Movie Rating Prediction

1 Introduction

In this project, we are trying to give a model such that given some input information of a movie, we are able to give a potential prediction of the movie rating, i.e. a possible score that this movie might achieve. Pushing this question into a further extent, it is also possible to analyze what are the common features of high rating or popular movies.

To achieve this, we use three main approaches: Linear Regression, Decision Tree, Proximal Gradient. For the first and second approach, we are trying to output a numerical result, while for the third approach offers a analytical result.

2 The Dataset

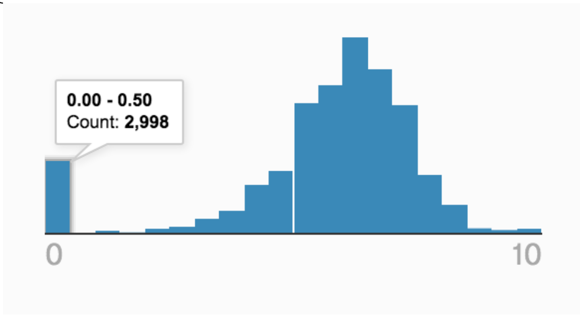
2.1 Description

We focus on the "movies_metadata.csv" file, which provides more comprehensive information of a movie such as production company, budget etc, of a movie. There are approximately 45.5k data points and 24 columns in this dataset. Among these 24 columns, there are two columns named "vote_average" and "vote_count", where these are correspondingly the **score** given by reviewers and the **number of reviewers** for each movie. This is the "target" for this model.

2.2 Messy

This dataset is messy. Three of the 24 columns have more than 50% of the data is missing (not counting zeros values, which are sometimes not informative). The rest of columns are as well complicated.

Take our target, vote_average for example, even though there are only 6 missing data, there are approximately 3000 data points that has value zero. Even though the data is not missing, this does not provide much information. A histogram given by Kaggle is shown below on the left:



As we can see, there are 2998 data points that has 0 value for vote_average. If we ignore the 0 values, we observe that the rest of the scoring is approximately Gaussian. Hence, we can see that we should also be dealing with zero values besides missing data. Similar situations can be observed in other columns.

3 Features and Preprocessing

3.1 Features Considered

Among the 24 columns, we only pick a portion of features we want to taken into consideration at this point. The six main features we taken into consideration is **Budget, Popularity, Release Date, Runtime, Genres, Production Countries**. We have also considered **Production Company**. However, we observe that there are too many categories existing, which makes the matrix large and sparse, not useful when doing Linear Regression.

3.2 Preprocessing

There are two main parts in preprocessing process. One is to solve the problem of "messy"; the other is to encode.

- **Genre** We use many-hot encoding to process Genre information. The reason for such encoding is because each movie can belong to several categories.
- **Popularity** Parse the data given in `float`. For those missing data, we replace them with 0.
- **Budget** Use Boolean encoding. We set three "levels" of budget: [0, 1, 20] $\times 10^6$ (dollars). For the mismatched and missing values, we replace them with 0.
- **Runtime** Use Boolean encoding. We compute 25%, 50%, 75% quantiles, and use these values

as reference when doing Boolean encoding. For missing value and zero value, we set them to the mean value to reduce variance.

- **Release Date** We only care which months are the movies released.We use one-hot encoding on month/quarter. For missing values, we randomly assign a month to it.

- **Production Countries** A quick examination into the Production Countries feature reveals to us that among the 45.5k movies, 21.1k of them are produced in the US. So, we replace this feature with a single vector indicating whether the movie is produced in the US.

3.2.1 Data after preprocessing

Below we included some graphs that illustrate the distribution of features after preprocessing the features. We see that after preprocessing, the features are distributed in a reasonable way than before.

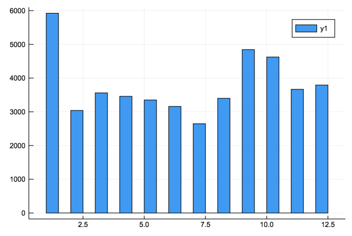


Figure 1: Histogram of Month

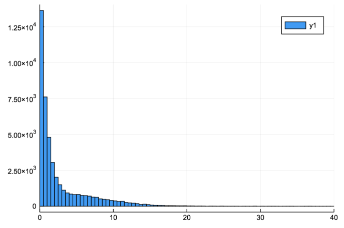


Figure 2: Histogram of Popularity

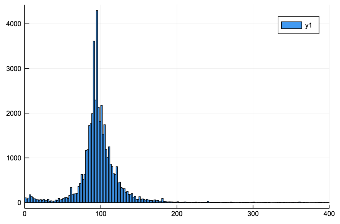


Figure 3: Histogram of Runtime

3.3 Preprocessing y target

To avoid useless movie ratings, we use the IMDB's weighted rating formula: Weighted Rating (WR) = $(\frac{v}{v+m} \times R + \frac{m}{v+m} \times C)$, where v is the number of votes for the movie, m is the minimum votes required to be listed in the chart, R is the average rating of the movie, C is the mean vote across the whole report.

The next step is to determine an appropriate value for m, the minimum votes required to be listed in the chart. We will use 95th percentile as our cutoff. In other words, for a movie to feature in the charts, it must have more votes than at least 95% of the movies in the list.

4 Linear Models

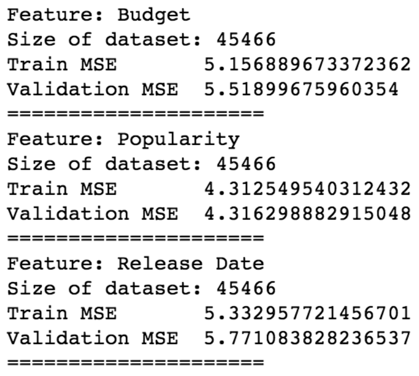
4.1 Prevent Overfitting

We split our dataset into train, validate, test parts, with portion 7:2:1. Before we test our output, we validate our model with validation set to prevent overfitting. That is, validation error is slightly bigger than training error; while both of them are relatively low.

We also use random seed, and shuffle our data before splitting.

4.2 Single Feature

We tried 5 out of 6 main feature individually to see which feature is most effective in predicting via Linear regression. We use **MSE** as our error metric. The result are shown in Figure 2 and Figure 3. By observation, we see that **Popularity** has the smallest MSE error. This indicates that Popularity it the starting point we want to pick.



```
Feature: Budget
Size of dataset: 45466
Train MSE       5.156889673372362
Validation MSE  5.51899675960354
=====================
Feature: Popularity
Size of dataset: 45466
Train MSE       4.312549540312432
Validation MSE  4.316298882915048
=====================
Feature: Release Date
Size of dataset: 45466
Train MSE       5.332957721456701
Validation MSE  5.771083828236537
=====================
```

Figure 4: Single Feature

```
Feature: Runtime
Size of dataset: 45466
Train MSE       5.03932152613925
Validation MSE  5.447464439268874
=====================
Feature: Genres
Size of dataset: 45466
Train MSE       5.1084894729629875
Validation MSE  5.5382583359176705
=====================
```

Figure 5: Single Feature

4.3 Three Models

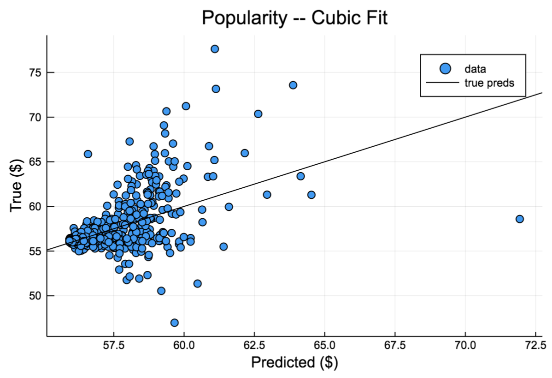
We plotted the data points **linear, quadratic, and cubic** with respect to y and see that Popularity and y is NOT simply linearly related. In fact, quadratic and cubic models, or even quartic model, might give better linear regression results. As follows, we fit all of them to see which has the smallest validation error. We find that **cubic** model has the least MSE error, and the fitting is shown below in Figure 4.

The validation error is 3.775, much smaller than single feature fitting, and still in a good range. We call this **cubic Popularity Model**. Now we add more features to Popularity and see whether there are some improvements.

4.3.1 Model I: Popularity, Budget, Runtime, Release Date

After a few tryouts, we figure out a first model that works better than cubic Popularity Model. When adding more features, we observe some small progress regarding the MSE error metric.

```
Feature: Popularity -- Cubic Fit
Size of dataset: 45466
Train MSE       3.5773601610781642
Validation MSE  3.7751873539057144
====================
```



We now have a tentative **Model I**, which includes $\boxed{\text{cubic Popularity}}$, $\boxed{\text{(linear) budget, (linear) date}}$.

The resulting plot is shown in Figure 5. The validation error for Model I is 3.6742, which is smaller than cubic Popularity Model.

We can actually add a little complexity in this model. We do a **pseudo-polynomial fitting** for Budget and Popularity. The choice of these two features are decided based on a bunch of experiments of several combinations. The reason we call it "pseudo" is because we only include a portion of polynomial fitting features, but exclude features such as "budget^3". We call this **Model I Beta**. The validation error is 3.5729. A lot better!

4.3.2 Model II and III:

Model II (and Beta) and Model III (and Beta)are both extensions of Model I (and Beta) adding different features. **Model II** is Model I + Production Countries, **Model II Beta** is Model I Beta + Production Countries; **Model III** and **Model III** are Model I adding Genre feature. The resulting plots are shown below.

Note that the validation error is smaller than the training error in Model III and Model III Beta. This is possibly because there is a few big outliers existent in the training set, and these two models happen to capture these outliers. The outliers are probably in Genres feature, because Model III and III Beta differ from previous models particularly by adding in Genres.

```
Popularity + Runtime + Budget + Release Date
Size of dataset: 45466
Train MSE       3.4813047466233034
Validation MSE  3.674248342579983
====================
```

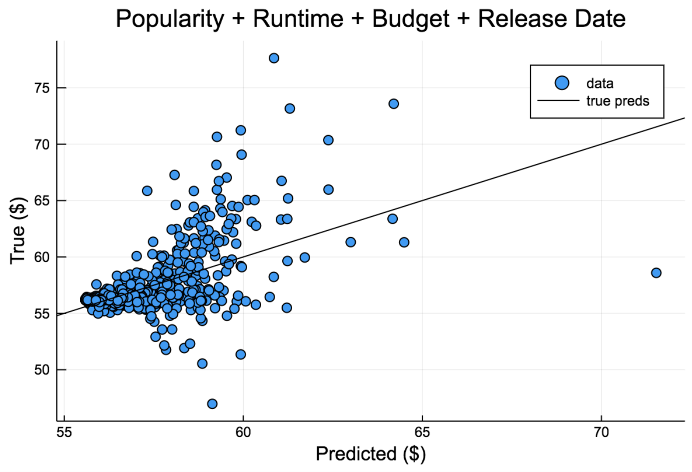


Figure 6: Model 1

```
Model II: Model I + New Production Countries
---------------------
Size of dataset: 45466
Train MSE       3.471749323916
Validation MSE  3.6590212880876067
=====================
```

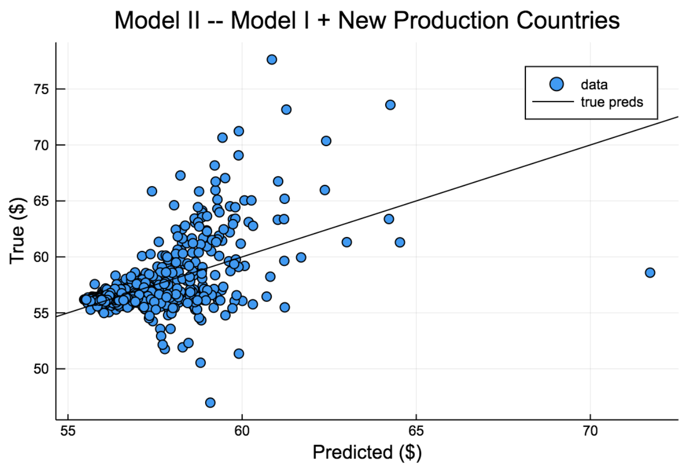


Figure 7: Model 2

```
Model III: Popularity + Popularity^2 + Popularity^3 + Genres + Budget
---------------------
Size of dataset: 45466
Train MSE       3.7860833722080147
Validation MSE  3.6046759805613604
=====================
```

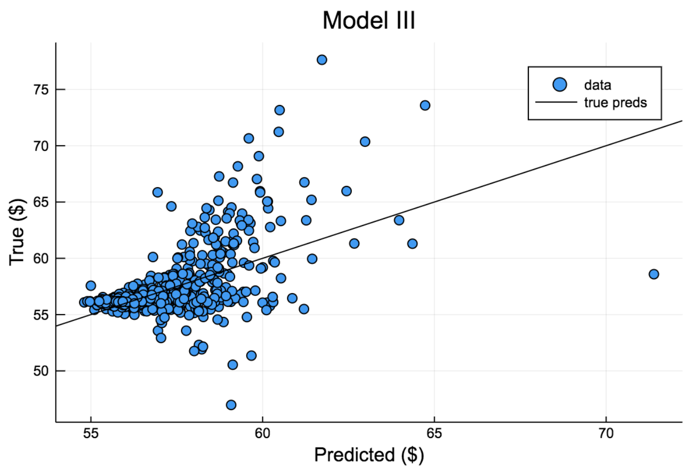
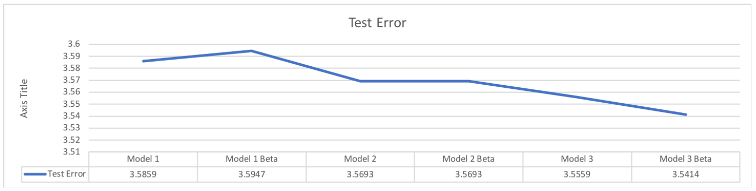


Figure 8: Model 3

4.3.3 Compare Test Error

Now we want to compare the test error among all the six models (including betas). The test error for the models are: 3.5859 (model I), 3.5947 (model I beta); 3.5693 (model II), 3.5693 (model II beta); 3.5559 (model III), 3.5414 (model III beta). The best model is Model III beta, with **test error** 3.5414.



	Model 1	Model 1 Beta	Model 2	Model 2 Beta	Model 3	Model 3 Beta
Test Error	3.5859	3.5947	3.5693	3.5693	3.5559	3.5414

4.4 Summary

Linear Regression is the most straightforward approach when dealing with this problem. As observed, the test error is pretty low. Since we have made sure that our model is not overfitting, we are pretty confident about this result. With that said, features have to be chosen carefully when doing the prediction.

5 Regression Tree

5.1 Introduction

Comparing to Linear Models (via Linear Regression) we have reasoned above, we now want to try a non-linear model. In this specific setting, we need to use Regression Tree (CART) to predict values in $\mathbb{R}$.

We reference course lecture note for further information, and we install Package `DecisionTree` to train our model. The Package documentation is https://github.com/bensadeghi/DecisionTree.jl.

Th advantage of Regression Tree (or Decision Tree in general) is that it is a nonlinear model, which means it has a wider acceptance of input data. At the same time, CART are very light weight classifiers and they are very fast during testing.

5.2 Regression Tree

5.2.1 The Model

We first try on Regression Tree. Train the model using training data, and see how the model is performing on testing data. In order to use the popularity feature more effectively, we converted our original popularity feature to a many-hot matrix. More specifically, the three columns in the popularity many-hot matrix are whether each popularity is greater than 75-quantile, 50-quantile, and 25-quantile popularity. Since we are dealing with regression tree, if a feature does not affect the outcome significantly, the tree will be less likely to use that feature. Thus, we simply use all of our features as our input.

5.2.2 Observation

The train error is approximately at average 2.857. The test error is approximately 4.229. It is slightly higher than train error, but it is still a satisfactory result comparing to linear regression.

5.3 Random Forest

Although our regression tree reaches a rather satisfying point, there are still space for further improvement. Indeed, we can do bagging on top of CART, i.e. Random Forest.

5.3.1 The Model

Similar to what we have done in the above model, we make use of the functions `Decision Tree` to build up a random forest model (using training data), and test that on testing data. Similarly, there is a function called `nfoldCV_forest(labels, features, nfold, nsubfeatures)`. For the newly included parameter `nsubfeatures`, which is referring to the number of features to consider at random per split.

Again, use MSE as the error metric. We compare the error with validation and test error.

5.3.2 Observation

The train error is approximately at average 2.779. The test error is approximately 3.756. By comparing the test error using Random Forest with that of regression tree, we can see that even though training error does not improve a lot, the overall test error of Random Forest is much better. Indeed, CART are usually not competitive in accuracy but can become very strong through bagging (Random Forests) and boosting (Gradient Boosted Trees). We are not using Gradient Boosted Trees in this setting, but we have observed an improvement when using bagging strategies.

We are confident with random forest. First, it has a low test error which is very similar to the best model of linear regression. Also, it is tolerant to unrelated data, so we don't need to choose over the best features.

6 Proximal Gradient

6.1 Introduction

In addition to linear models and regression tree method, we also considered proximal gradient methods. We basically tried combinations of (1) Quadratic loss (2) L1-loss (3) Quantile loss (4) Huber loss and (1) Zero regularizer (2) Quadratic regularizer (3) Non-negative constraints using (1) features used in model I, which are popularity, run time, budget, release date, genres and production countries, with offsets, and (2) only popularity, as popularity is the predictor that best explains movie ratings by itself.

6.2 Quadratic Loss

We started by using all the features of Model I with ZeroReg. Since Quadratic Loss is the same as using MSE as the error metric, we calculated the error using MSE directly. The training error is 3.97 while the validation error is approximately 4.03. Note that QuadLoss with ZeroReg is actually the same as linear regression, however the errors are slightly larger than higher than those generated by linear regression because in proxgrad method, we start off at $w = 0$ and look for the right w by gradient descent at each iteration. Using a stepsize of 0.1, the descent rate is really slow and it takes a large number of iterations for the algorithm to converge. We are already using $10,000$ iterations, but since some entries of w have magnitude of $e - 7$, the algorithm has NOT converged even though it has reached the maximum iteration. That explains why the validation and training errors in proxgrad are larger than those in linear regression. But we have reasonable confidence that given enough iterations, the algorithm should converge to the global minimum and reach the same errors as linear regression.

Next we tried using Non-negative constraint. The errors resemble those we got using ZeroReg, with training error being 4.05 and validation error being 4.11. The plots of these two regularizers are made as below for a better illustration. From the plots we can also verify that the two methods do produce very similar predictions.

We also tried the two regularizers using Popularity as feature along, but the results are not so promising, so we disregard these models in this case. And

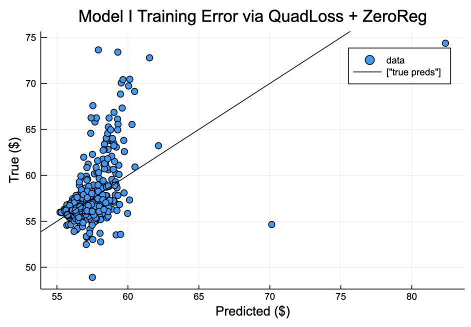


Figure 9: Quadloss with ZeroReg

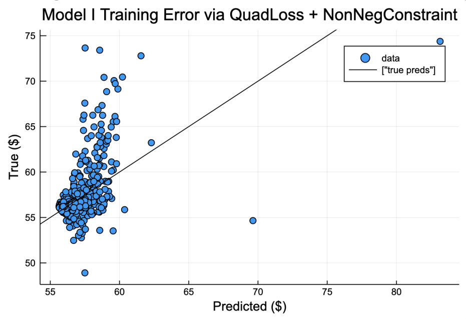


Figure 10: Quadloss with NonNegConstraint

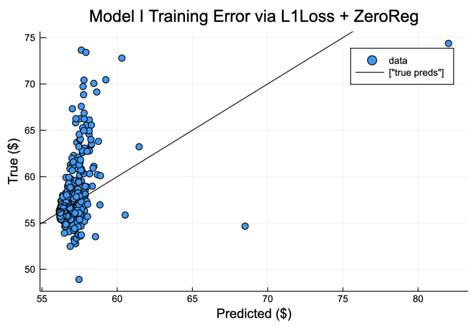


Figure 11: L1 loss with ZeroReg

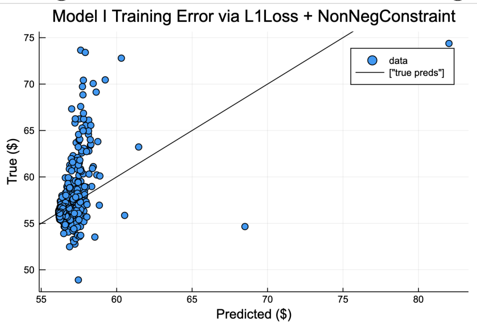


Figure 12: L1 loss with NonNegConstraint

the Quadratic Regularizer does not make satisfying predictions as well, and we disregard this regularizer, too.

6.3 L1 Loss

We then studied the data using L1 loss. First we tried using Quadratic regularizer and we get both training and validation errors over 2000. It seems that quadratic regularizer is not a very good regularizer for our data, so we no longer consider this regularizer in our further analyses. It is also reasonable not to use quadratic regularizer on our data, since some of the entries in w have magnitude really small (10e-7), the quadratic regularizer would enlarge these entries, thus making the predictions unaccurate.

Next we looked at Non-negative Constraint and Zero Regularizer using Popularity as the only feature. We have tried fitting the model with all features together, but the result was not so satisfying, so we decided to go with Popularity along. The training error and validation error for both Non-negative Constraint and Zero Regularizer are the same, i.e., 4.41 and 4.53, respectively. This is not a bad result, but still not as good as that in the last section. The error plots are shown as below.

6.4 Quantile Loss

We then experimented using Quantile loss. By trials and errors, we found that the mean of feature “Popularity” is approximately at the 0.7 quantile, so we chose $\alpha = 0.7$ in fitting the following models.

We started with using Popularity as the single feature, but did not get good results under Non-negative constraints, so we switched to using all features combined. Under Non-negative Constraint, the training error and validation error are 8.57 and 8.37, large enough for us to disregard this model. We then tried using Zero Regularizer and get 13.41 and 11.26 as errors. They are even larger than those under NonNegConstraints. So it looks like quantile loss may not be a good loss function to use in our dataset. The plots of the training errors are generated as below. Note that the errors this time look quite different from those before, as the data points are quite evenly scattered around the prediction line. Also, the two regularizers produce quite similar predictions as always.

6.5 Huber Loss

Finally, we considered huber loss. Since Non-negative Constraint performs similarly as Zero Regularizer, and sometimes slightly better, we decided

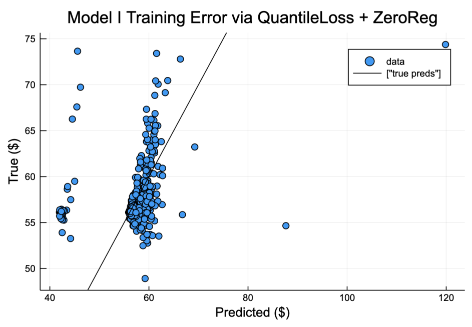


Figure 13: Quantile Loss with ZeroReg

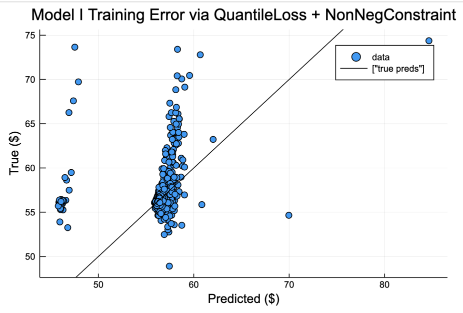


Figure 14: Quantile Loss with NonNegConstraint

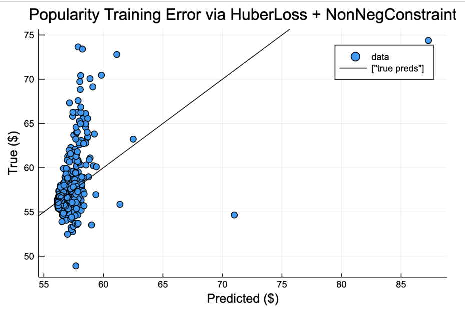


Figure 15: Huber Loss with NonNegConstraint

to go with Non-negative Constraint this time. We first tried using Popularity as the single feature and get training and validation errors as 4.48 and 4.34, and then tried using all features to get errors over 10. So the single feature performs better than everything combined under huber loss function. The single feature model's error is also presented as below.

6.6 Summary of Proxgrad

From all the previous analyses, we see that among all proxgrad methods (using multiple loss functions and regularizers), QuadLoss with ZeroReg and Quadloss with NonNegConstraint perform the best. We run the two models on our test set, and get the test errors to be 3.89 and 3.94. The error plots are shown as below.

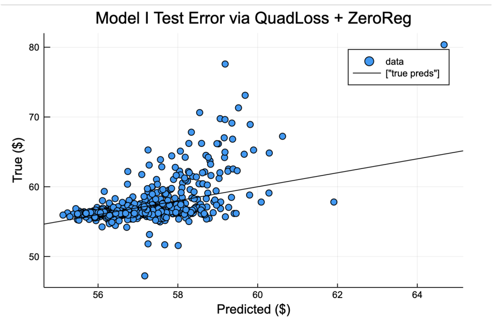


Figure 16: Quad loss with ZeroReg

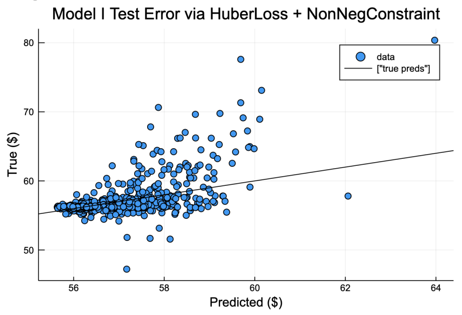


Figure 17: Quad loss with NonNegConstraint

Note that test errors of prox grad methods are slightly higher those got from linear regression, which has test error as low as 3.54. The possible reasons are:

- It takes long for proxgrad algorithm to converge, and even though we are taking 10,000 iterations, it still does not converge sometimes. So while linear regression makes prediction using the minimized w, proxgrad is making predictions using non-optimal w, resulting in higher errors.

- While linear regression is implicitly using mean squared error as the loss function, proxgrad algorithms are using lots of different loss functions. However, we are still comparing each model using MSEs. As the objective of proxgrads are not to minimize MSE, it is reasonable that it does not give as low MSE errors as liner models.

Given all the above analyses on proxgrad, we are not very confident with this method given its shortage compared to linear models as mentioned above. The predictions of proxgrad have higher MSE than those produced by linear models, so in real production, we are more willing to use linear models instead of proxgrad models.

7 Summary of Strategies

	train MSE	test MSE
Linear Model	3.523	3.541
Random Forest	2.779	3.756
Proximal Gradient	4.054	3.892

As we can see from the table above, linear model has the least test MSE. However, both linear model and proximal gradient descent has a very similar train and test error (proximal gradient train error greater than test error is probably due to the way we choose train data and test data randomly). Random forest method has the second lowest test error and its test error differs significantly from train error as expected.

The good point of using linear model is that it produces interpretable result and it has the best test error among the three methods. It also does not take a long time to run. However, we need to choose which features to include and see if we need to add polynomial features, which takes a lot of time and effort.

For proximal gradient, it has the highest test error but still satisfying result. The problem is that it needs a lot of time to run especially when we only get an acceptable result until 10000 iteration. We also need to see a combination of loss functions and regularizer to see which pair performs the best.

The merit of random forest lies in its simplicity. We only concatenate all the data (converted to many-hot/one-hot encodings) together and use it to produce a rather satisfying result. It also takes a very short amount of time the run.

8 Application in Production

Referring to the summary above, we are will to apply our model in production. We are especially recommending our Random Forest model, since this is the model that performs better, and that when applying this model, no specific attention needs to be payed.

Our model can be applied in two different aspects. From the aspect of audience/users, our model serves as a possible method for them to have an expectation of a movie, which might be helpful in their choice of movies. From the aspect of cinemas, our model might also help cinemas in deciding the time slots assigned to each movie. When assigning time slots to movies, cinema might want to give better time slots for a movie with higher predicted rating to gain higher revenue (for cinema).

9 Weapon of Math Destruction and Fairness

9.1 Weapon of Math Destruction

There are three questions we need to answer when considering Weapon of Math Destruction listed as below:

- Are outcomes hard to measure? **No**. Movie rating is a numerical value, so it is a quantifiable result. Also, since we are pointing to the IMDB rating, we can always check that if our prediction is correct or not. (In other words, we are just predicting a possible score for the movie on IMDB).
- Could its predictions harm anyone? **No**. Movie rating will not harm anyone. (The rating itself is unharm).
- Could it create a feedback loop? **Probably** The movie rating prediction might let popular movies more popular, and unpopular movies even less popular.

9.2 Fairness

Fairness is not an important criterion. Our dataset comes from IMDB website, and we analyze movie rating prediction based on that. One might say that IMDB consists mainly of US users, and might be biased. This is true, but on the other hand, only US users (or mostly) will refer to the website. This creates a closed cycle. (An realistic example would be, the IMDB dataset does not include much ratings from Chinese audience; yet,Chinese audience would usually not go to IMDB as reference for movie ratings).