r/TwoSentenceHorror used many similar words in their posts, but in different contexts. In order to improve classification between those two subreddits, we could assign emotions to words and use those emotions as weights instead of the number of occurrences. An indirect way to improve learning is to parse our data better. There was a lot of trouble parsing the data into a nice form because there are a lot of exceptions for whether or not certain characters should be removed. For example, quotes don't belong in a word, but apostrophes can sometimes be used as quotes as well as conjunctions. Also, different keyboards have different quotes which turn out to be different characters. Some redditors do not follow the formatting rules of the subreddit, so manual labor is also required for parsing the data further.