Shawshank Regression: Predicting Movie Ratings

Benjamin Bush, Ben Choi, Andrew Cukierwar, William Tong

***Abstract*— In this paper, we compare the pros and cons of different learning models in the context of predicting movie ratings. The learning models we examined include: linear regressions, decision trees, Gaussian Processes, SVMs, and neural networks.**

I. INTRODUCTION

We selected the TMDB 5000 Movie Dataset for our application project. We were excited about the versatility of the types of analyses we could perform with this dataset and hoped to gain better insight to questions like:

- How can we provide the best possible movie recommendation based on a user's preferences?
- What makes a movie popular? The content of the movie itself or the people who create the movie (e.g. director and cast)?
- What types of models will work best for solving different problems?

The link to our dataset may be found on Kaggle [1] or on our repository [2]. Documentation for each milestone may be found on our repository's wiki [3].

II. PROBLEM FORMULATION

For most tasks, we decided to try to tackle the problem of the average rating of a given movie. Ratings are on a scale of 1 to 10, with 1 being the lowest rating and 10 being the highest. Since we would be predicting real-valued outputs, our project was a regression problem.

A. Error Measure

Because we were working in a regression setting, we chose squared loss as our loss function and mean squared error (MSE) as our error measure:

$$l(h_{\mathbf{w}}(\mathbf{x_i}), y_i) = (h_{\mathbf{w}}(\mathbf{x_i}) - y_i)^2$$

We know that the squared loss estimates the mean label at x_i and that this loss function is differentiable everywhere. Unfortunately, the squared loss function tries to accommodate every sample, so it is sensitive to outliers and noise.

III. IMPLEMENTATION

We implemented our project in Python and used the sklearn and pyGPs software packages. We also used Jupyter Notebooks for development because it made data exploration easy and led to fast prototyping.

We had many categorical features, such as director, a list of actors in each movie, etc. Because we were working in a regression context, we needed a way to extract these categorical features. For each of the milestones, we used a one-hot encoding for the features that we selected. As a result, we had a very sparse feature space since one hot encoding creates new binary columns, indicating the presence of each possible value from the original categorical feature space.

IV. PERFORMANCE EVALUATION

For each model, we evaluated the predictions using a 10-fold cross-validation using MSE as our error measure. Our regression problem predicts the average rating of movies on a scale of 1 to 10, so our validation errors will be on the same scale.

A. Linear Model

For our linear model, we selected the following features:

- One-hot encoding of top actors
- One-hot encoding of genres
- Vote count
- Popularity

Vote count represents the total number of votes cast for a movie and popularity is a metric of how many people marked a movie as a favorite or added a movie to their watchlist. We used the sklearn

Linear Regression Model to construct our linear model. The linear model achieved a validation error of 1.24. Considering our regression problem tries to predict the average vote score on a 10-point scale, we found this error measure to be reasonable. We did not add a regularizer into our linear model. Because the dataset was relatively small (about 5,000 samples), training our linear model did not take very long and we achieved an adequate level of accuracy. On the other hand, selecting a one-hot encoding for our categorical features expanded our feature space by $O(K)$, where K is the number of categorical features. Fortunately, the linear model has a relatively low VC-dimension, so this did not have too large of an impact on our model.

B. *Decision Tree*

For our decision tree, we selected the following features:

- One-hot encoding of top actors
- One-hot encoding of genres

We used sklearn's DecisionTreeRegression model with no bagging or boosting. Our decision tree regressor performed about as well as the linear model, earning a validation error of 1.25. Unfortunately, decision trees are especially prone to overfitting. If the feature space was expanded, the decision tree may have overfit the training set and may not have generalized well to unseen examples. It is encouraging, however, that we observed a relatively small loss from our 10-fold cross validation.

C. *Gaussian Process*

For our Gaussian Process (GP) model, we selected the following features:

- One-hot encoding of top actors
- One-hot encoding of genres
- Vote count
- Popularity

We set the GP's priors with means equal to the average movie rating of 6.09. We used three different kernels with varying covariances: RBF, Rational Quadratic, and Matern. For each kernel, in addition to finding the squared loss, we also found the negative log predictive densities (NLPD). The GP model produced a validation error of 2.02, which ended up being higher than any of our other learning models. The GP model most likely overfit, and this could be due to the hyperparameters of the kernel functions not being ideal. The GP had the lowest training error of all of the models (except the neural network), adding validity to this intuition.

D. *SVM*

For our SVM implementation, we selected the following features:

- One-hot encoding of top actors
- One-hot encoding of genres
- Vote count
- Popularity

We used sklearn's RBF SVM and cross-validation modules to analyze our data. For the SVM, we chose a gamma value of 0.1 and a C value based on a 3-fold cross-validation that yielded the lowest error. In scikit-learn, gamma defines how far the influence of a single training example reaches (low values meaning ”far”) and C trades off misclassification of training examples against simplicity of the decision surface (low C makes the decision surface ”smoother”). Our SVM implementation achieved a validation error of 0.87, which was the lowest validation error among all of our learning models (except the neural network). Despite its strong performance, however, we found that the SVM suffered from extremely long training times. As the SVM requires the use of a 3-fold cross-validation to determine the best C value, the implementation became much more time-intensive than other learning models like the linear regression model.

E. *Dimensionality Reduction*

The PCA/dimensionality reduction involved us looking at all of our features, constructing principle components, and then seeing if we can capture most of our information with only a few principal components. We started with the full set of features from the previous models, which included 34 of the following features:

- One-hot encoding of top actors
- One-hot encoding of genres
- Vote count
- Popularity

We used sklearn's Imputer and PCA packages to perform our dimensionality reduction. More specifically, we used the Imputer to get rid of null values and then ran a PCA with whitened data. We found that over 96% of the variance is explained by a single principal component. This is pretty major, as it means that we can reduce the dimensionality from 34 to 1 without significant information loss.

Using this information, we reconstructed the SVM and linear models using two principal components and compared the results to our previous validation error values without PCA. The linear model with PCA produced a validation error of 1.44 and the SVM with PCA produced a validation error of 1.54. Both models performed worse with PCA, but the dimensionality reduction made the models much quicker to train than using all 34 original features. Additionally, while the SVM with PCA increased validation error by 0.67, the linear model with PCA only increased validation error by 0.2. Although our problem is relatively small in its feature space (only 34), we can imagine that for an immense dataset, a PCA like this would be incredibly beneficial for producing relatively equitable results with much less computation and time.

F. Neural Network

For our neural network implementation, we selected the following features:

- One-hot encoding of top actors
- One-hot encoding of genres
- Vote count
- Popularity

For the neural network, especially compared to the other models, it was critically important to normalize/whiten the data in order for the neural net to give us a good model. We had to make sure to normalize in order to have the same range of values for each of the inputs, otherwise the network would have been ill-conditioned. We used sklearn and keras to implement our neural network. We used the keras API (built on the Tensorflow backend) to create a multi-layered, fully-connected neural network. For the data whitening, we used the sklearn preprocessing library. More specifically, we used the MinMaxScaler to perform transforms (and later for our predictions, inverse transforms). Our input layer had 34 nodes, 2 hidden layers with 34 nodes each, and an output layer with 1 node. For our model, we used a ReLU activation for all input and hidden layers and sigmoid for the output layer. For our optimizer, we chose adam. We fit our model using a 10-fold cross validation split with epochs=100 and batchsize=30.

Our model achieved a validation error of 0.0328. Although our regression problem normally tries to predict the average vote score on a 10-point scale, the neural network with normalized data tries to predict on a scale of 0-1. Even so, we found these error measures to be extremely encouraging. Obviously when we talk about neural networks, the largest barrier we most often face is computation time. Running the model fit for 100 epochs was not cheap in terms of timing and took significantly longer than the other models, even an expensive model like SVM.

V. Lessons Learned

Overall, we found the neural network and SVM models to produce the lowest validation errors. Because the SVM seemed to produce the best validation error without overfitting the data too much, we used a t-test to determine whether or not the SVM was statistically significant from the other models using a significance level of .05. For each individual t-test, our null hypothesis was that the SVM had the same error as the other model, while our alternative hypothesis was that the SVM had a lower error than the other model. Our results yielded p values of less than .05 for all models compared to the SVM, so we conclude with 95% confidence that the SVM has a lower error when compared to each individual model. Computationally, however, these two models were the most time-intensive to train. Although PCA/dimensionality reduction could speed up the training process, it leads to a noticeable increase in the validation error for some of our models. While it is tempting to discard the other learning models we implemented in favor of the neural network or SVM, other models like the linear regressor trades off accuracy for computational efficiency. The difference in time complexity means that under time-sensitive situations, it might be preferable to exchange worse accuracy for quicker performance.

This means that even though the neural network or SVM might have better performance, other models like linear regression or the decision tree still have practical applications and uses.

References

[1] https://www.kaggle.com/tmdb/tmdb-movie-metadata/data
[2] https://github.com/BenjaminBush/cse517a_project
[3] https://github.com/BenjaminBush/cse517a_project/wiki