Analysis
Kaggle Username: Rohan Chandra | Final Kaggle Score: 0.7222 | Final Rank: 18/59

1. **Handling the data:** Since we were not given the labels for the test set, there was no way to evaluate the performance of our features. So the first step was to create a validation set by splitting the training data in an 80:20 ratio for cross evaluation. This ratio was not random, but a careful matching of the base score on kaggle with the base performance score on the validation set.

2. **Metrics:** The three metrics over which I engineered my model are:

   - **Type of Vectorizer:** `CountVectorizer` creates a feature matrix based on the count of the feature present in an example. The `TfidfVectorizer` introduces a different scheme for weighting the features.
   - **Built-in analyser:** The default analyser chosen in the vectorizer is `word`. Other suitable options are `char` and `char_wb`. An alternative is to create you own analyser which provides an added advantage of using a fixed vocabulary alongside ngrams and many other features, whereas the built-in analyser is limited in its flexibility. However, I removed the custom analyser after I realized it was outperformed by the built in analyser.
   - **Usage of additional features:** Such as IMDb.

3. **Best Model:** The best model that worked for me was training the example sentences concatenated with the trope and the `genre` attribute of the IMDb dataset and passing that through a `TfidfVectorizer`. Surprisingly, the built-in analyser outperformed the custom analyser. Additonally, the lexical features used were ngrams of the order $(1 - 8)$. Addionally, I normalized the feature matrix after it came out of `fit_transform`. One more interesting idea was to train an *additional* `vectorizer` for a fixed vocubulary taken from Jordans 2013 paper on spoiler detection. I tried many other IMDb attributes but other than `genre`, none seemed to have much of an effect.

4. **Intuition**

   - **IMDb:** The motivation for using IMDb features is that I reached a limit to how much I could increase the score with simply the sentence examples provided in the training dataset. Using the `genre` attribute bumped my score upto 69%. After further tuning the `vectorizer`, I finally reached a peak score of 72%. I also considered other IMDb features such as `runtime, first_aired, and number of episodes`. Thrillers, drama, and mystery shows are typically longer about 60 minutes as compared to comedy sitcoms that are of 30 minutes. This could have been an excellent feature since thrillers and mystery shows tend to be spoiler heavy. Also, the year when the show first came out is also slightly important. Shows that have been running for a while now are generally deeper into the plot story and contain more spoilers than shows that have started airing recently. However, these features did not work which I explain in my error analysis below.
   - **Trial Options:** After fixing the options that seemed to make an improvement, namely `analyser` and `ngram_range`, I began tweaking and tuning these parameters and recorded scores for each trial run. I discovered that I obtained the highest score by using `analyser = 'char_wb'` and `ngram_range = (1,8)`. `char_wb` allows for use of features like *he s* and *he k* which could be part of *he said/says* and *he kills/killed.* Thus using word boundaries is also useful.
   - **Using the tropes:** TV tropes are used widely to describe the tone of a scene. It turns out that using tropes along with the sentences improves accuracy than just using plain sentences.
   - **Using a vocubulary:** I defined a global list that contained certain unigrams and bi grams that are common in spoilers. I used a additional `vectorizer` for this purpose and and to tune it separately. Although it did not improve the score, this step makes sense intuitively.

5. **Error Analysis**

   - **Why no other IMDb attribute other than `genre` seemed to work**: While studying the data, I discovered that a majority of the data came from TV shows. This imbalance skewed certain attributes such as `runtime` which was either 30 or 60 minutes at best for T.V shows. The imbalance would suggest that if the classifier sees a new example with a large runtime that could be for a movie, it would treat it as an anomalous T.V show and might classify it incorrectly. So we might need to have a larger and more diverse set of data samples.
   - **Why Concatenation of features matched/slightly-outperformed Stacking of trained feature matrices:** This one was puzzling. All the intuition suggested that training different sets of features like the sentences and the IMDb attributes separately through 3 different `vectorizers` and then combining the overall normalized feature matrix would largely increase performance. But it seemed to have no effect. On the other hand, clubbing the features together and then passing them into a single `vectorizer` proved to be equally efficient if not a little better. My reasoning for this anomaly is that combining feature matrices led to a huge feature matrix which gives my machine a memory error. I use an ultra book which is inferior to regular laptops in terms of computation resources. Thus I was forced to limit the maximum number of features using the `max_features` option in `TfidfVectorizer` and thus could not utilize the full power of the stacked feature matrix, which explains why accuracy score didn't increase.
   - **One-Class Classification:** I decided to adopt a model similar to one-class SVM where I train only on positive examples to see what would happen. As expected, it failed miserably since we have to discriminate between two classes and if we supply only one class, there is not much to classify.