Genre Classification - Report

CS109B Project - Group 24

Aaron Myran
Ihsaan Patel
James Fallon
Tyler Finkelstein

Introduction

Is it possible to predict a movie's genre based solely on its poster? What if other data points, like a movie's cast size, popularity, and even tagline are included? In this project, we applied a variety of data science and machine learning techniques to movie posters and metadata in order to predict a movie's genre.

The project took place in 4 stages or milestones:

1. Data collection and exploratory data analysis
2. Determining methodology for structuring training data (both posters and metadata)
3. Using traditional classification models to predict genres
4. Using deep learning techniques to predict genres

For further information and to view the actual code used, please visit our github repo.

Collection and Exploration

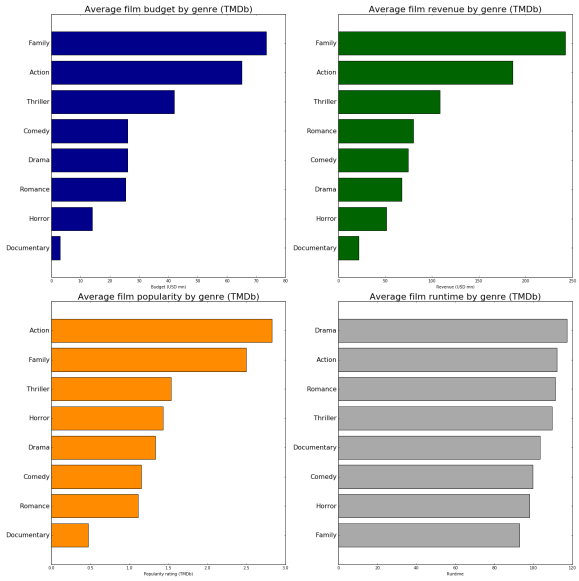
Data collection

We leveraged two main sources of data: TMDb (The Movie Database) and IMDb (The Internet Movie Database). We first used the TMDb API to access its list of popular movies, which totaled 19,000+ movies, and then pulled more detailed information about each movie, including its IMDb id, using the TMDb API as well. Lastly, we augmented this data by aligning the films with their aforementioned IMDb id, and pulling down additional details on the movies from IMDb using the IMDbPY library.

Given the large size of the dataset and the storage space required for each poster, we decided to filter down the dataset to a more manageable size. We hypothesized that there could be fundamental differences between english-language and non-english language films, and any type of text feature engineering with different languages could be hindered as well. Therefore, we filtered out the 7,000+ records where the movie's language according to TMDb was not english. Some of the remaining movies did not have a poster or genre, in which case they were also filtered out, leaving a final dataset of 11,678 records.

Exploratory Data Analysis

We performed some EDA to better understand how certain features varied by genre and to confirm prior intuition that these features could be useful as predictors in the models we were planning to use. As the chart to the right illustrates, certain genres (Family and Action) have higher budgets and revenues than others (Horror, Documentary), but this does not necessarily hold true for popularity (Horror is #4) and runtime (unsurprisingly, Family has the shortest runtime).



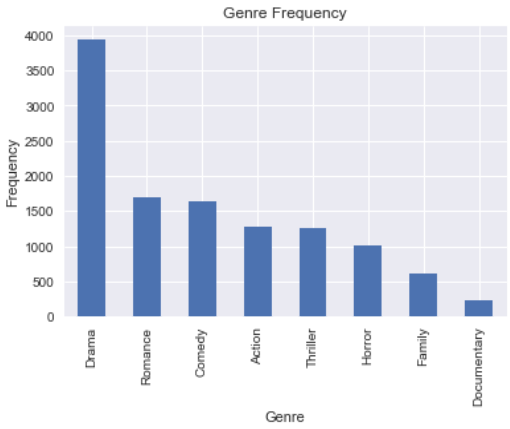
Methodology

The main methodological challenges at this point in the project were 1) how to structure our response variable and 2) what predictors to include.

Structuring the response variable

The issue of response variable posed a number of unique challenges. This was not a binary classification problem - we needed to predict movie genre across many categories as the EDA showed. Further complicating this challenge, the two databases we used had different genre labels and both allowed multiple genres to be assigned to each film. We settled on a multi-class approach as the various evaluation metrics would be easier to interpret and compare between various machine learning models.

To assign a specific class to each movie, we first took the intersection of the TMDb and IMDb genres for each movie, based on the idea that if both databases assigned a specific genre or genres to a movie, then this genre was a more accurate representation. For those movies with multiple genres in the intersection, we initially decided to use joint genre categories (i.e. “Romance” and “Comedy” would be “Comedy - Romance”). However, the machine learning models were not able to distinguish these very well as compared to single-label genres, so instead we selected the genre that occurred more frequently in the rest of the data.



This approach still left deeply imbalanced classes, with some genres (Drama, Comedy, Romance, Action, etc.) having 1,000+ records and others (War, Westerns, etc.) having <100 records. We therefore consolidated genres based on the team’s prior beliefs of similar themes, for instance combining Science-Fiction, Adventure, Action, War, Westerns, and Fantasy under “Action”.

While there are still large imbalances between the genres, we believe there is enough of a sample size to train a model to distinguish between them and any further balancing could be handled using the in-built model functionality (further discussed below).

Choosing Predictors

Given the relatively large dataset, we decided to include a large number of features, selecting them on the basis of domain knowledge and from insights gained during EDA. In addition to posters, we included features related to resource level (budget, # of production companies / countries), text descriptions (overview, tagline), popularity (revenue, votes), date (release month and year), runtime, size of various departments (art, cast, etc.), and a number of others for a total of 74 features.

We created features from the text fields by clustering them based on keywords. To do this, we first transformed each text field into a tf-idf matrix which gets word counts for each record and then scales down the word counts based on the occurrence of the word in the entire training set. As part of the tf-idf transformation we "stemmed" the words ("fishing", "fished", "fisher" would all be reduced to "fish"), filtered out common english terms ("a", "the", etc.), used n-grams of lengths 1,2,and 3 ("John","John likes", "John likes movies"), and limited the max dataset frequency of a term to 10% (essentially the size of a genre as there are 8 genres) and the minimum dataset frequency to 5 (based on previous models we'd seen).

With the transformation complete, we applied k-means clustering to the tf-idf matrix, tuning the number of clusters based on whether the cluster key-words made intuitive sense with the genres in that cluster. For instance, overview cluster #3 had key words of "murderer", "town", "killer", "investigator", and "detective", and the top genres in this cluster were Thriller and Horror. For overview cluster #4, which had key words of "Love", "falls", "woman","man", and "story", the top genres were “Romance” and “Drama”.

Non-Deep Learning Classification Techniques

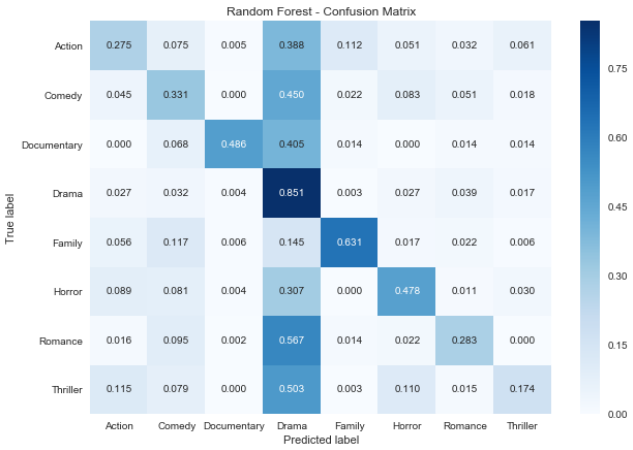
We first attempted to predict movie genres using only metadata and two tree-based models, random forest and xgboost, which have proven to be effective in winning Kaggle competitions.

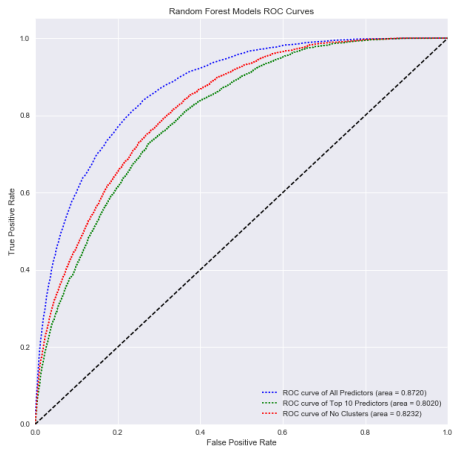
Random forest essentially grows decision trees while randomizing the subset of predictors used to grow those trees. The model then takes the mode of the outcomes of all the trees and records that class as the model's prediction. Xgboost is a gradient boosted model, which, like random forest, uses decision trees. While random forest grows trees independently of other trees, xgboost looks to minimize the residuals produced by the model by growing additional trees based on the residuals of the previous trees. This can lead xgboost to perform better than random forest if tuned correctly but also makes it more prone to overfit if the model fits the residuals too well. Tuning the XGBoost model takes longer as well since trees are not grown separately and there are more parameters to tune.

Both models’ hyperparameters were tuned using a gridsearch and 5-fold cross-validation, with the f1 score as the evaluation metric to optimize for given the large number and unbalanced nature of the classes. Both models do provide a hyperparameter to automatically balance classes, so this parameter was tuned in the gridsearch as well. Train and test data were shuffled and split 70% - 30%, respectively.

Random Forest

The optimal hyperparameters chosen by gridsearch were max_features = auto, min_samples_leaf= 1, class_weight = balanced_subsample, max_depth = 100, and n_estimators = 500. The confusion matrix for the test set showed that the model performed well on predicting Dramas, primarily because it picks Drama as the genre for most classes. However, the model still does relatively well for the Family, Documentary, and Horror genres.Of particular concern is that the training set yielded 100% accuracy, indicating potential overfitting during training. However, when we ran the model using smaller amounts of predictors (filtering based on feature importance), the ROC curves showed that the model using all of the predictors was still superior.

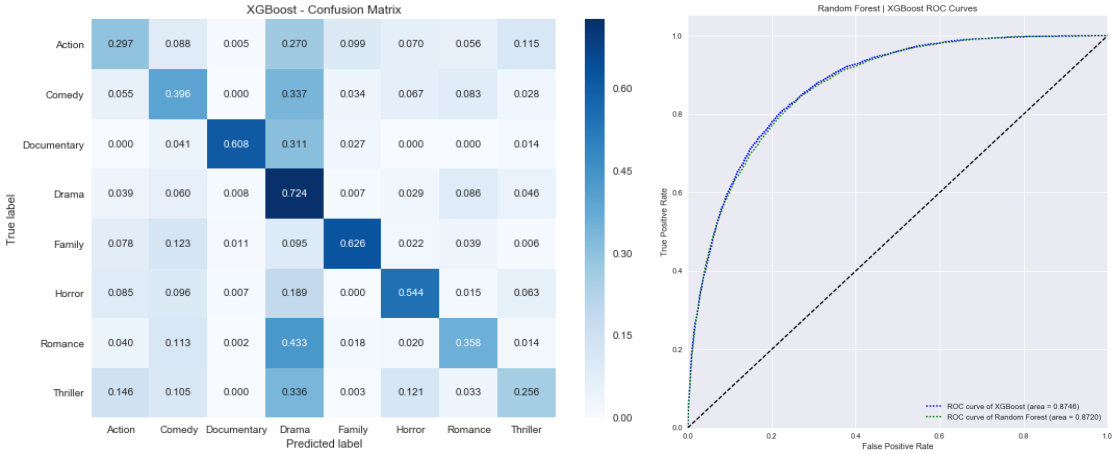




XGBoost

The optimal hyperparameters chosen by gridsearch were gamma = 0.2, min_child_weight= 0, max_depth = 5, subsample = 0.6, colsample_bytree = 0.9, reg_alpha = 0.1, eta = 0.1, and n_estimators = 1000. Similarly to the random forest model, the xgboost training model had ~100% accuracy, but the test set confusion matrix showed the xgboost model was better at predicting non-Drama genres.

Both models performed similarly on a number of metrics, with the xgboost slightly better in terms of F1 score and AUC and random forest slightly better on accuracy, log-loss, recall, and precision.



Deep Learning Classification Techniques

Data Pre-Processing and Model Parameters

Before running any deep learning models, we needed to process the posters into a model readable format. While the downloaded posters had dimensions of 750 (height) x 500 (width) pixels, we quickly realized that this was too big for any of our models to handle given the large dataset size. We initially tried to reduce dimensionality by converting the images to grayscale (3-D to 2-D), however visual inspection of posters indicated that it was hard to differentiate between poster genres without color. Therefore, we kept the color but reduced image size to 150 x 100 pixels (we also tried 225 x 150 pixels, however performance did not improve and the models took longer to execute).

With the data processed, we then trained and evaluated 3 deep learning models using a GPU cluster on AWS: (1) a CNN trained on images only, (2) a CNN pre-trained on other image data, and (3) a combined network trained on both images (CNN) and metadata (MLP).For each model, we used Adam (adaptive moment estimation) as the stochastic gradient optimizer, batch size of 10, balanced classes by setting the ‘class_weight’ parameter in Keras to ‘auto’, and 200 epochs. For the learning rate, we used a decay function where the learning rate started at .001 but was divided by 10 if the validation loss did not improve in 10 epochs. The models rarely ran through 200 epochs due to an early stopping function that ended the training if validation loss did not improve in 20 epochs.

CNN Trained on Images Only

We experimented with a number of different structures for the non pre-trained CNN, eventually settling on multiple sets of two convolutional layers and one max pool layer with consistently increasing the number of kernels in the convolutional layers. We steadily increased the number of kernels because

they were dealing with reduced image data, allowing us to extract more features. We stopped adding convolutional layers once the GPU's RAM ran out. Once the convolutional arrays were flattened, we included 4 dense MLP layers interspersed with dropout layers to prevent overfitting and ensure the log-loss function did not explode on the test set.

Unfortunately, we were not able to get the model to predict anything besides the dominant class, resulting in an overall test set accuracy of 0.3454 . This could be for a number of reasons, including lack of distinguishable genre information in movie posters and/or systemic problems in the dataset and modeling, however we were unable to discover a particular reason.

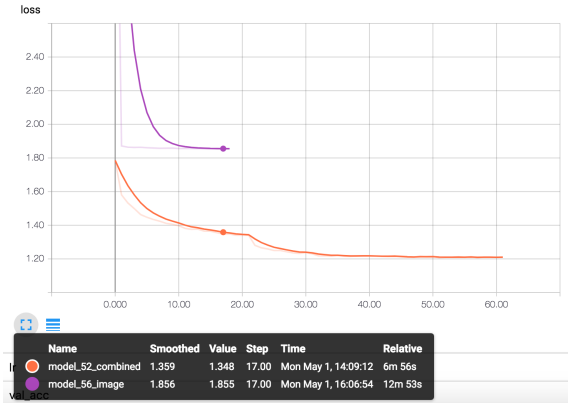
Pre-Trained CNN on Images Only

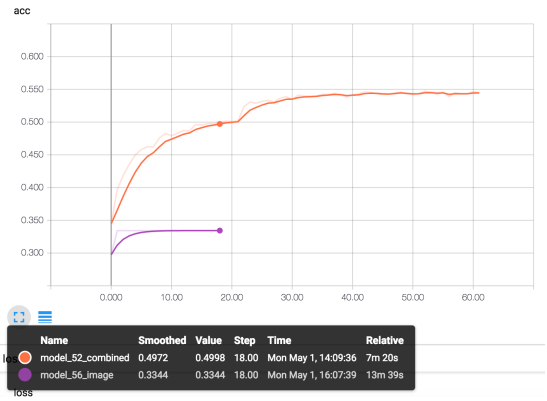
We then trained a new model using a pre-trained CNN provided by Keras, VGG16, and the same MLP layers as the non-pretrained model. While this model also only ended up predicting the dominant class, it's log-loss was significantly worse than the non-pretrained model, as shown to the right:

This poor performance is likely due to the fact that we froze all of the CNN layers, meaning the model was using features from other images to try and predict genre. One thing to test in the future would be to un-freeze some of the higher level CNN layers so that they could be trained on the posters themselves.

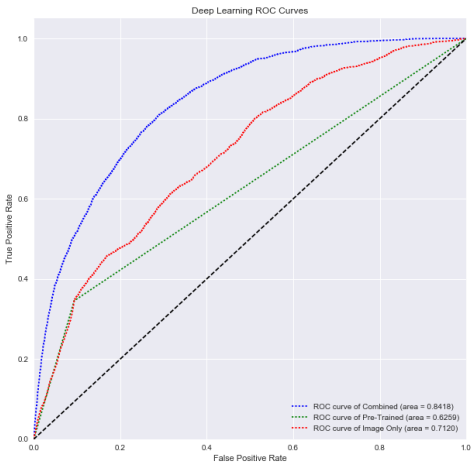
Combined Network

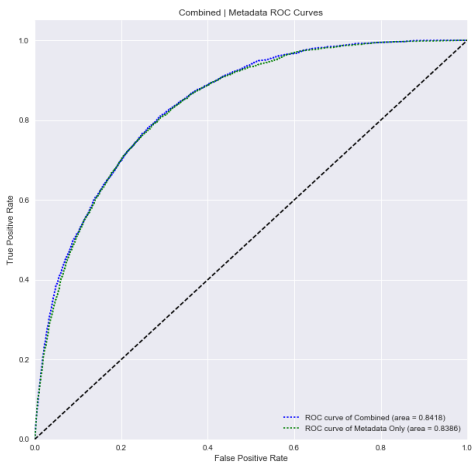
Given the poor performance of the CNNs, we included metadata as well by building an MLP network for the metadata and merging it with the non-pretrained image network. To avoid the aforementioned overfitting issues, we interspersed dropout layers in the metadata MLP. The model performed significantly better, achieving an accuracy of and a lower log-loss than the CNN's by themselves.





However, this performance appears to be based solely on the metadata model, as a model using only metadata performs nearly identically.

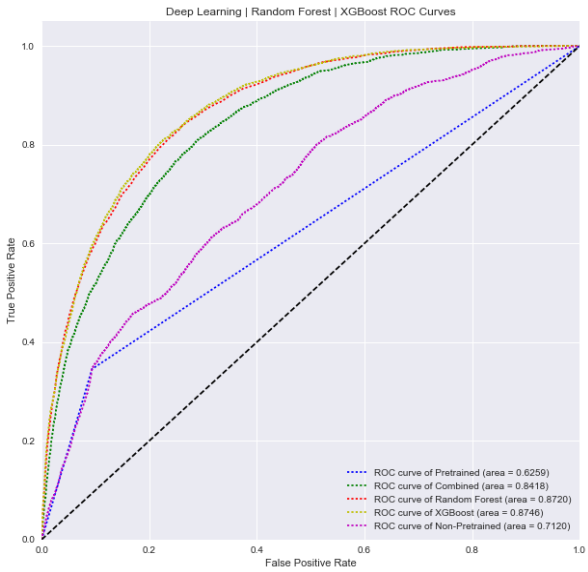




Conclusion

None of the deep learning models perform as well as the traditional classification models on any of the evaluation metrics, although the combined network does come close. This is likely due to the aforementioned difficulties training the CNNs and a potentially suboptimal structure for the metadata MLP network, which is harder to determine than using the gridsearch for the traditional models.

Going forward, further investigation should be done on the poster classification task, specifically to determine whether it is in fact possible to distinguish genres based on posters. In addition the metadata / combined network should be further optimized.



	Random Forest	XGBoost	Non-Pretrained Network	Pretrained Network	Combined Network
Accuracy	0.5104	0.5056	0.3454	0.3454	0.4569
Log-Loss	1.3665	1.3722	1.8412	22.6108	1.4920
Precision	0.5139	0.4928	0.1193	0.1193	0.4456
Recall	0.5104	0.5056	0.3454	0.3454	0.4569
F1	0.4764	0.4909	0.1773	0.1773	0.4425