Project Report
Predicting The Reach of a Show

STA-208 Project by Rishad Raiyan Joarder

Objective

The number of viewers any show/movie will reach is an important metric for taking the business decisions for production houses and streaming services. For this project, we try to determine the reach of a show using various types of features ranging from genres, languages and all the way to release dates. Plus, we will also look a bit deeper into few of the features and their effect on the prediction.

Visualizing The Data

Our dataset contains various types of data including categorical, numerical, Boolean etc. for different movies and shows streaming on Netflix. Although Netflix does not publicly reveal their viewer count, we decided to use each show's IMDb vote count as a surrogate for the resultant variable.

Missing Data

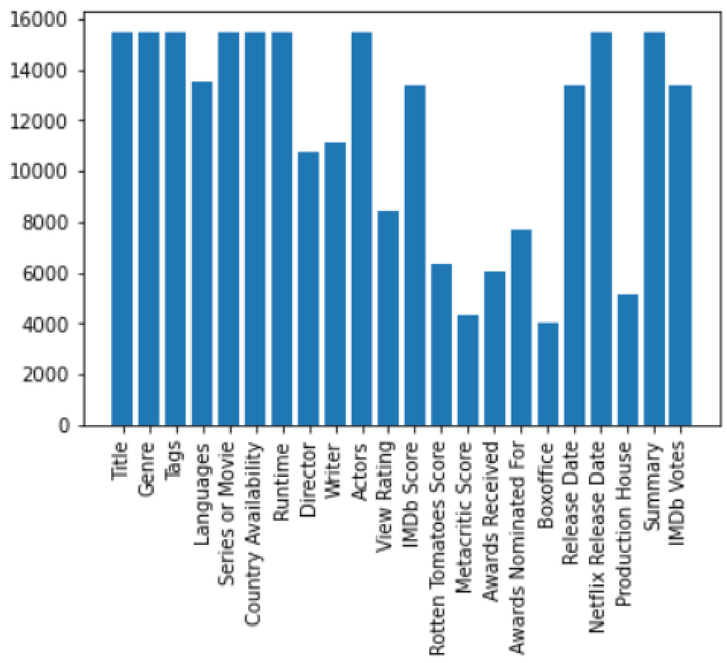


Figure 1 Features/Columns from the Data plotted along with their non-null data count

The above figure shows us the data columns we are working with along with the number of data count available for each column. Where the white parts refer to the unavailability of data. From this plot we get an idea about the first issue about working with this data. The presence of many missing feature data per entry.

Multi-Category

Looking at each of the features a bit closely, we notice that some of the features can belong to multiple categories. For example, lets plot the data count for genres.

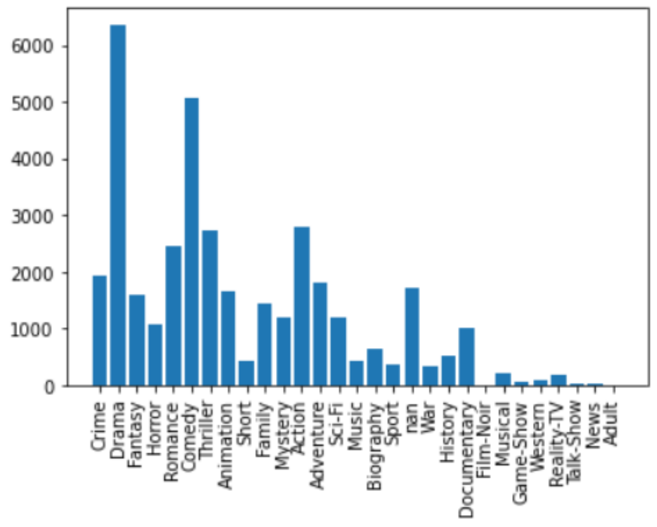


Figure 2 Data Count vs Genre for all the entries present

Now if we plot which genre each entry belongs to, we can see that many the entries belong to multiple genres. Like for example, a show can easily belong to comedy and drama at the same time.

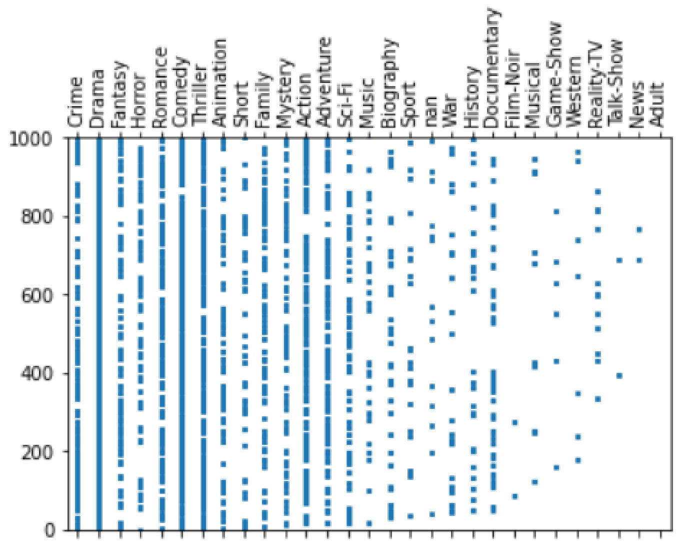


Figure 3 The Sparse matrix representation for the first few entries and their Genres which cleary shows that some of the entries do indeed belong to multiple genres

Sparsity

When you think about any show/movie one of the very first things that come to your mind would be the cast. The actors in a show/move is a categorical variable. But compared to the number shows out there, the number of shows a certain character appears in is extreme small. This ultimately leads to features like these having a very sparse representation as showed below.

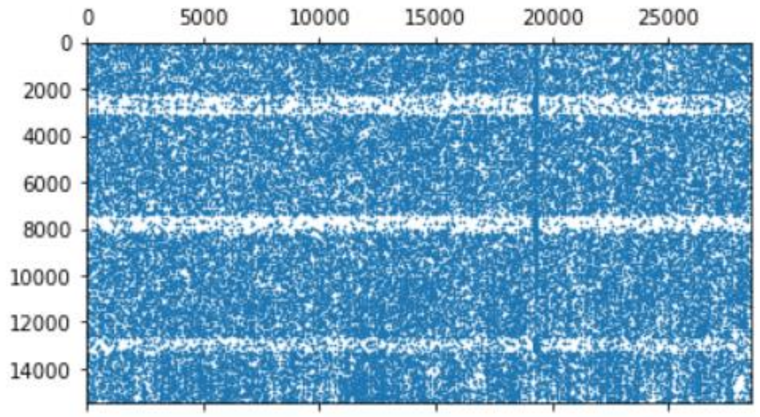
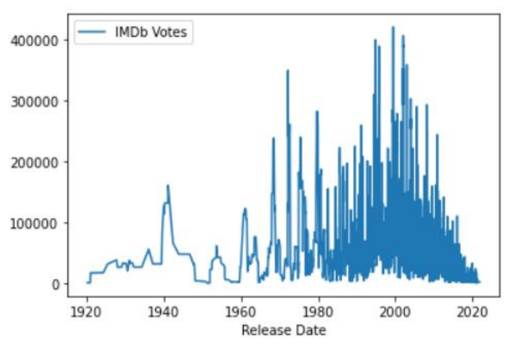


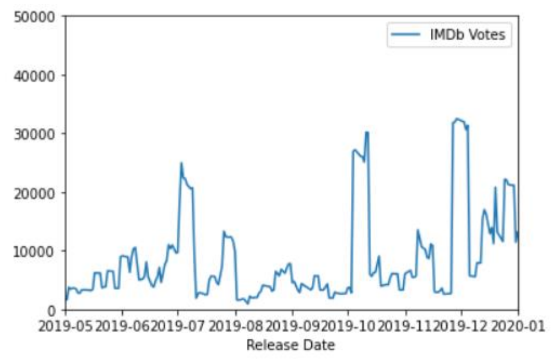
Figure 4 One Hot Encoding Representation of the Actors feature where each row is a different movie/show, and each column is a different actor

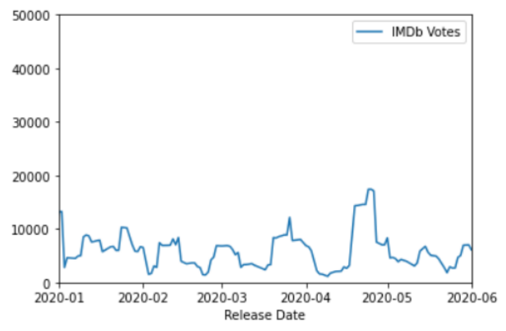
This is an extremely large matrix with more columns than rows. Note that this is for only a single feature. When stacked together with the rest of the categorical features, the combined feature matrix will be extremely hard to train. Therefore, we need a process to reduce the dimensionality of this feature set.

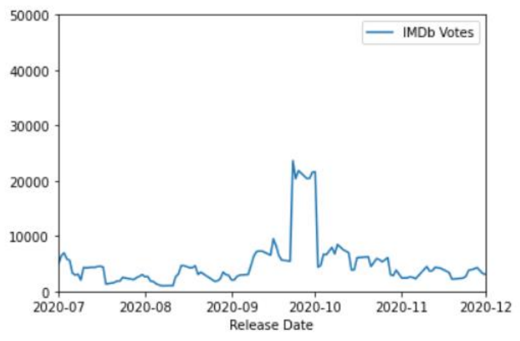
Timeseries Data: Release Date

Plotting the resultant variable with respect will give us a good idea about the trends and consequently how to handle the datetime data associated with our prediction. The bottom plots the average of the resultant variable at each date. We first give an overall view and then zoomed in views at a few different ranges. And we end up seeing a type of pattern. This data must be encoded in a way so that we can exploit this patter in prediction.









The resultant varaible vs. release date. First an overall view and then zoomed in at a few different ranges. The zoomed in plots show a type of monthly trend in data with certain peaks near the end and the beginning of a month

Other Data Types

The dataset also contains a few different features that we are excluding for now, but it might be useful for future endeavors. For example, each show/movie contains a summary text which could be processed using some type of NLP algorithm to improve predictions. The movie poster might also be a predictor since people sometimes base their decisions of watching something based on the posters. Also, since we wish to make predictions for future shows, we exclude some features like box office and ratings etc. since these will not be available.

Evaluation Method

Before diving into different types of imputation systems and encoding schemes, we would like to use this section to explain the metrics we used to evaluate a method.

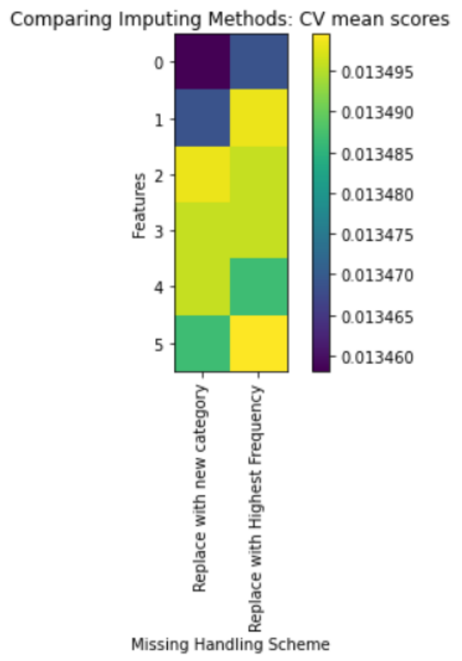
Since we are mainly evaluating imputing and encoding schemes, we use a gradient boosted tree with 120 trees as the predictor for each case. For each case, we calculate 5-fold cross validation scores, in this case MSEs. We then compare the mean and the variance of these CV scores. Any model with a low mean CV-score is an estimator with low bias. Any model with a low variance in CV scores has a low variance. This helps in detecting over fitting. We want to make sure our model has both low bias and low variance. For most of the method evaluations, we tend to lean on these two values. Since the final model is the same, this should give a good comparative analysis between different methods.

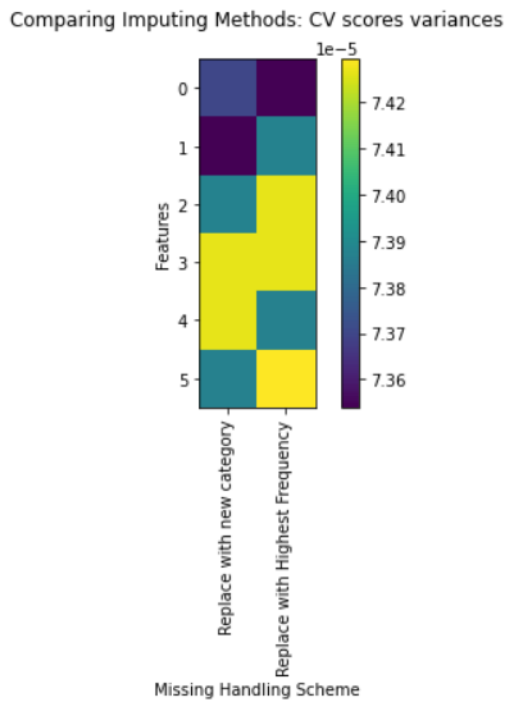
Handling Missing Data

In this project, we try out only 2 separate imputing techniques.

1. Replacing the missing features with a new unknown category. This is essentially the same as having a 'zero' label
2. Replacing missing features with the most frequent category

The results are shown below for a few different features. (Lower is better)





Both these imputing schemes produce very similar results

Each Row in the above matrices correspond a different column/feature and the columns correspond a different imputation scheme.

The first matrix shows the K-Fold CV scores (MSE) mean and the second one shows the variation in the CV scores (MSE).

Looking at these numbers, we conclude that both of these schemes pretty much have similar effects, and we cannot prefer one over the other.

Trying Out Different Types of Encoding

How we choose to encode the various types of features availabe in the dataset dictate how well the prediction algorithm can perform. Encoding schemes provide a few challenges.

1. A lot of the feature data has a very large number of categories. Keeping all of them is impossible. Training a regressor on a such a sparse matrix will not yield any valid results. Thus we need to keep only the important categories and drop the rest. We need to come to a trade-off in how many categories we choose and how we choose them.
2. The dataset includes datetime data. We need a way to process it in such way that makes sense and helps with the predictions
3. The dataset hass missing data which have been imputed before reaching the encoding stage. So the encoding scheme has to be robust enough so that any single feature does not dominate the output

Figuring out a scheme that deals with all these challenges at once is a difficult task. So, to make the problem a bit simpler we make some assumptions.

1. The features are independent
2. We choose a XGBoost regressor, try a 5-fold cross-validation using MSE as the scoring parameter and we base the judgement of our encoding scheme based on the mean and the variance of the MSE score from these 5 trials. In other words, a good model should have a low mean of MSEs over each of the cross-validation trials. This resembles a low bias. The variance between these MSEs should also be low. This resembles a low variance (Quick Note about Cross-Validation: for CV, we split the dataset into k-folds, train our model with (k-1) folds of the data and measure its MSE against the remaining test fold. We do this process k times with a different testing set each turn while also resetting the model at the beginning of each trial. By default, we set k to 5 for time considerations. A larger value would give us more precise values but also would take longer computation time)

Based on these assumptions, we take the feature columns one at a time and drop the rest and train regressor based on this. We try out different encoding schemes and vary the N parameter (How many categories to keep per feature) within a reasonable range and compare them with each other as well as with the no categories dropped condition.

Working With Categorical Features

This includes features such as a movie/show's 'Genre', 'Tags', 'Languages', 'Regions Available' and so on. On top of the vanilla one multi-category one-hot encoding we try a few more encoding methods and vary the N.

1. Keep all N-topmost occurring categories. Given the data, we calculate which categories have the highest occurrences, and only keep those in the encoding scheme. Any other categories are regarded as the 'zero - class'. An entry might have multiple of these most-occurring categories. In that case, we keep them all. (Extra: if a class has multiple categories, lowering N will take them out one by one until there is only one category left per entry)

2. Keep N-topmost impactful categories. This encoder goes over the data and groups it by category and then calculates the average value of the resultant/target variable for each group independently. Essentially, it is a measure of how much a single category would affect the target. Then we only keep the categories with the highest average and drop the rest.

A Baseline

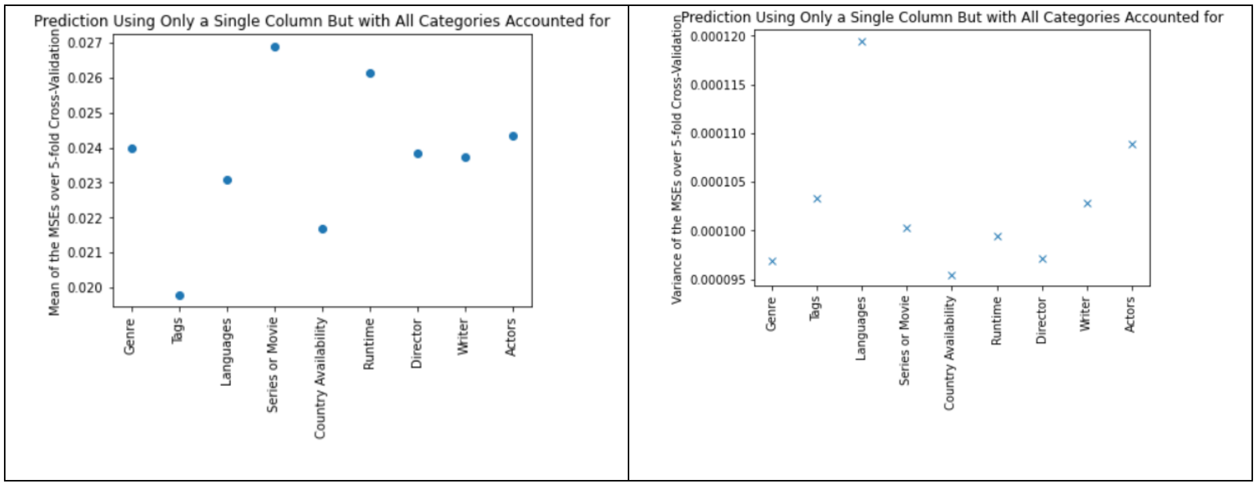


Figure 5 Baseline Scores of mean and variance for regression which considers all categories for each individual feature separately. We will be dropping a few features and comparing the results against the baselines

The two curves above show the performance of the vanilla multi-category one hot encoder for working with column at a time. The first curve shows the mean of the MSEs for our XGBoost model for the 5-fold CV. The second one shows the variance in those MSEs. We take these as the baseline since they do not drop any of the categories. Although the mean of MSE is low, this method tends to overfit and display a high variance. This is also an artifact of having such many categories.

Another takeaway from these plots is how well each individual column can fit the data. For example, the "Country Availability" feature has both low mean and variance meaning it is a very good estimator of how much reach any movie/show will get. From common sense that makes sense. If a show is available in countries with more viewers, it will have a larger audience. The same goes for the "Tag" parameter. Meaning specific tags would net you higher viewers. On the other end of the spectrum, we see that "Movie/Series" has a very large mean MSE. This gives a hint that whether your show is a movie or series does not impact reach as much say the other features.

Performance of Different Encoders Visualized Per Category

(All the graphs can be found in the notebook. Here we only show a select few)

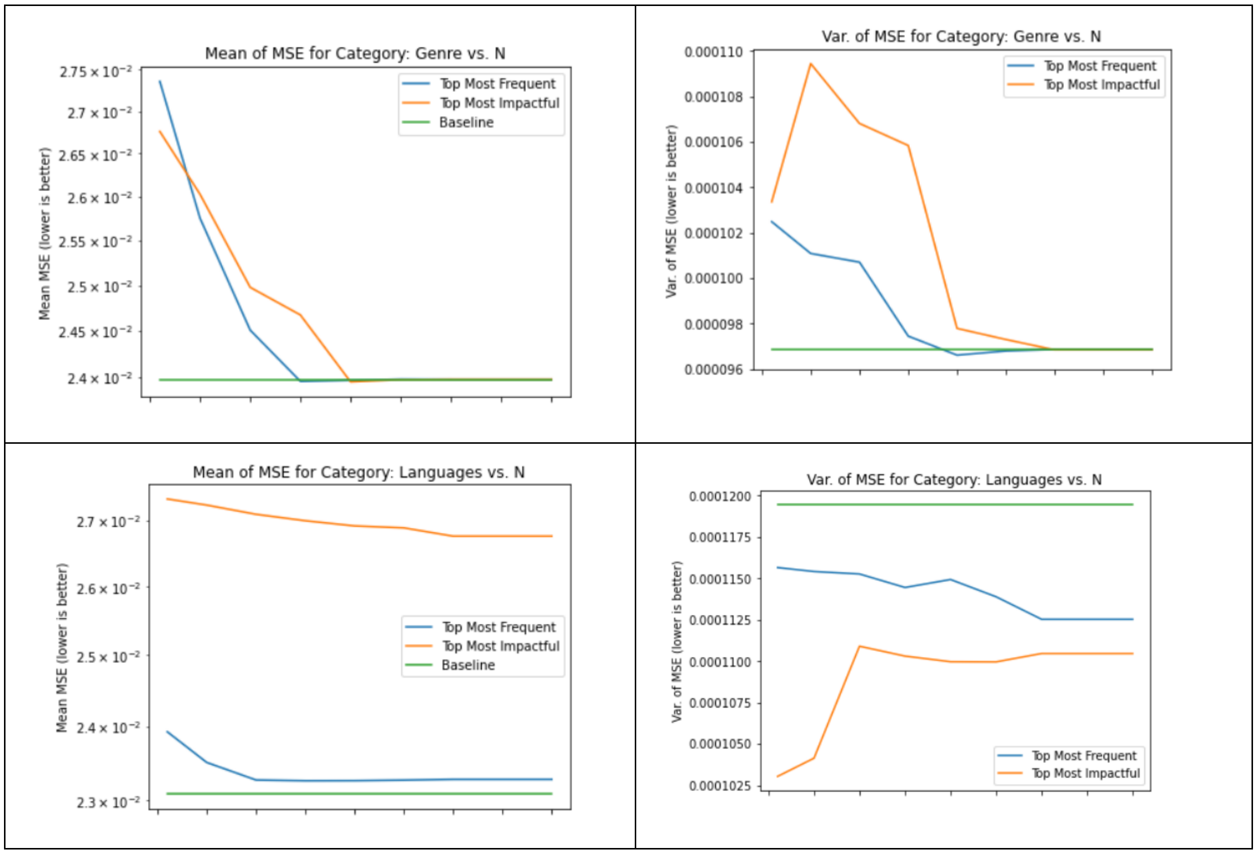


Figure 6 Showing the performance of different encoding schemes with using only a single column for regression. (Showing different features per row with mean CV scores in the first column and variance of CV scores in the second column)

The first thing to notice from these plots is that the errors pretty much flatten out for N greater than 10 - 20. So, we can actually just work with *N = 20* for the regression and assuming we have a large enough dataset, we should be able to capture the correct features!

Another interesting thing is that for the extremely sparse features like "Actors", "Directors" and "Writers", the error curves flatten out as early as *N=5*. This essentially means that except the top few actors/writers/directors, the reach of a show/movie does not depend greatly on its cast/crew.

The Baseline contains all categories for a given feature and it's going to be impossible to fit or even work with the data when we consider all the features. So even though the baseline does have the lowest errors, it is not something we can use. Also, for some of the features, the other encoding methods have a lower variance since those are calculating lower number of categories.

Surprisingly, we see that in almost every case the error for the "Topmost Frequency" is lower than "Topmost Impactful". However, "Topmost Frequent" also shows large variance.

Despite this, for most categories, the variance is around the range of *1e-4*. Using this value and the values used for standardization, this corresponds to roughly a std. deviation of *15.34*. Whereas our surrogate variable for the reach, has a max value of roughly *235,000* and a mean of *43,000*.

Considering these values, our variance is extremely small. So, for the rest of the analysis, we will use Top-20 most Frequent Categories for all the features.

The project code has these as the default schemes!

Date-Time Data

We have 2 different datetime data included for the dataset. The 'Release Date' and the 'Netflix Release Date'. Plotting the viewer data vs. datetime shows us a that there is a monthly as well as a yearly correlation in the data. For most of this section, we will work with the Show's Release Date rather than its' Netflix Release date.

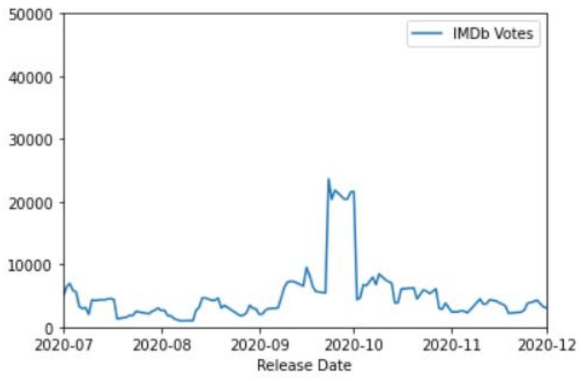


Figure 7 Relationship between resultant variable vs. release date

Trying out Year Encodings

The year is encoded as an offset from a chosen year and then normalized between 0 to 1. Below we tried out the performance of our model without the datetime data and with the datetime data using different offset years. For the rest of columns, we use the best encoding we have found so far(saved as the defaults in the code so they do not need to be explicitly specified).

A quick formula for the year encoding

$EncodedData=(Year-Offset)/NormalizationFactor$EncodedData=(Year−Offset)/NormalizationFact

or

Where the normalizing factor is chosen to make the offset year be encoded to 0.0 and the most recent year (2021) to be encoded to 1.0. The results are shown below

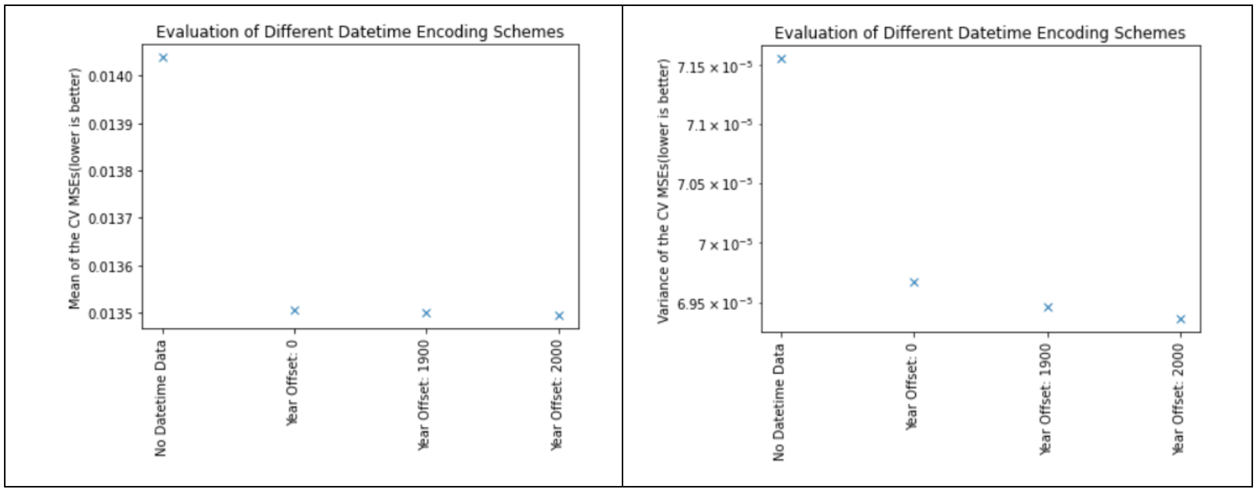


Figure 8 Performance of not using date time in prediction and using different encoding schemes for datetime. Clearly, using the datetime improves prediction but using different schemes does not improve results that much

We notice 2 things from here.

1. Having the Datetime data does indeed give us a better prediction both in terms of bias and variance. Even though the impact is not very high
2. Having the year offset set to values around 2000 yeild better results than no offset at all(0)

Month and Day Encodings

From the above plots for 'IMDb votes' vs 'Release Date', we a type of trend with the day of the month. To encode this type of feature, the day is encoded as an ordinal variable with values between (0, 1] signifying the start and the end of the month. We tried two different month encodings.

1. Encoding the month as a One Hot vector
2. Encoding the month as an Ordinal variable between (0, 1] (0 excluded) (The rest of the encoding scheme is kept to the defaults with the year offset being 2000)

The results are shown below

Encoding Scheme	Mean of the CV MSEs	Variance of the CV MSEs
No Datetime data	0.014039	7.155e-5
One Hot Encoded Month data	0.013453	6.95e-5
Ordinal Encoded Month data	0.013497	6.94e-5

Analyzing the effect of Day of the Month, Month and Year

To figure out the effect of these three parts individually on the resultant variable, we run an OLS and check the beta co-efficient to figure out which has a higher effect. For these experiments, all the other features/columns are dropped, and the prediction is made based only on the release date data. So, the larger absolute value beta co-efficient correspond to a larger impact. We try out both having all months combined into a single variable and having each month separately one hot encoded.

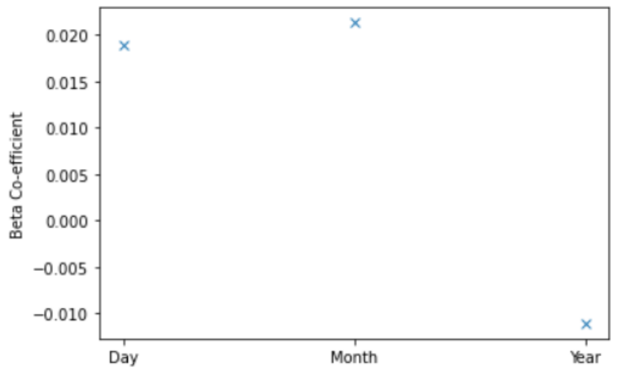


Figure 9 The effect of day, month and year on the prediction, Higher absolute value means a larger impact showing that month and day have similar impact while year has a much lower negative impact

The Beta co-efficient while fitting for day, month, and year. From these co-effs we conclude that the Day of the month and the exact month are very important in prediction where year has a much lower impact. The negative co-efficient for the year gives the impression that older movies have a larger reach.

Diving deeper into separating out the months to check the impact of each individual month from January to December, we get the below plot.

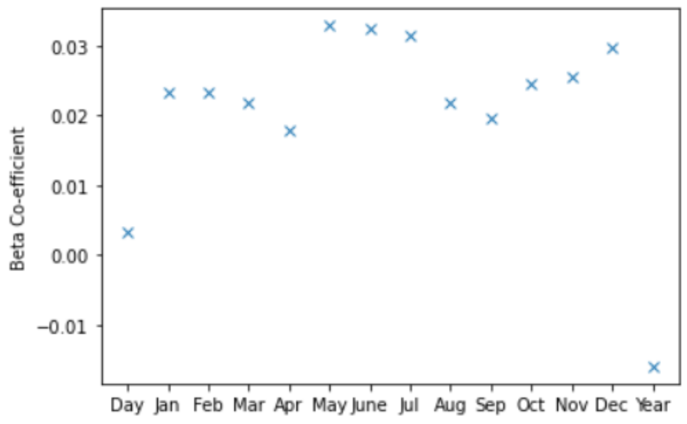


Figure 10 The impact of each individual month on the resultant variable. Showing that some specific months guarantee larger reach

From these betas, we observe that movies/shows coming out in May, June July and December get higher reach than the rest. This makes sense since they represent the summer months and the Christmas vacations. Whereas shows coming out in April have the lowest reach.

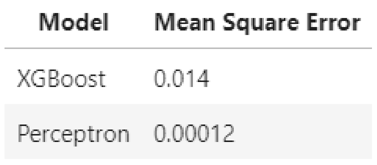
Numerical Data

We originally had a few different numerical features like critic/viewer scores, boxoffice etc. But since the goal of this project is to predict how well a new show/movie will perform in the future, it does not make sense to deal with these types of data as it is extremely unlikely to have these data. Rather they can be used to enhance/modify the dependent/resultant variable. That is something reserved for future works.

Besides these, for the rest of the numerical data, it is sensible to assume them as ordinal and they are just normalized between [0, 1] and used as a feature directly without any other types of processing

Trying Out a Perceptron

In this project, we tested out different encoding and imputation schemes using a much simpler data regressor. Now we use the best methods found from there onto a perceptron to get even better results. The different perceptron models which can be found inside the notebook. The final results are given below



Model	Mean Square Error
XGBoost	0.014
Perceptron	0.00012

About the Pipeline

Example codes for the data processing pipeline can be found inside the notebook titled pipeline_examples and each individual pipeline component has their own notebooks showing off what they do and how its done. In this section of the report, we would like to give a brief description of the parts of the pipeline. The main components of the pipeline are

1. Data Loading and clean-up
2. Imputing
3. Encoding
4. Model Evaluation

The data-loading part is straight forward. Additionally, it drops columns that are unnecessary. The imputing and encoding portions have modularity. Each column/feature has a separate encoding and imputing object attached with it that can be easily swapped out for a different one. The Notebooks Encoding Tryouts and Imputation Tryout demonstrate a few of these modules and how to use them with corresponding codes. For a more detailed look, try the documentation using an IDE for them.

Results

Some key observations from all of these are given here. Firstly, for simple imputation schemes, the overall performance does not change that much. Secondly, for very sparse categorical data encoding schemes, keep the most frequent 10-20 categories give essentially the same performance and even sometimes better performance than keeping all of them. This can be attributed to the smaller number of fitting parameters and the much lower risk overfitting. On top that, it also makes the model more robust to missing data to have lower categories. We also notice that there is a good correlation between the reach of a show and the month and day it was released in. Another thing to note that the imputation and encoding classes used in here can be extended and new classes can be easily implemented and swapped in for the current classes with minimal effort. This makes any type of future work in using this codebase extremely easy.

For encoding, we expected the target encoding to perform better. But for some reason, it failed in comparison to most frequent encoding. We also expected to get much lower variance from keeping the topmost categories across the board. This was true only for some of the features. That is also a bit unexpected

Future Works

1. Making use of NLP on the movie/show summary to improve prediction results
2. Handle missing data using regression while using the observable columns as the features and the missing one as the dependent variable.
3. Figure out a better measure for impact for the different categories withing a feature and use that metric to design a better target encoding scheme
4. Use CNN to process movie posters and use that data to make better predictions
5. The dataset came with a few different ratings from different websites for each show/movie. It would be interesting to see the impact of different feature on the movie rating. Actually, the modular design of our pipeline can very easily allow this change with very few lines of code