Feature Engineering Analysis

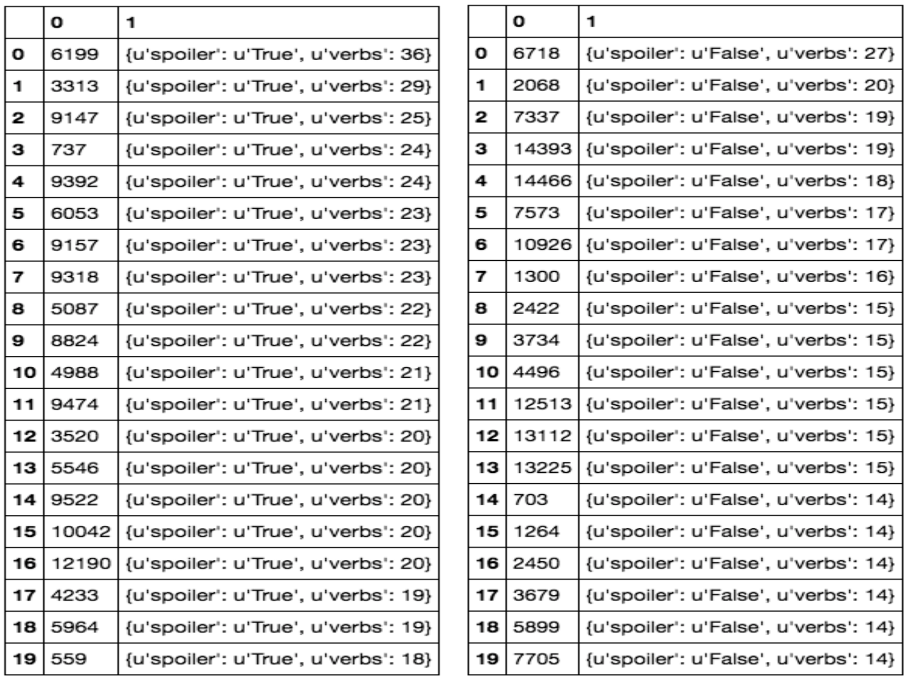
Paul Laliberte' | Kaggle: Paul_8 | CSCI-5622

1. Overview

The best overall accuracy that I achieved was a .70711, obtained with two features, and which was about a 6% increase on the default public leaderboard score. For cross validation, I used the sklearn k-fold with a $k = 5$. This was helpful in that an increase in my cross validation accuracy (of several percentage points) did result in a higher leaderboard accuracy, but there was some deviation between the two. I will give a brief overview of my feature selection process and why I believe certain features were better than others.

2. Feature Selection

The most obvious feature that came to mind was the length of a particular spoiler candidate. After running some iterations on the training set (separate from classify.py script) I came to the conclusion that a sentence consisting of approximately 38 - 42 words resulted in spoilers the majority of the time. I achieved a score very close to my best when combining this feature and a tf-idf vectorization (all default parameters). I soon arrived at a better method; counting up the number of verbs in a sentence (any type of verb). For this I implemented parts-of-speech tagging using the nltk library, and counted the number of verbs in any given sentence (Figure 1). I settled on labeling any sentence that had greater than 13 verbs as "1" (spoiler) and a "0" (not spoiler) otherwise, and then incorporated the new feature into my pipeline. Spoilers tended to have a lot of verbs within them (killed, killing, shot, ect.), and long sentences have a fair amount of verbs too. This way I could incorporate verbs and length into one feature essentially.

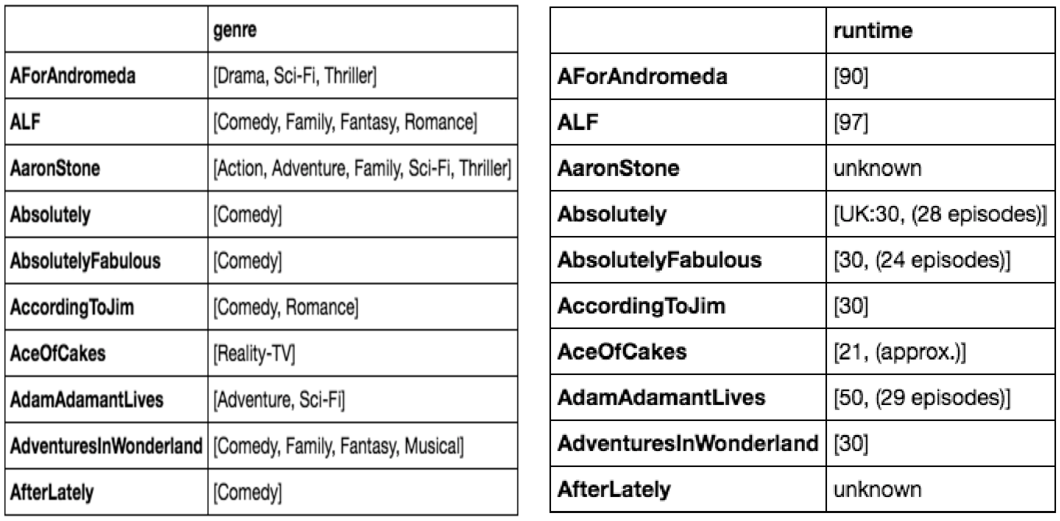


	0	1
0	6199	{u'spoiler': u'True', u'verbs': 36}
1	3313	{u'spoiler': u'True', u'verbs': 29}
2	9147	{u'spoiler': u'True', u'verbs': 25}
3	737	{u'spoiler': u'True', u'verbs': 24}
4	9392	{u'spoiler': u'True', u'verbs': 24}
5	6053	{u'spoiler': u'True', u'verbs': 23}
6	9157	{u'spoiler': u'True', u'verbs': 23}
7	9318	{u'spoiler': u'True', u'verbs': 23}
8	5087	{u'spoiler': u'True', u'verbs': 22}
9	8824	{u'spoiler': u'True', u'verbs': 22}
10	4988	{u'spoiler': u'True', u'verbs': 21}
11	9474	{u'spoiler': u'True', u'verbs': 21}
12	3520	{u'spoiler': u'True', u'verbs': 20}
13	5546	{u'spoiler': u'True', u'verbs': 20}
14	9522	{u'spoiler': u'True', u'verbs': 20}
15	10042	{u'spoiler': u'True', u'verbs': 20}
16	12190	{u'spoiler': u'True', u'verbs': 20}
17	4233	{u'spoiler': u'True', u'verbs': 19}
18	5964	{u'spoiler': u'True', u'verbs': 19}
19	559	{u'spoiler': u'True', u'verbs': 18}

	0	1
0	6718	{u'spoiler': u'False', u'verbs': 27}
1	2068	{u'spoiler': u'False', u'verbs': 20}
2	7337	{u'spoiler': u'False', u'verbs': 19}
3	14393	{u'spoiler': u'False', u'verbs': 19}
4	14466	{u'spoiler': u'False', u'verbs': 18}
5	7573	{u'spoiler': u'False', u'verbs': 17}
6	10926	{u'spoiler': u'False', u'verbs': 17}
7	1300	{u'spoiler': u'False', u'verbs': 16}
8	2422	{u'spoiler': u'False', u'verbs': 15}
9	3734	{u'spoiler': u'False', u'verbs': 15}
10	4496	{u'spoiler': u'False', u'verbs': 15}
11	12513	{u'spoiler': u'False', u'verbs': 15}
12	13112	{u'spoiler': u'False', u'verbs': 15}
13	13225	{u'spoiler': u'False', u'verbs': 15}
14	703	{u'spoiler': u'False', u'verbs': 14}
15	1264	{u'spoiler': u'False', u'verbs': 14}
16	2450	{u'spoiler': u'False', u'verbs': 14}
17	3679	{u'spoiler': u'False', u'verbs': 14}
18	5899	{u'spoiler': u'False', u'verbs': 14}
19	7705	{u'spoiler': u'False', u'verbs': 14}

Figure 1: verbs

The two features brought me to the .70711 mark. From this point on I tried a variety of additional features: genres (Figure 2), runtime (Figure 2), stemming (porter, snowball), word net lemmatizer, count vectorizer, database of key plot words, individual words that appeared more in spoilers, and parameter adjustments in the tf-idf vectorizer (stop words, punctuation removal, bigrams).



	genre
AForAndromeda	[Drama, Sci-Fi, Thriller]
ALF	[Comedy, Family, Fantasy, Romance]
AaronStone	[Action, Adventure, Family, Sci-Fi, Thriller]
Absolutely	[Comedy]
AbsolutelyFabulous	[Comedy]
AccordingToJim	[Comedy, Romance]
AceOfCakes	[Reality-TV]
AdamAdamantLives	[Adventure, Sci-Fi]
AdventuresInWonderland	[Comedy, Family, Fantasy, Musical]
AfterLately	[Comedy]

	runtime
AForAndromeda	[90]
ALF	[97]
AaronStone	unknown
Absolutely	[UK:30, (28 episodes)]
AbsolutelyFabulous	[30, (24 episodes)]
AccordingToJim	[30]
AceOfCakes	[21, (approx.)]
AdamAdamantLives	[50, (29 episodes)]
AdventuresInWonderland	[30]
AfterLately	unknown

Figure 2: genre and runtime (left element in array)

3. Feature Analysis

Intuitively, I estimated that around four uncorrelated features would produce the greatest accuracy. I incorporated the genres 'Mystery,' 'Action,' and 'Thriller,' which were the most indicative of spoilers. If a genre matched I labeled it as "1" and "0" if it did not (later I change it to "1" if any of them appeared and "0" otherwise). For runtime, I did not find any correlating factors within the training set. Logically, I assumed that longer episodes (add movies to this group) would have a higher probability of producing a spoiler, since there is more film content. I incorporated them without any obvious correlation on the basis that they provided some diversity to my features. There were quite a few television shows which did not have accurate runtime representations when queried from IMDbPY, the database I was using (labeled 'unknown' in Figure 2). This was the most likely reason they did not have a strong effect. The genre feature did not have any substantial effect either, but I left it in my pipeline because it did not harm my accuracy. Individual words that appeared more often in spoilers and key plot words did not have any effect.

I turned my attention to adjusting my tf-idf vectorizer using natural language processing methods. After some review of possible techniques, I settled on developing a tokenizer to use porter and snowball stemming to pass into my vectorizer. Since stemming reduced words to their 'stem' form this seemed like a viable option to make features stand out more. Unfortunately, this didn't seem to result in any additional accuracy, though snowball stemming appeared to be a little better than porter stemming (did not drop my accuracy as much). In my last attempts I incorporated bigrams of range [1,2], and developed a word net lemmatizer. Bigrams created a larger amount of features, which I concluded could possibly be my problem. The lemmatizer was a last-ditch effort, and I did not have high hopes for it because of its similarity to stemming.

Overall, I was surprised that the genres did not contribute more. I have two theories as to why my accuracy did not increase above .70711:

1. The tf-idf vectorizer had too much influence for any of the other features to have any real effect. However, I did not receive any favorable scores when I did not include it.
2. Outside of the tf-idf vectorizer and the verb count my features simply did not have any strong correlations to the the training set, regardless of my underlying intuition.