Movie Genre Popularity Prediction

Joseph Keller, Joseph McDonald, Kevin Scott, Roger Sarvate
{Joe.Keller,Joe.McDonald,Kevin.Scott, Roger.Sarvate}@dsu.edu

Figure 1 – Which movie genres are trending in popularity at a given time?

1. Task

Our project attempts to answer the question: In a given month and year, which movie genre (comedy, action, drama, etc.) will be the most popular or most successful in terms of box office returns? We focused on domestic (U.S.) films and aimed to build a classification model based on past movies' genre and earnings information in conjunction with a number of indicators that are representative of public "mood" (e.g. economic and/or political indices and variables) to see if we could pinpoint which movie genre is most likely to be the most popular at a given time. The most challenging and interesting aspect of this project was attributing changes in various genres' popularity among consumers to numerous potential factors, including seasonality, the state of the economy, current events, and more, and attempting to create an appropriate model to quantitatively represent those links in a stable, predictable manner.

2. Approach

Our approach to this project consisted of four primary steps:

1. Data Preprocessing
2. Data Exploratory Analysis and Modifications
3. Preparing and Adding Predictor Variables
4. Model Testing and Selection

Data Preprocessing:

During the preprocessing phase, we transformed our raw Kaggle movies dataset into the appropriate format so that it could be fed into the model for training to make predictions about the most popular genre in a month (based on all movies released in theatres that month). All the steps undertaken in this portion of the project required a good amount of coding and usage of **pandas** and **numpy** libraries and associated methods for data transformation (see next section – 3. Dataset and Metric – for more specific information). We also decided on and created the derived metric which we felt was the best proxy for genre success or popularity – **ROI (Return on Investment**), expressed as a percentage and defined as:

$$\boldsymbol{ROI\ for\ Genre\ G} = \frac{\sum Revenue\ for\ movies\ in\ Genre\ G}{\sum Budget\ for\ movies\ in\ Genre\ G} - 1$$

A Note about the "ROI" Success Metric:

Note in the above equation that we used a dollar weighted average for calculating the average ROI in each month. For example, our modified dataset included an average comedy genre ROI in Jan 1980 (and all subsequent months) calculated as the total revenues of all comedy movies released in that month divided by the total budgets of all comedy movies released in that month, minus 1. The reason we did not calculate average genre ROI as a straight average of all individual movie ROIs is that movie budgets and revenues tended to vary drastically (as revealed in the data exploratory analysis section), and furthermore, the "typical" ROI for a movie was strongly dependent on its budget. Low budget films often gather enough revenue to recoup costs many times over, leading to ROI percentages that are commonly in the thousands. For this reason, it is not surprising that the bulk of the movies in our dataset are on the lower-budget end (less risk to produce and a higher likelihood of recouping costs), as shown in the below graph:

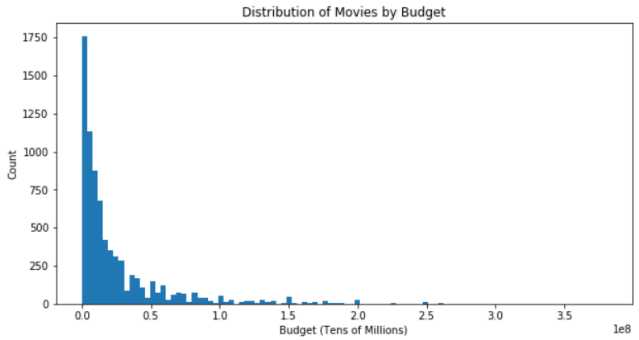


Figure 2: Most movies represented in our dataset are on the lower-budget end (0 to 5 million USD).

We therefore used the dollar-weighted average in calculating ROI to lessen the "noise" that would likely occur in our success metric due to the fluctuations in the mix of large vs. small budget films from month-to-month.

Data Exploratory Analysis and Modifications:

For the data exploratory analysis we ran histograms and examined distributions of different variables in the dataset to get an idea of which variables are meaningful to use as input features (or as potential "popularity" variables) and which needed modifications. The coding portion of this step involved using **matplotlib** to build some graphs/plots, as well as basic statistics methods within **pandas** data frames (i.e. "group by" method to summarize certain classes by monthly aggregates).

The first two phases of this project (preprocessing and exploratory analysis) were *iterative*. That is, as a result of information uncovered during the exploratory step, we modified the dataset in certain ways so that it would be better suited for training a model to make predictions (see 3. "Dataset and Metric – Preprocessing" section for more detail on these modifications).

Preparing and Adding Predictor Variables

The predictor variable preparation portion required researching potential input features that we thought would be useful in predicting movie genre popularity movement and importing time series data sets for these indicators (for example, "Consumer Confidence" from FRED website). Also, we created "derived" indicator variables that represented conditions that we believed would influence the dependent variable – for example, a binary (0/1 for "No"/"Yes") that indicates whether or not a given month is within a year of a presidential or midterm election. Thus, coding was required for this step to import external datasets as indicators, to derive these variables from scratch, and to transform all indicators into the correct format so they could be joined to the primary movie genres dataset as input variables.

Model Testing and Selection:

Choosing a model to make predictions for our dataset involved running through the full series of basic, standard machine learning classifier algorithms (Linear, Logistic Regression, KNN, Decision Trees, Neural Networks, SMVs) to determine which model type and what hyperparameters gave us the best prediction accuracy on our test set. We determined this using extensive cross-validation or by testing many different combinations of different hyperparameters for each model type. During this step, we used pre-existing Python libraries for each model type (from **sklearn**), but we built the code required for the setup of the data (e.g. **train_test_split** method), and for running through different combinations of hyperparameters to optimize performance (using such methods as **GridSearchCV** and **cross_val_score**). After this, we printed out the classification accuracy (**score** method) on both the training set and the test set. For most of the models, we also included a sub-section where we did an assessment of "by-class" classification accuracy to discern each model's *specificity* vs. *sensitivity* in predicting a specific top genre by month correctly (for more information, see the "Predicting Simple Genre" section of "3. Dataset and Metric").

The following is more information about our approach for each model type. Classification accuracy scores and related information for each model will follow in the "4. Results" section.

Linear SGD Classifier:

We used the **linear_model** class from **sklearn** to implement the linear classifier. Specifically, **linear_model**.**SGDClassifier** was used to implement a stochastic gradient descent-based version of the linear classifier (in other words, updating the model parameters based on an iterative process of gradient descent using one training example at a time). Another important class implemented here was **RFE** (recursive feature elimination), which was used to perform backwards selection and eliminate features (out of a total possible 21 input variables) with less predictive value.

Neural Networks (MLP Classifier):

The class we used for the neural network classifier was **MLPClassifier** in **sklearn.neural_network**. A wide range of parameters were tested for this model (including different hidden layer sizes and structures, solvers, and activation functions) to determine which yielded the highest test set accuracy. The "solver" was either set to lbfgs (Limited-memory Broyden-Fletcher-Goldfarb-Shanno, an optimization algorithm within the family of quasi-Newton methods) or adam (a modified SGD algorithm that uses the concept of "momentum" to implement adaptive learning rates). The activation functions we tested included logistic, rectifier ("relu"), hyperbolic tangent, or the identity function. The same tests were then done for the Simple Genre classification.

Following this procedure is an additional section titled "Accuracy by Class for Highest Scoring Non-Trivial Simple Genre Classifier (NNET)". This is where we took the highest scoring MLPClassifier model and computed the prediction accuracy by each of the three individual target classes in the Simple Genre dataset ("Horror", "RFC", "Other").

K-Nearest Neighbors (KNNs):

For implementing the KNN classifier model, we used **KneighborsClassifier** from **sklearn.neighbors**. The hyperparameter tuning was then done using **cross_val_score** method and graphing the misclassification rates for each possible value of k, the number of nearest neighbors (from 1 to 100). The minimum misclassification error-producing value of k was selected, and the procedure was repeated for several variations of the training data (see next section, "3. Dataset and Metric – Other Versions of the Training Dataset" for more details).

Following this was the additional section where we tested by-class classification accuracies for the Simple Genre dataset for the highest performing KNN model.

Decision Trees:

For implementing the Decision Tree classifier model, we used **DecisionTreeClassifer** from **sklearn.tree**. The **GridSearchCV** method was then used to test a range of combinations of parameters for the tree, including "criteria" ('gini' or 'entropy', as method for calculating tree impurity), max_depth (from 5 to 20, for limiting how many nodes, and therefore how complex the tree could become), and min_samples_leaf (from 1 to 50, for determining the minimum number of training samples that must remain at each "leaf" of the tree). The GridSearchCV method tests every combination using 5-fold CV on the training dataset to determine the "best", or highest scoring parameters. The same test was then done for the Simple Genre classification.

Following this was the additional section where we tested by-class classification accuracies for the Simple Genre dataset for the highest performing DT model.

Logistic Regression:

For implementing the logistic regression classifier model, we used **LogisticRegression** from **sklearn.linear_model**. Following this, a wide variety of models were tested, including different regularization parameters (C=0.01, 0.1, 1, 10, 100, etc.). Also, the models were tested on different variations of the primary training dataset (see "3. Dataset and Metric – Other Versions of the Training Dataset" for more details) as well as the Simple Genre classification.

Following this was the additional section where we tested by-class classification accuracies for the Simple Genre dataset for the highest performing LR model.

Support Vector Machines (SVMs):

For implementing the SVM classifier model, we used **SVC** from **sklearn.svm**. Similar to the Decision Trees, the hyperparameters were optimized using the **GridSearchCV** method to test a range of combinations of parameters for the model, including the kernel type (linear, polynomial, radial basis function or "RBF", and sigmoid), different values for the regularization parameter C (0.001, 0.01, 0.1, 1, 10, 100, 1000), and the degree of the polynomial kernel (2,3,4,5,6). The same test was done for the Simple Genre classification.

Following this was the additional section where we tested by-class classification accuracies for the Simple Genre dataset for the highest performing SVM model.

3. Dataset and Metric

The primary dataset we are using is The Movies Dataset from Kaggle, specifically movies_metadata.csv (https://www.kaggle.com/rounakbanik/the-movies-dataset#movies_metadata.csv). This is a listing of around 45,000 movies created since the early 1900's and is based on information taken from the GroupLens website as well as IMDB (Internet Movie Database).

However, the number of relevant examples for our model ended up being much smaller due to the way we processed the data set into a monthly time series (see next section for more detail). The reason we are using the metadata file is that it contains important information relevant to our prediction model, such as genre, release date, budget, and revenue.

Detailed Information about Preprocessing Steps

For purposes of our project, we needed to transform the raw data from a listing of individual movies (one movie per record) to a time series dataset that summarized the "popularity" of each genre for each month and year (e.g. "Jan 2007") in our chosen time period. More specifically, this required the following preprocessing steps:

Removing N/A Records and Fixing Data Types

Not all of the columns in the original "Movies" dataset were applicable or useful for our purposes. After importing the raw data CSV into a dataframe, columns with null values were filtered out. Also, metrics that would be important for building our dataset were converted to their appropriate data types (release date converted to "date" format, revenue/budget converted to "int", etc). Also, the raw data included movies that were never released and foreign (non-U.S. based) movies; these were filtered out.

Creating One-Hot Encoded Vectors for Movie Genres

There were 20 distinct genres represented in the movies dataset. Since "genre" is the ultimate classification target variable for our model, we needed to represent each movie genre in the appropriate numerical format. Thus, we created 20 binary variables, one for each genre. If a particular movie fell in genres X and Y (for example, action and comedy), X and Y variables would be equal to 1, and all other genre variables would be 0.

Substitutions for Missing "Revenue" or "Budget" Data

Since ROI is our "success" metric, the "Revenue" and "Budget" fields were very important for our purposes. Unfortunately, as many as a third of the movie records in the raw dataset were missing "Revenue", "Budget", or both. Rather than discard all records missing either of these fields, which would have led to a significant loss of data, we took a more proactive data mining approach and instead attempted to salvage some of these data for use in our model by following these steps:

- Calculate the average of the "Revenue" field for each of the 20 genres for all records that are *not* missing this data. Feed a separate Excel file into the Python notebook containing these averages.
- Do the same for the Budget field.
- For those records missing *either* Budget or Revenue, substitute with the average value for that genre, calculated in the above steps.
- For those records missing *both* fields, discard the record entirely.

Adding in ROI Percent ("Popularity") Metric

As described previously, we chose ROI (Revenue/Budget – 1) as our "success" metric for judging released movies' popularity. This was done at the individual record (movie) level, but as mentioned in the "Approach" section, we eventually opted to take dollar-weighted averages to calculate aggregate genre ROIs when it was evident that ROI varied wildly by movie budget level (low vs. high).

Summarizing Dataset as Time Series (and filtering out pre-1980 data)

The most substantial transformation of the raw movies data for our modeling purposes was to aggregate the movies by month and year (e.g. January 1980, February 1980, and so on) and include the average ROI for each genre for each month, so we obtained a *genre popularity time series* (named **ts_genre_pop** in the notebook) of a little under 500 month data points (extending from 1980 to 2017).

The reason that 1980 was selected as the lower cutoff year is that we were concerned about the sparsity of data (based on number of movies) prior to this point in time. The nature of our classification dataset required there to be enough representation in each of the movie genres in order to accurately determine which genre was trending in popularity in each month. For example, in earlier years (prior to 1980), there were several months with no movies of a given genre that year in the data, so it wouldn't have been meaningful to judge that genre's success relative to other genres.

Converting to Classification Dataset

This step involved identifying the top genre in each month from the genre ROI values for each of the 20 genres. This "top genre" target variable is ultimately what we used for training the model, since our goal was

classification, not predicting actual ROI values (regression).

Combining Low Frequency Genres into "Other" Category

20 target variables is a large number to work with for a predictive model. Furthermore, many of those genres (e.g. "Western", "Documentary", etc.) had relatively few movie counts overall in the dataset. Therefore, we decided to retain the 9 most common movie genres based on yearly average frequency since 1980 ("Drama", "Comedy", "Thriller", "Action", "Romance", "Crime", "Adventure", "Horror", and "Science Fiction"), and aggregate all others into an "Other" category. The result was that we were left with 10 genre categories with greater representation and therefore more (hopefully) statistical credibility as far as patterns and ROI-related behavior.

Adjusting for ROI Variation by Budget Layer

Discussed above and in the "Approach" section.

Joining External Datasets and Predictor Variables

The training data that we fed into our models was not solely based on the Kaggle movies dataset, but was a mixture of derived variables and external economic and political variables joined to the aggregated time series version of the movies dataset.

This is a listing of all the predictor variables we used for training our models (21 in total):

- *Pres_election*: Binary variable (0/1) – does the given month in question fall in a year of a presidential election (e.g. 1980, 1984, 1988, etc.)?
- *Midterm_election*: Binary variable (0/1), same as above, except for years that are midterm elections for members for both houses of Congress.
- *Post_9/11_Attack*: A binary dummy variable (0 if before Sep-2001, 1 if after) that represents a presumable shift in the relative popularity of movie types with the American public before vs. after the 9/11 terrorist attacks.
- *RGDP_Chg_V1, V2, V3, V4:* These variables represented the economic indicator "Real GDP growth" represented in 4 different ways. This was necessitated by the fact that the "Real GDP growth" dataset from FRED was quarterly, not monthly. For **V1,** we used a "forward" view, meaning that if Q1 of 1980 was 3.0, for example, we extended this value to Jan, Feb, and Mar of 1980. For **V2**, we used " backward" view, meaning that if Q1 of 1980 was 3.0, we extended this value to Jan 1980, as well as Nov and Dec of 1979. For **V3**, we used a "midpoint" view, meaning that if Q1 of 1980 was 3.0, we extended this value to Dec 1979, Jan 1980, and Feb 1980. For **V4**, we determined each of the monthly values between quarter points using linear interpolation.
- *Cconf:* Monthly Consumer Confidence dataset from FRED.
- *Unemp:* Monthly unemployment rate dataset from FERD.
- *Month_jan* through *Month_feb*: Binary variables, 12 in total, indicating which month of the year it is.

Once all 21 of these predictor variables were merged with our original monthly time series dataset, our dataset was ready to be used for training the models.

Predicting "Simple" Genre – Alternative Mode of Classification:

Naturally, predicting one of 10 different possible target variable (genre) values using 21 variables and under 500 data points total for training, cross-validation, and testing raised concerns about the sufficiency of our data for properly training the models, and about the ability of certain classifiers (e.g. KNN) to function properly with such high-dimensional data.

For these reasons, we decided to implement an alternative mode of classification to the regular 10-genre predictions: the **"Simple Genre" model**, which consisted of only three condensed target variable categories – Horror, RFC (Romance/Family/Comedy), and Other (for all other genres).This way, we hoped the models would be more responsive and better at classifying the top genre than with 10 possible labels. The results and comparison to the 10-genre classification are discussed in the next section.

Other Versions of the Training Dataset:

The linear SGD classifier used recursive feature elimination to whittle down the number of predictor variables to those most relevant. To test other combinations of input variables, we also created different variations of the training dataset, each with a different combination. For example, while the extension

"*_sim"* denoted the Simple Genre dataset, "*_msim*" was the same dataset only with months as predictor variables, "*_nomo*" was the version *without* monthly predictor variables, and "*_nmsim*" was the Simple Genre dataset without monthly variables. This gave us a wider range of combinations to test, with the aim of getting a better response and classification accuracy from the models.

4. Results

Please see Appendix A for a graphical visualization comparing all model results.

10-Genre Classification:

The table below shows the training set accuracy, test set accuracy, and hyper-parameters for the model of each type which yielded the highest test set accuracy for the 10-genre classification dataset.

Model/Algorithm	Hyper-parameters	Training Score	Test Score
Linear Classifier (LinearSGD)	**Features used:** 16	18.6%	**16.8%**
Neural Networks (MLPClassifier)	**Solver** = 'adam', **Activation** = 'tanh', **hidden_layer_size** = [10, 100]	25.0%	**27.0%**
K-Nearest Neighbors	**K** = 81	26.0%	**25.0%**
Decision Trees	**Criterion** = 'entropy', **max_depth** = 5, min_samples_leaf = 41	27.5%	**28.3%**
Logistic Regression	**C** = 1.0 (default)	24.9%	**26.5%**
Support Vector Machines	**C** = 100, **degree** = 6, **kernel** = 'poly'	32.2%	**23.9%**

Simple Genre Classification:

Model/Algorithm	Hyper-parameters	Training Score	Test Score
Linear Classifier (LinearSGD)	**Features used:** 5	46.4%	**39.8%**
Neural Networks (MLPClassifier)	**Solver** = 'lbfgs', **Activation** = 'tanh', **hidden_layer_size** = [10, 100]	57.0%	**42.0%**
K-Nearest Neighbors	**K** = 33	51.0%	**42.0%**
Decision Trees	**Criterion** = 'gini', **max_depth** = 5, min_samples_leaf = 45	49.7%	**45.1%**
Logistic Regression	**C** = 100 (default)	47.3%	**41.6%**
Support Vector Machines	**C** = 100, **degree** = 6, **kernel** = 'poly'	47.3%	**39.8%**

Finally, for most of the model types, a "by-class" classification score was also measured across the three different target variable classes in the Simple Genre dataset for the highest-scoring model:

Model/Algorithm	"Horror" Accuracy	"RFC" Accuracy	"Other" Accuracy
NNET	33%	16%	69%
KNN	13%	16%	82%
Decision Trees	50%	93%	8%
Logistic Regression	3%	24%	82%
SVMs	3%	0%	98%

Discussion of Results by Model Type:

(Note: for simplicity the following discussion, the 10-genre dataset is referred to as "**TG**" and the simple-genre dataset is referred to as "**SG**" hereafter)

Linear Classifier (SGD):

This was our lowest-performing model on both the TG and SG data. This is not suprising, because linear classifiers are limited in their ability to represent relationships between input variables and the dependent variable, since the only possible hypothesis function type is a straight line.

NNETs (MLPClassifier):

Out of all the parametric learning algorithms we tested (i.e. those that "learn" a defined hypothesis function

H(x)), we might expect neural networks to yield the highest classification score because of their adaptability. This turned out to be true, although only by a small margin (**27.0%** score on TG, and **42.0%** on SG) second only to the non-parametric Decision Tree). The best MLPClassifier leveraged the hyperbolic tangent (tanh) function to better capture the clearly non-linear relationship between inputs and target variables.

K-Nearest Neighbors:

KNN is the classifier we predicted would perform the poorest on our TG data because KNN generally is not designed for high-dimensional data (we have 21 input features) and small datasets (under 500 data points). After optimization of the hyperparameters, our KNN classifier scored **25.0%** on the test set (with K = 81 neighbors), which was the lowest of all the parametric models except for SVM (23.9%). We wouldn't expect a KNN classifier to generalize well to data of this nature, particularly with such a huge number of nearest neighbors. At the same time, however, we also predicted that KNN would benefit the most by switching from 10 target variables to 3 in the SG data because of the sharp reduction in input feature dimensions. This may explain why, on the SG, KNN performed the second-best out of all models (though tied with NNETs on the test score) at **42.0%**,

Decision Trees:

The optimal Decision Tree (with a max depth restricted to 5 nodes, and each leaf restricted to no less than 41 samples) **was the best-performing classification model we tested, at 28.3% test score on TG and 45.1% on SG**.

We attribute this to the fact that DTs are non-parametric, i.e. they are not restricted to a specific hypothesis function that is fit to the data, and therefore DTs are inherently more flexible and accurate as classifiers which "segment" the data using a series of logical conditions. At the same time, we might be concerned about overfitting when using decision trees – however, the restriction on tree depth and minimum samples per leaf prevented overfitting, as shown by the slightly higher performance on the test set compared to the training set.

Logistic Regression:

In contrast to linear models, classification using logistic regression is better suited to data that includes binary variables (as in our dataset) in addition to continuous variables. This may be the main reason our LR models performed the second-best out of all the parametric methods on both TG and SG, at **26.5%** and **41.6%** respectively.

Support Vector Machines:

Unlike neural networks, which are designed to fit a wide variety of data, SVMs are best at finding maximum margins of separation between different classes of data using either linear decision boundaries, or pre-defined kernels which map the data to a higher dimension to be linearly separated. Unfortunately, our data did not seem to lend itself to neat separation in any dimension. As a result, our optimized SVM models, though leveraging a sixth-degree polynomial kernel and a radial kernel (RBF), performed the poorest out of all the models except for linear classifiers on both the TG and SG test sets, at **23.9%** and **39.8%** respectively.

Discussion of the Simple Genre "By-Class" Classification Scores:

One dismaying observation we made with many of our models was that the "optimized" model was often nothing more than a classifier with a "constant", or flat output that matched the most-common target variable value in the training dataset. For example, several of the classifiers simply returned a "Horror" or "Other" genre output for every single test observation. This is not useful for our purposes because it gives us no information about the relationship between independent and dependent variables. In order to help assess whether a given model was making meaningful predictions with a reasonable level of *sensitivity* (i.e. responsiveness to input variables), we looked at how the model performed on each individual class for the Simple Genre classification model (where model accuracy for class X = % of test observations of that class that were classified correctly).

The results of this "by-class" test (shown in the third table of the "Results" section) only confirm that there is indeed an imbalance in the accuracy levels between each of the three classes. For example, the SVM scored **98%** on "Other" category, but this was at the expense of accuracy in the other two classes. Essentially, the model "figured out" that it could maximize its accuracy simply by predicting the most common target label for almost every observation (a "naïve" classifier of sorts). The same was true of other models, although **NNETs** and **Decision Trees** performed better in this regard, with a

greater balance between scores (and therefore degree of discernment between classes on the part of the model).

Conclusions and What We Learned:

There were at least two factors that made the nature of our prediction task difficult and elusive before we even started training our models: 1) We were taking our best shot at choosing a set of variables that, in our judgment, captured public sentiment in order to tease out a correlation between movie success and environmental factors that in reality may not even exist, or may be largely drowned out by noise and idiosyncratic factors. It could be that our choice of variables (economic and sociopolitical indicators) missed the mark in capturing "public mood", or perhaps the correlation itself of these variables with the genre of movies earning the most at the box office (our "success" metric) was very weak to begin with. 2) The question we were trying to answer necessitated formatting our data in a way that left us with a relatively small number of training examples for our models. We weren't trying to model box office success for *individual movies*, but instead, the relative performance of *monthly aggregates* of movies by genre, which limited us to just under 500 points (after filtering out pre-1980 data). Also, in retrospect, the monthly grouping may have left us with too little data in each monthly "bucket" to yield a stable ROI metric for training the models on ten different target classes, as evidenced by the volatility in the monthly average ROIs over time. But any larger level of grouping (e.g. half-year, year, etc.) would have left us with too few data points to be of any practical use.

Aside from the previously discussed details about the relative performance of various model types (and the valuable knowledge that a non-parametric model like Decision Trees might be best for classification in a case like this), we learned this simple fact: ***predictive models based on machine learning algorithms are not magic*.** Though these models can discern relationships between variables that we often cannot see, in cases where the relationship is weak or non-existent, or if we haven't properly trained the models with enough data or correctly optimized parameters (we do believe we addressed these issues to extent possible in this project, given the limitations of our data), the output of the models will yield little new information, as seen in the cases of "flat" output predictions of one class or almost exclusively one class.

A hypothetical question we asked ourselves while working on this project was: how applicable would such a modeling and prediction task be in a real business setting? For example, would it help a movie producer to know that presidential approval rating plays a role in which movies earn the most at the box office (if in fact we had detected such a clear relationship, which we didn't)? On one hand, this is interesting from a sociological perspective. However, making predictions in the future based on this data would require one to know how presidential approval rating will change in the next few years. This could be highly inaccurate and/or impractical for someone attempting to use the information for important business decisions. On the other hand, such knowledge could lead to further studies that might allow one to better predict consumer behavior in actual business cases.

Note: We discovered late in our review of the notebook prior to submission that we had mistakenly left out the variables related to presidential approval ratings (*Approving, Disapproving, Unsure_NoData*). Please see Appendix B for an update of "best" models found in the Results section based on including these three variables. While we did not have time to reflect in detail on the change to these results, Appendix B shows the direction of the impact created by including these variables in our final list of predictors. Hyperparameters for the "optimal" model in each category (i.e. that which produced the highest accuracy on the test set) also shifted.

While the updated results seen in Appendix B do not change the core of our conclusions written above, we did see slight changes (mostly improvements) in the classification scores for each model. More notable was the shift in by-class classification scores, particularly the NNETs. With the inclusion of presidential approval rating variables, we see that the accuracies for the three target classes are now more balanced for NNETs, which means the models are having some success in incorporating this new information to cut out noise and make better predictions.

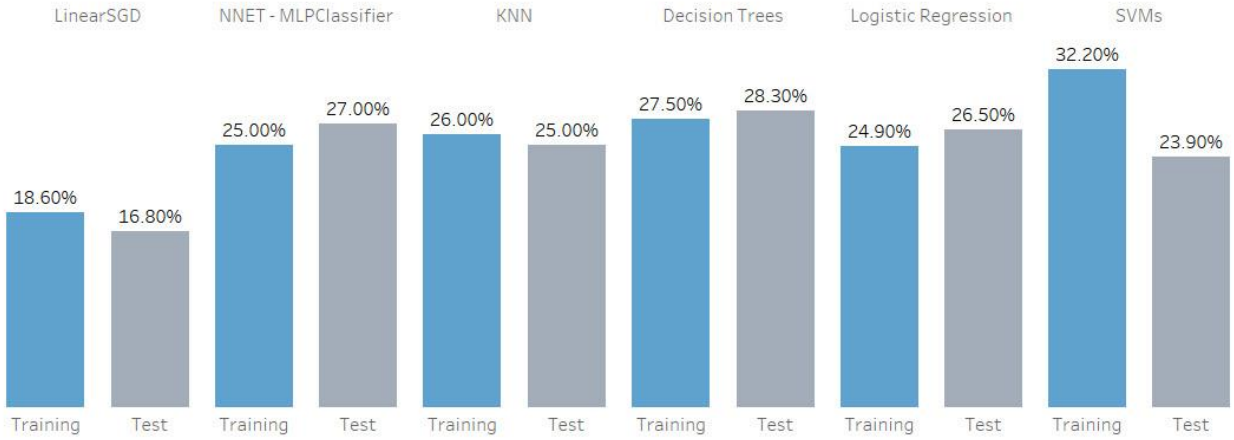
Tasks of Each Team Member

In addition to the below described roles, each team member contributed important ideas and suggestions and raised concerns which impacted the direction of the project and the final product.

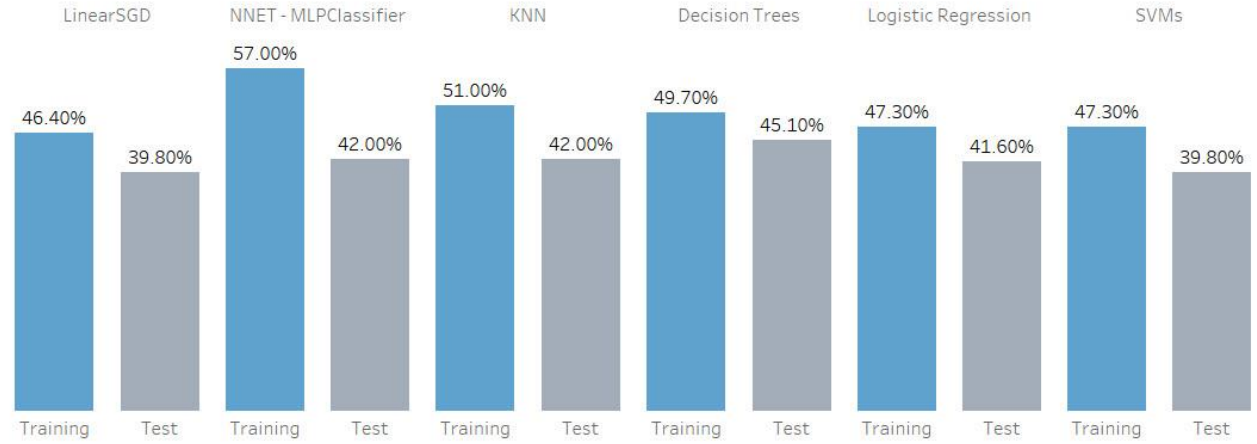
Task	**Related File names**	**Teammate**
Researched/derived sociopolitical predictor variables and integrated into training dataset	*election years.csv, pres_approval.xlsx, main ipynb file*	Joe McDonald
Researched/derived economic predictor variables and integrated into training dataset	*realgdp.xlsx, unemp.xlsx, consumer_conf.xlsx, main ipynb file*	Kevin Scott
Substitutions for missing revenue/budget data with genre Averages	*averages.xlsx, main ipynb file*	Joe Keller
Data Pre-processing and Time Series conversion	*movies.csv, main ipynb file*	Roger Sarvate
Exploratory analysis and primary dataset modifications	*main ipynb file*	Roger Sarvate
Dataset modifications and variants (nmsim, nomo, msim, etc.) for optimizing models	*main ipynb file*	Joe McDonald
Models: Linear classifier	*main ipynb file*	Roger Sarvate
Models: Logistic regression	*main ipynb file*	Kevin Scott
Models: Neural Networks	*main ipynb file*	Joe Keller
Models: K-Nearest Neighbors	*main ipynb file*	Joe McDonald
Models: Decision Trees	*main ipynb file*	Roger Sarvate
Models: Support Vector Machines	*main ipynb file*	Roger Sarvate

Appendix A.
Model Results Comparison
(Highest Classification Accuracy by Model)

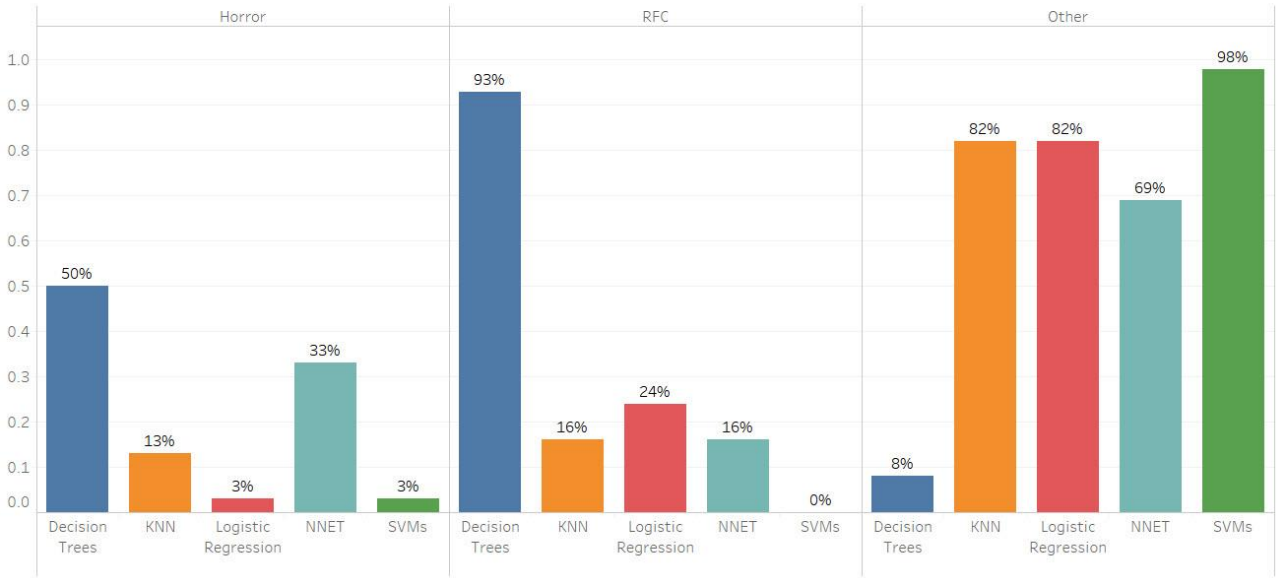
10-Genre Classification



Simple Genre Classification



Simple "By-Class" Score



Appendix B.
Model Results Updated With Inclusion of Presidential Approval Rating Variables

Model/Algorithm	Hyper-parameters	Training Score	Test Score
Linear Classifier (LinearSGD)	**Features used:** 12	17.8%	**18.6%**
Neural Networks (MLPClassifier)	**Solver** = 'adam', **Activation** = 'tanh', **hidden_layer_size** = [10, 100]	25.0%	**27.0%**
K-Nearest Neighbors	**K** = 85	25.0%	**26.0%**
Decision Trees	**Criterion** = 'entropy', **max_depth** = 5, min_samples_leaf = 42	26.9%	**24.8%**
Logistic Regression	**C** = 1.0 (default)	25.1%	**26.5%**
Support Vector Machines	**C** = 0.001, **degree** = 2, **kernel** = 'linear'	24.9%	**26.5%**

Simple Genre Classification:

Model/Algorithm	Hyper-parameters	Training Score	Test Score
Linear Classifier (LinearSGD)	**Features used:** 5	46.4%	**39.8%**
Neural Networks (MLPClassifier)	**Solver** = 'lbfgs', **hidden_layer_size= [100]**	54.0%	**41.0%**
K-Nearest Neighbors	**K** = 49	49.0%	**41.0%**
Decision Trees	**Criterion** = 'gini', **max_depth** = 5, min_samples_leaf = 43	49.7%	**45.1%**
Logistic Regression	**C** = 100 (default)	47.9%	**42.5%**
Support Vector Machines	**C** = 100, **degree** = 6, **kernel** = 'poly'	46.4%	**39.8%**

By-Class Classification Score:

Model/Algorithm	"Horror" Accuracy	"RFC" Accuracy	"Other" Accuracy
NNET	33%	26%	60%
KNN	3%	32%	80%
Decision Trees	43%	24%	63%
Logistic Regression	10%	24%	80%
SVMs	0%	0%	100%