Predicting Genres by using Movie Plot Summaries

By Shallow Learning Team
Mina Vukovic
Prabesh Pokharel
Shae Ziegler

Introduction, Goal of the Project and Motivation

Background:

- Natural Language Processing: using ML to derive meaning from language

Goal:

- Predict movie genres given plot summaries.

Motivation:

- Use NLP for a daily problem of classifying movies based on their genre
- Allows to more accurately tag movies in order to better recommend movies to the users preference, or based on commonalities of plot summaries across genres

Speaker: Shae

Related Work

Gupta, Kunal. “Predicting Movie Genres Based on Plot Summaries.” Medium, 11 June 2019, https://medium.com/@kunalgupta4595/predicting-movie-genres-based-on-plot-summaries-bae646e70e04.

Reddy, Krushith. “Predicting Movie Tags Using Plot Summaries.” Medium, The Startup, 19 June 2019, https://medium.com/swlh/predicting-movie-tags-using-plot-summaries-5f0d7ddb5827.

The authors of these models used multilabel classification, making their problem more difficult. This allowed them to attach multiple genres to one movie summary.

Speaker: Shae

Related Work cont.

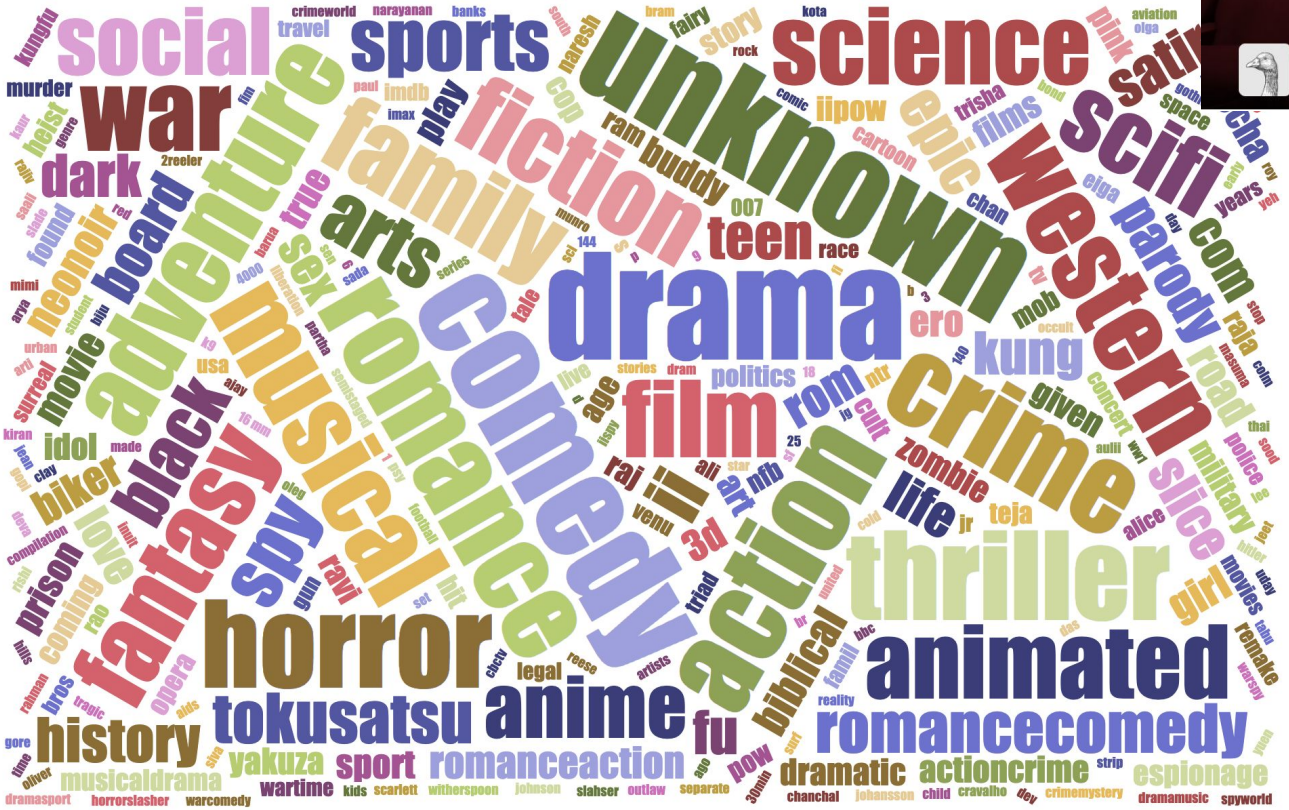
We found this research paper, “MPST: A Corpus of Movie Plot Synopses with Tags.” They set of 70 tags for movie plots and the multi-label associations. They investigate how these tags correlate with plots. They used multiple methods to see if it affected their results.

Their results had an F-1 score overall of 36.9 which they considered to be very low.

	F1	TR	TL
Baseline: Most Frequent	29.7	4.225	3
Baseline: Random	4.20	4.328	**71**
Unigram (U)	37.6	7.883	22.6
Bigram (B)	36.5	7.216	19.6
Trigram (T)	31.3	5.204	15.4
Char 3-gram (C3)	37.0	7.419	22
Char 4-gram (C4)	37.7	7.799	22.6
2 skip 2 gram (2S2)	34.2	6.289	19.4
2 skip 3 gram (2S3)	30.8	4.951	12.8
Bag of Concepts (BoC)	35.7	7.984	29
Concepts Scores (CS)	31.1	4.662	7.8
Word Embeddings	36.8	6.744	13.2
Semantic Frame	33.4	5.551	13.4
Agent Verbs	32.9	5.050	7.2
Patient Verbs	33.1	5.134	7.4
U+B+T	37.2	8.732	30
C3+C4	**37.8**	8.662	28.8
U+B+T+C3+C4	37.1	9.991	36.8
Al lexical	36.7	10.046	37.6
BoC + CS	35.7	8.165	29.4
All features	36.9	**10.364**	39.6

Speaker: Shae

Genres from Our Dataset



Dataset

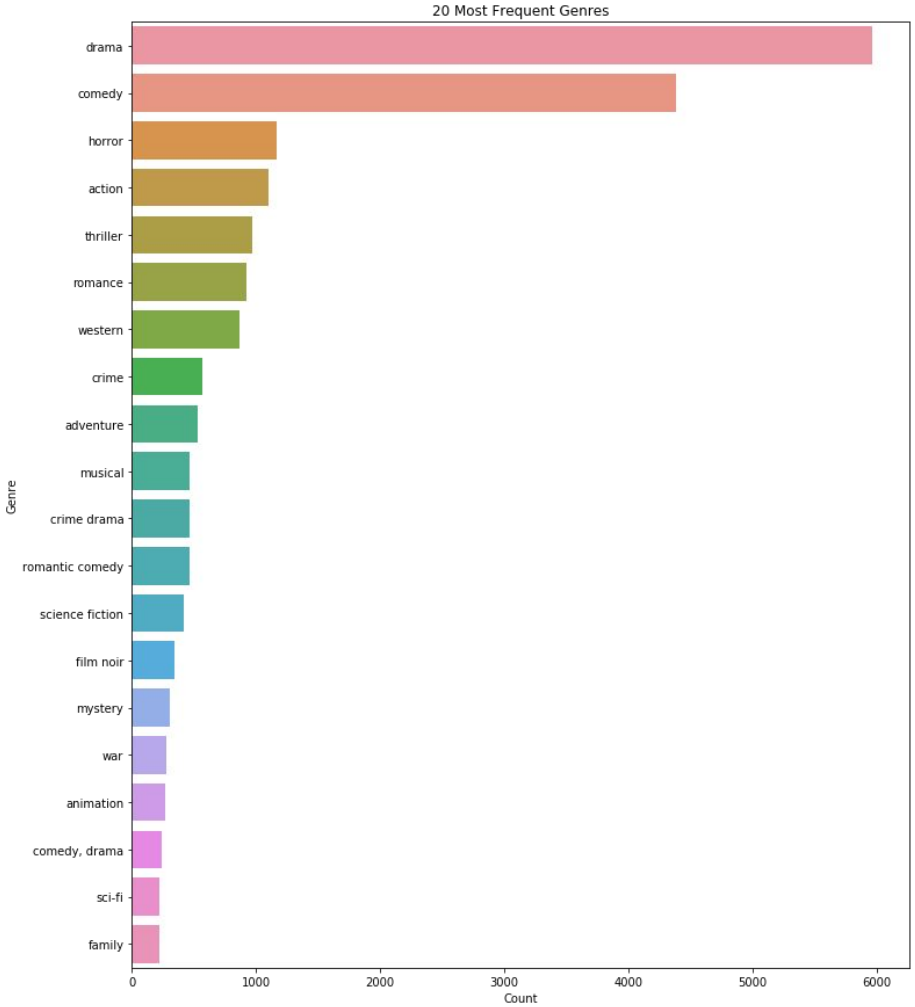
Wikipedia Movie Plots

Plot descriptions for ~35,000 movies

JustinR • updated a year ago (Version 1)

- Data set from 2265 different genres
- We focused on genres with a large amount of plot summaries

Speaker: Shae

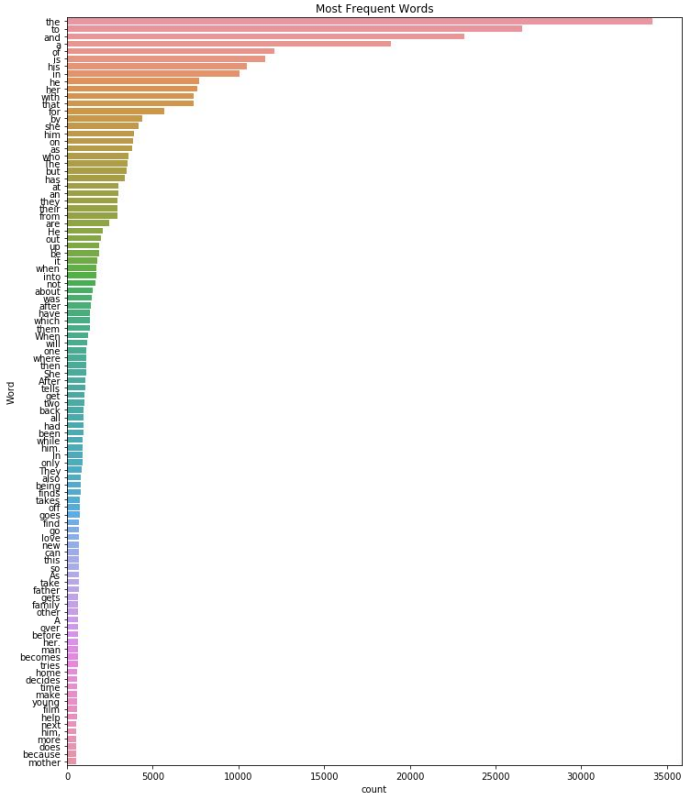


- To begin with we used genres Horror and Comedy
- Horror initially had 1167 plot summaries
- Comedy had 4379
- We restricted our training data to be 1000 random plot summaries of each category
- Our test data became the rest of the plot summaries in that genre

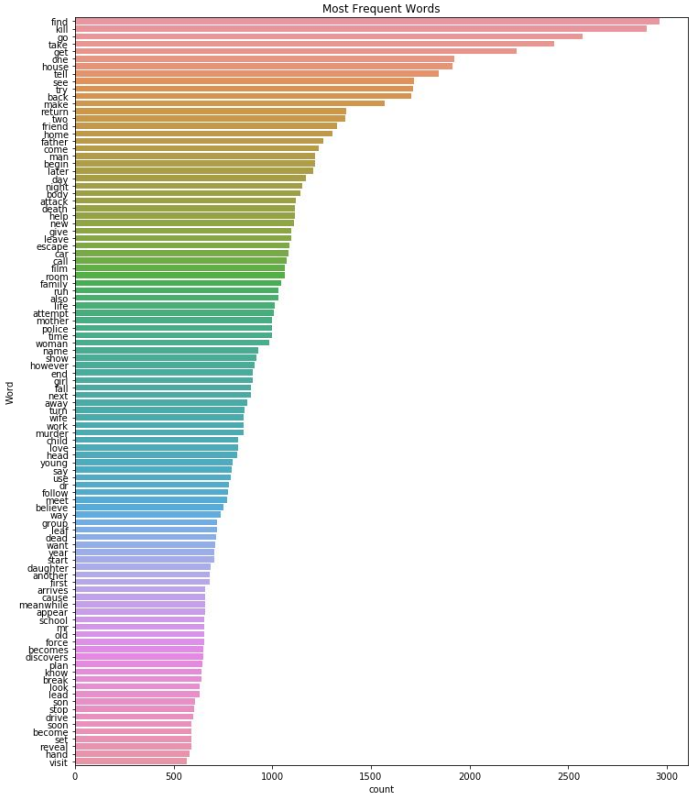
Speaker: Shae

Methods: Text Processing

Before Normalization



After Normalization



Speaker: Shae

Methods: Normalizing the text

We needed to go through the plot summaries and normalize all of the words it contained. These steps are as follows:

1. **Lower case** all the text to make it similar
2. **Punctuation** have no meaning so we deleted them
3. **Stop-words** such as “a”, “the”, “an” “in” have no meaning so we deleted them.
4. Expand **Contraction words**: I’m -> I am
5. **Accent Character:** we converted accented characters to regular characters.
6. **Lemmatization**: words like “played” and “play” are same so deleted “-ed” “-es” using lemmatization technique.

We used nltk library to normalized the texts in the plot summary.

Speaker: Shae

Methods: Cleaning the data

For our original two genres, Horror and Comedy, we created individual data frames based on what we wanted to use for testing and for training. For our first run we split the data into 80% for training and 20% for testing. Then we also did a 50% testing, 50% training and lastly a 60%training and 40% testing

```
#Function to create dataframes for each genre; using 1000 elements for training
def dataframe_creator(genre):
    genre = df_original.drop(df_original[df_original['Genre'] != genre].index, inplace = False)
    genre.drop(['Release Year','Origin/Ethnicity','Cast', 'Wiki Page','Director','Release Year'],axis=1,
               inplace = True)
    genre = genre.sample(frac=1)
    genre = genre.reset_index(drop=True)
    genre_train = genre[0:1000]
    genre_test = genre[1000:len(genre)]
    return (genre_train,genre_test)
```

Speaker: Shae

Methods: TF-IDF vectorization

Now our data is ready to feed into the ML model? **NO**

We now need to do Word embedding where human language in mapped to vectors to feed to the ML model.

We are using TF-IDF method for mapping words into vector.

Speaker: Mina

Methods: TF-IDF vectorization cont.

TF = frequency of word

IDF = uniqueness of word

TF-IDF = frequency * uniqueness. Represents the importance of a word relative to the document

Use this to find which words are most important in different genre plot summaries

$$w_{x,y} = tf_{x,y} \times \log\left(\frac{N}{df_x}\right)$$

TF-IDF

Term x within document y

$tf_{x,y}$ = frequency of x in y

df_x = number of documents containing x

N = total number of documents

Speaker: Mina

Methods: Running the ML models

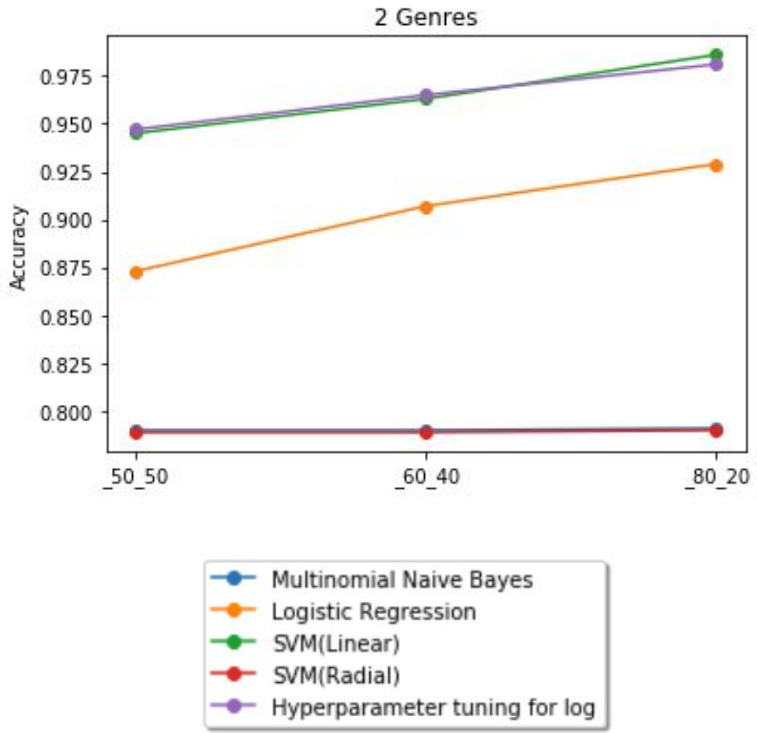
1. Multinomial Naive Bayes Classifier
2. Logistic Regression Classifier
3. SVM Classifier - Linear Kernel
4. SVM Classifier - Radial Basis Function Kernel
5. Logistic regression with hyper parameter tuning

To test the performance of our classifier we used accuracy as a measure which is a percentage of texts that were predicted with the correct tag.

Speaker: Mina

Results: Accuracies with 2 genres (horror, comedy):

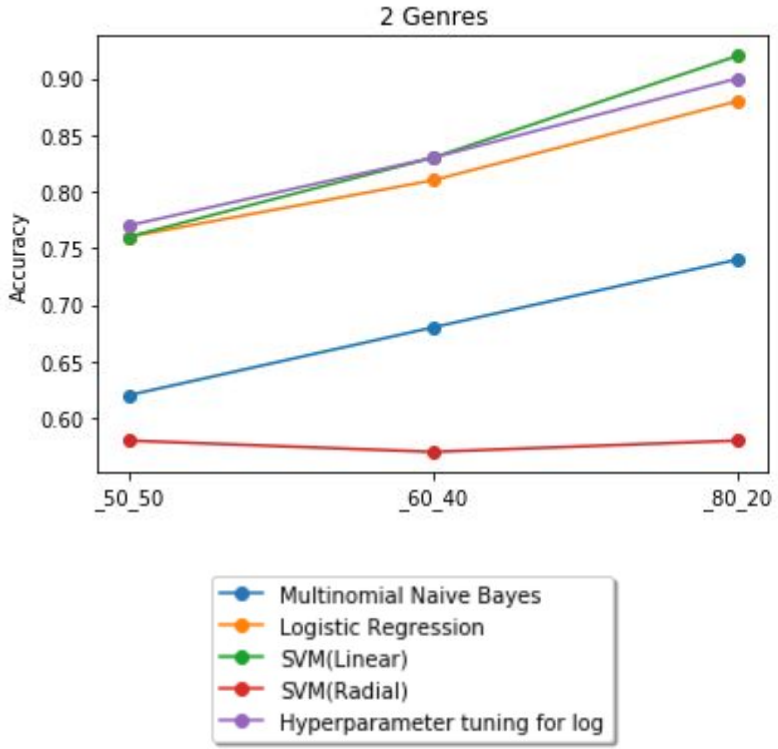
Classifier	50:50	60:40	80:20
Multinomial Naive Bayes	0.790	0.790	0.791
Logistic Regression	0.873	0.907	0.929
SVM (Linear)	0.945	0.963	0.986
SVM (Radial Basis Function)	0.789	0.789	0.790
Hyperparameter tuning for log	0.947	0.965	0.981



Speaker: Mina

Results: Accuracies with 2 genres (drama, comedy):

Classifier	50:50	60:40	80:20
Multinomial Naive Bayes	0.620	0.683	0.739
Logistic Regression	0.755	0.814	0.876
SVM (Linear)	0.765	0.833	0.918
SVM (Radial Basis Function)	0.577	0.577	0.577
Hyperparameter tuning for log	0.765	0.830	0.896

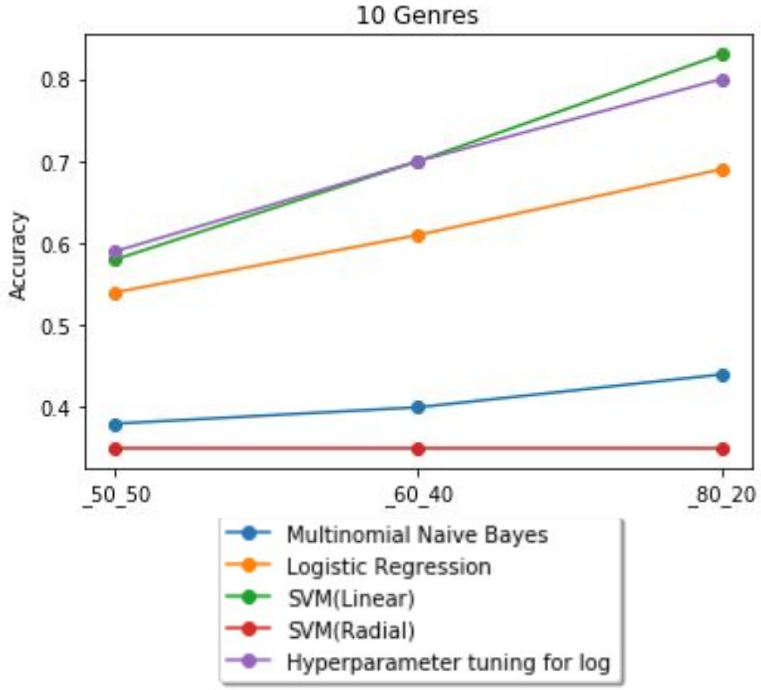


Speaker: Mina

Results: Accuracies with top 10 genres

Horror, comedy, drama, action, thriller, romance, western, crime

Classifier	50:50	60:40	80:20
Multinomial Naive Bayes	0.378	0.403	0.438
Logistic Regression	0.537	0.613	0.692
SVM (Linear)	0.582	0.701	0.832
SVM (Radial Basis Function)	0.352	0.352	0.352
Hyperparameter tuning for log	0.586	0.703	0.804



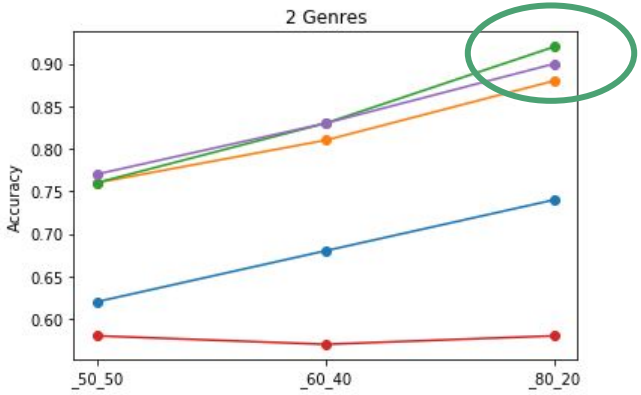
Speaker: Mina

Discussion: Classification on Horror vs Comedy and Comedy and Drama

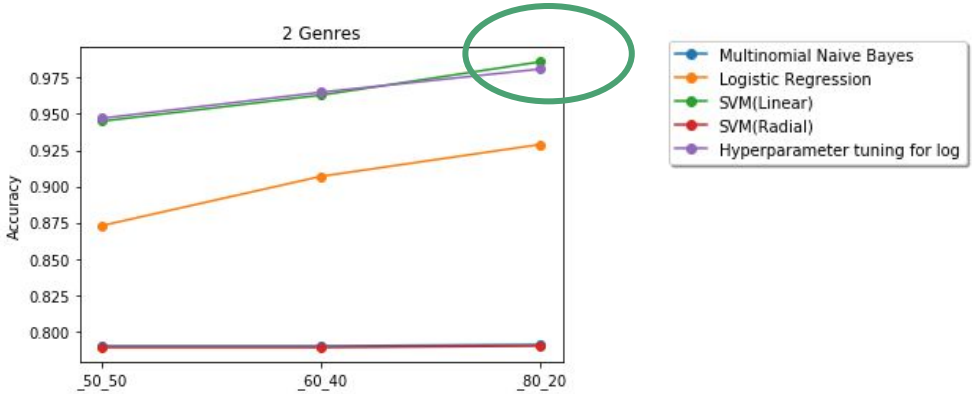
Words like **blood, murder, kill** that appears a lot in the plot summary of Horror Movie unlike in Comedy Movie. So the classification task for Horror and Comedy Movie was very successful. (98% accuracy)

Words that appear in Comedy Movie and Drama Movie are very similar. So, the classification task was not successful as in comparison to the classification of Horror and Comedy Movie. (91% accuracy)

Drama and Comedy Classification



Horror and Comedy Classification

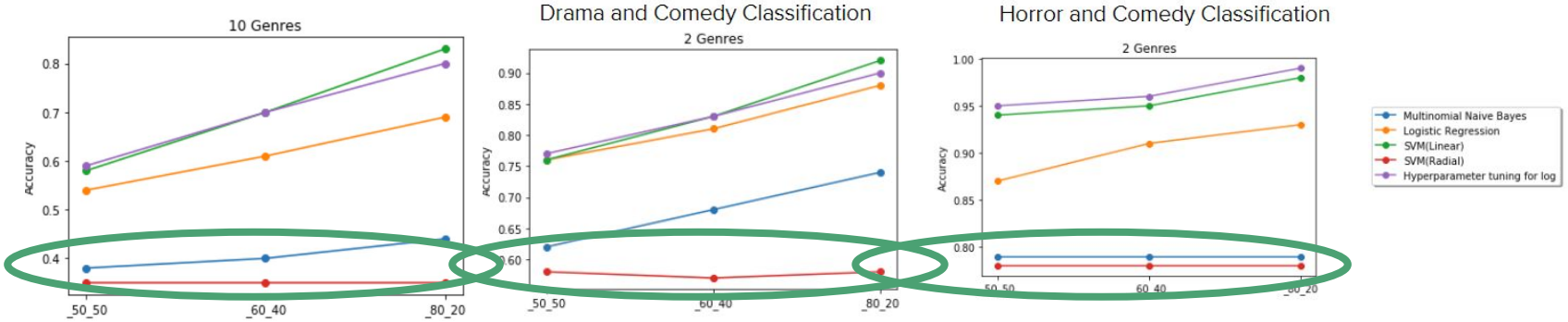


Speaker: Prabesh

Discussion:

Something we are still struggling to explain is why did the Radial Basis Function Kernel for SVM performed so poorly.

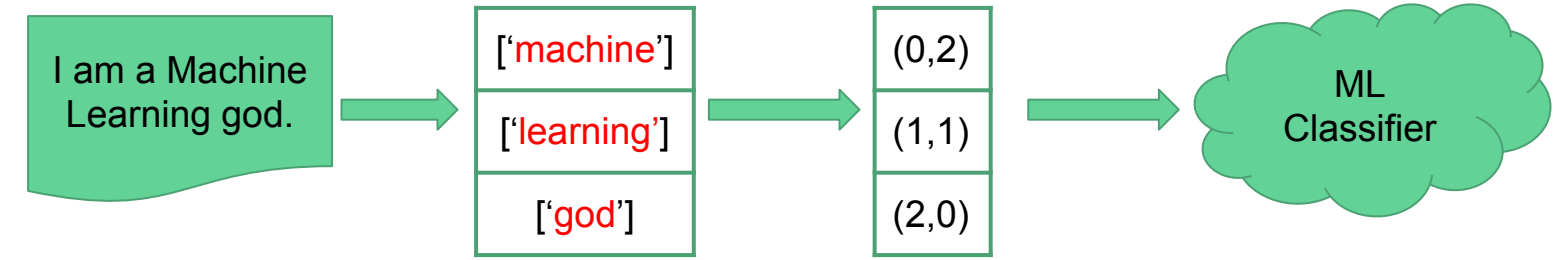
- Hypothesis that's not test yet: Sample size is not same for all 10 genres. For example: 80% of drama plot summaries > 80% of crime plot summaries.
- Hyperparameter tuning



Speaker: Prabesh

Discussion: What we learnt

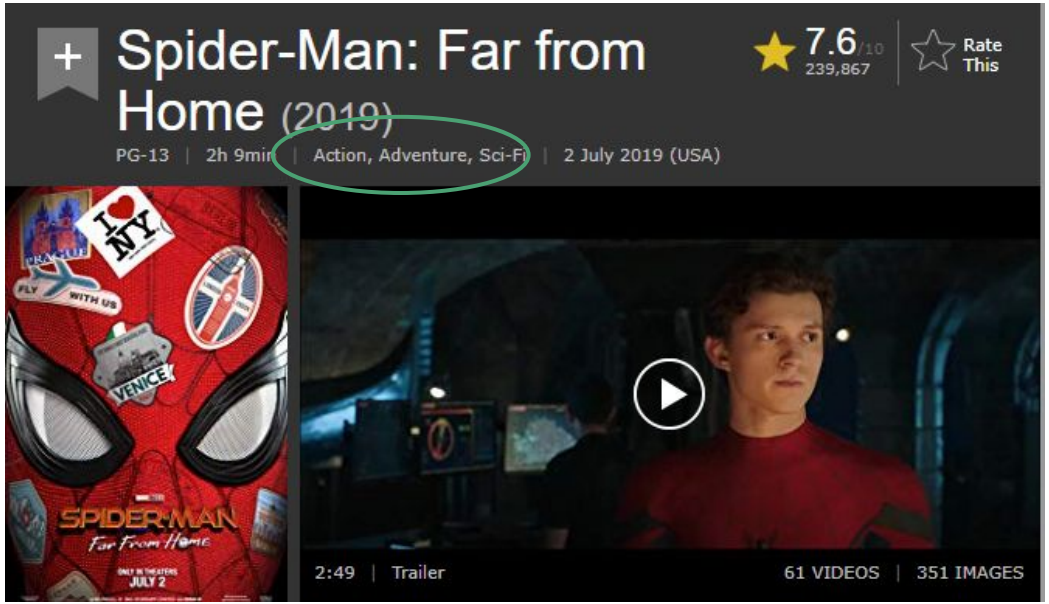
- how to prepare the text data by extracting features to make it ready to use by Machine Learning Classifiers.



Speaker: Prabesh

Future Work - Where do we go from here

1. Implementing multi-label classifier



Img src:https://www.imdb.com/title/tt6320628/?ref_=nv_sr_srsg_0

Speaker: Prabesh

Future Work - Where do we go from here

2. Use ensemble method + Hyper-parameter tuning for Naive Bayes, SVM(linear/ radial basis function)that we used for genre classification

For example for svm:

```
('The best classifier is: ', SVC(C=100.0, cache_size=200, class_weight=None, coef0=0.0, degree=3,
  gamma=0.10000000000000001, kernel='rbf', probability=False, scale_C=True,
  shrinking=True, tol=0.001))
```

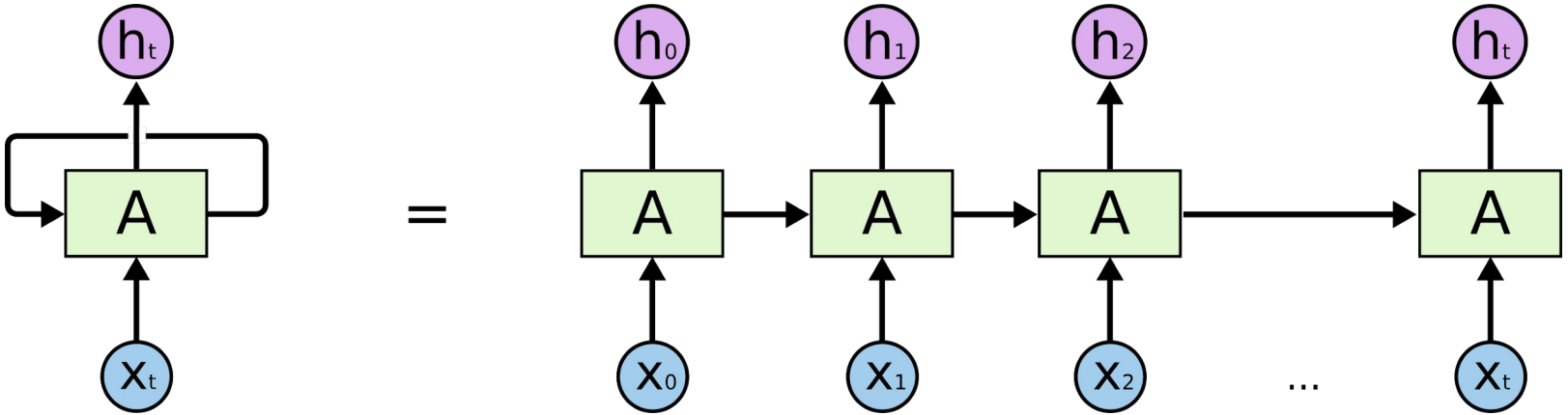
Img src: https://ogrisel.github.io/scikit-learn.org/sklearn-tutorial/auto_examples/svm/plot_svm_parameters_selection.html

Speaker: Prabesh

Future Work - Where do we go from here

3. GO DEEEEEEP

Implement deep neural network models RNN- LSTM



Img src: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Speaker: Prabesh

Future Work - Where do we go from here

4. Take advantage of pre-trained model like Word2Vec, Google Bert

```
8]: # Failed attempt
    model = gensim.models.KeyedVectors.load_word2vec_format('./GoogleNews-vectors-negative300.bin', binary=True)
    test_x_w2v = model[test_x]
    test_x_w2v
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
```

Speaker: Prabesh

References

- https://www.nltk.org/
- https://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html
- https://www.analyticsvidhya.com/blog/2018/04/a-comprehensive-guide-to-understand-and-implement-text-classification-in-python/
- https://www.kaggle.com/jrobischon/wikipedia-movie-plots
- http://filotechnologia.blogspot.com/2014/01/a-simple-java-class-for-tfidf-scoring.html