[General Machine Learning] Predicting Movie Popularities Using Their Genomes

Jing Siang Ng (geraldjs) Ting An Ian Ngiaw (ianngiaw) Bili Xu (xbili)

Abstract

The 38 billion dollar movie industry has its successes and its flops. But which characteristic contributes to a movie's popularity? In this report, we investigate the correlation between movies' characteristics and their popularity using supervised learning algorithms.

1 Introduction

We used supervised learning algorithms to train models on genome tag scores to predict movie popularity scores.

The inputs to our algorithm are the genome tag scores. We then use regression models such as Ridge Regression, Support Vector Regression, Random Forest Regression, and Gradient Boosting Regression to output a predicted popularity score.

We also used classification models such as Decision Stump Boosting, SVM, Random Forests, and Gradient Boosting Classification to predict which category a movie belongs to i.e. popular or unpopular.

2 Dataset and Features

2.1 Movielens 20M Dataset

The genome tag score dataset was obtained from GroupLens[1], and is known as the MovieLens 20M dataset. A genome tag is a single characteristic exhibited by a movie (for example, atmospheric, thought-provoking, realistic, etc.). Each genome tag score reflects the relevance of a tag to a movie on a scale of 0 to 1. They were computed using a machine learning algorithm based on user-contributed content that includes tags, ratings and textual reviews.

The movie popularity scores were obtained from The Movie Database (TMDB) by accessing their open API. Each popularity score is a composite score calculated by TMDB based on unique page visitor views, number of ratings, number of times marked as favorites and number of watched list additions on the TMDB website.

The size of dataset we are performing our research on is 10,344 movies and we randomly split them up into 3 disjoint sets of ratio 3:1:1. Training set takes up 60% while cross-validation and test set takes up 20% each. For optimization of each algorithm, we made use of the training and cross-validation sets. While for comparison between algorithms, we used optimal values from learning on training and cross-validation sets before comparing using our test set.

[1]The MovieLens 20M Dataset
https://www.kaggle.com/grouplens/movielens-20m-dataset

Tags	Score
animal movie	0.54475
animation	0.98575
antartica	0.0375
apocalypse	0.1435
⋮	⋮

Table 1: Genome Tag relevance score for Toy Story (1995)

Movie	Score
Toy Story (1995)	3.220556
Jumanji (1995)	2.252717
⋮	⋮
Finding Nemo (2003)	4.666026
⋮	⋮

Table 2: Popularity score for movies

2.2 Preprocessing

We preprocessed the genome tag score data by centering it on the mean and normalizing the variance. i.e. $\mu = 0$ and $\sigma^2 = 1$. All methods were ran on these normalized data.

The process is as follows:

$$\mu = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}$$

$$\text{For each } i,\ x^{(i)} := x^{(i)} - \mu$$

$$\text{For each } j,\ \sigma_j^2 := \frac{1}{m}\sum_{i=1}^{m}(x_j^{(i)})^2$$

$$\text{For each } j,\ x_j^{(i)} := \frac{x_j^{(i)}}{\sigma_j}$$

Centering the mean ensures that attributes are treated on the same scale, while normalizing the variance rescales the different attributes to make them more comparable.

3 Methods

Each of the following methods was implemented using the SciKit Learn Python package (`sklearn`), with the exception of Jenks Natural Breaks Optimization which was not found in `sklearn`. We implemented a custom algorithm with `MATLAB` and saved the classified data as a `MATLAB` array, which is then loaded into Python.

3.1 Regression

Since we were predicting continuous values, a regression model would likely yield a good prediction. Here we applied regression models found in and out of the course syllabus.

3.1.1 Ridge Regression

We first trained a model using unregularized ridge regression where we minimized the cost function:

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - x^{(i)}\right)$$

$$\text{where } h_\theta(x^{(i)}) = \theta^T x^{(i)}$$

We then trained a model using l^2-norm regularized ridge regression where the cost function now consists of the regularization parameter α.

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - x^{(i)}\right) + \alpha\|\theta\|_2^2$$

$$\text{where } \alpha > 0$$

3.1.2 Kernelized Ridge Regression

After obtaining undesirable results (that will be shown in Section 3), we moved on to Kernelized Ridge Regression with radial-basis function and polynomial kernels of degrees 2 and 3.

$$\text{r.b.f. } K(x,z) = \exp(-\gamma\|x-z\|_2^2), \gamma \in \mathbb{R}$$

$$\text{poly. } K(x,z) = (x^T z + \gamma)^d,\ d \in \{2,3\}, \gamma \in \mathbb{R}$$

We chose to use Kernelized Ridge Regression as there may be correlation between features that cannot be expressed linearly.

3.1.3 Support Vector Regression

Finally, we decided to train a Support Vector Regression model using the same kernels specified in the previous subsection, minimizing the following cost function:

$$J_C(\beta) = C\sum_{i=1}^{m} L(K^{(i)T}\beta, x^{(i)}) + \frac{1}{2}\beta^T K\beta$$

$$\text{where } L(z,y) = \max(0, |y-z| - \varepsilon),\ \varepsilon > 0$$

$$\text{and where } K = \begin{bmatrix} K(x^{(1)},x^{(1)}) & \dots & K(x^{(1)},x^{(m)}) \\ K(x^{(2)},x^{(1)}) & \dots & K(x^{(2)},x^{(m)}) \\ \vdots & \ddots & \vdots \\ K(x^{(m)},x^{(1)}) & \dots & K(x^{(m)},x^{(m)}) \end{bmatrix}$$

3.1.4 Random Forests Regression

In Random Forests Regression, we first randomly select with replacement examples from the training data set and build a regression decision tree of a specified maximum depth.

This is repeated for a specified number of times. The predicted output is the average output value from each decision tree.

3.1.5 Gradient Boosting Regression

Similar to Decision Stump Boosting taught in class, the algorithm uses Decision Trees instead of Decision Stumps as weak learners.

We can then tune the number of decision trees used, and the maximum depth of each decision tree.

3.2 Classification

We then move on to experiment with several classification models.

3.2.1 Identifying Classes

Since movie popularity scores are continuous, we need to first identify different classes of popularity before we proceed with developing classification models. To do so, we

utilize a method known as Jenks Natural Breaks Optimization.

Algorithm 1 Jenks Natural Breaks Optimization

procedure JENKS(x)
  $SDAM \leftarrow \sum_{i=1}^{m}(x_i - \mu)^2$
  for $i \leftarrow 1 : m-1$ **do**
    $l \leftarrow \{x_1 \dots x_i\}$
    $r \leftarrow \{x_{i+1} \dots x_m\}$
    $\mu_l \leftarrow \frac{1}{i}\sum_{j=1}^{i}(x_j)$
    $\mu_r \leftarrow \frac{1}{m-i}\sum_{j=i+1}^{m} x_j$
    $SDCM \leftarrow \sum_{j=1}^{i}(x_j-\mu_l)^2 + \sum_{j=1}^{m}(x_j-\mu_r)^2$
    $y_i \leftarrow \frac{SDAM-SCDM}{SDAM}$
  end for
  $\hat{i} \leftarrow \arg\max_i y_i$
  return $\hat{i}$
end procedure

3.2.2 Decision Stump Boosting

The first classification algorithm we tried was Decision Stump Boosting as taught in class. Where a decision stump is defined as follows:

$$\phi_j, s(x) = sign(x_j - s) = \begin{cases} 1, & \text{if } x_j \geq s \\ -1, & \text{otherwise.} \end{cases}$$

3.2.3 Support Vector Classifier

Then, we decide to train a Support Vector Classifier model using the same kernels for regression, minimizing the following loss function (hinge loss):

$$J(\theta) = \begin{cases} 0, \text{ if } 1 - y^{(i)}\theta^T x^{(i)} < 0, \text{otherwise} \\ \frac{1}{m}\sum_{i=1}^{m} log(1 + exp(-y^{(i)}\theta^T x^{(i)})) \end{cases}$$

3.2.4 Random Forests Classification

This is similar to Random Forests Regression, but the leaves of each Decision Tree are discrete class labels, rather than a continuous value.

3.2.5 Gradient Boosting Classification

This is similar to Gradient Boosting Regression, but the leaves of each Decision Tree are discrete class labels, rather than a continuous value.

4 Results & Analysis

We first talk about methods taken to prevent overfitting. We used hold-out cross-validation throughout when running all our learning algorithms. We tune our model parameters on the cross-validation set and compare the performance of our algorithms on a test set.

Algorithm	Kernel	Error
Ridge Regression	-	0.8328
RR w/ Regularization	-	0.8006
Kernelized Ridge Regression	R.B.F.	0.2455
	Poly-2	0.2311
	Poly-3	0.2303
Support Vector Regression	R.B.F	0.2825
	Poly-2	0.2646
	Poly-3	0.2651
Random Forests Regression	-	0.3125
Gradient Boosting Regression	-	0.2513

Table 3: Regression algorithm performance, using mean squared error on test set.

4.1 Regression Results

From Table 3, it is evident that Ridge Regression (with and without regularization) are the worst performing as expected as it assumes little or no mutlicollinearity. Ridge Regression also assumes linear relationship between genome tag scores and popularity but features and outputs often do not have a linear relationship.

On the other hand, the best performing algorithm is Kernerlized Ridge Regression is polynomial kernel of degree 3.

In Figure 1, we show how the predictions of popularity vary with increasing true popularity. The green lines showing the actual popularity score from TMDB and the blue dots indicating the predicted probability based on genome tag scores.

For Gradient Boosting Regression, it turns out that larger number of trees (> 700 trees) had the lowest error. Whereas in Random Forests Regression, a relatively smaller number of trees yielded the optimal error.

4.2 Classification Results

Before we talk about the observations from classification algorithms, we will first discuss our results from Jenks Natu-

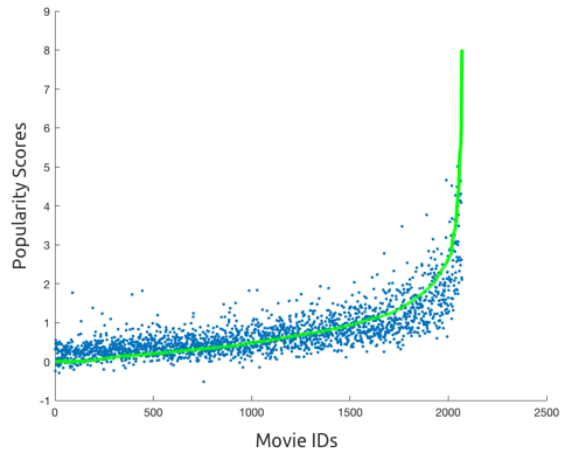

Figure 1: Kernelized Ridge Regression

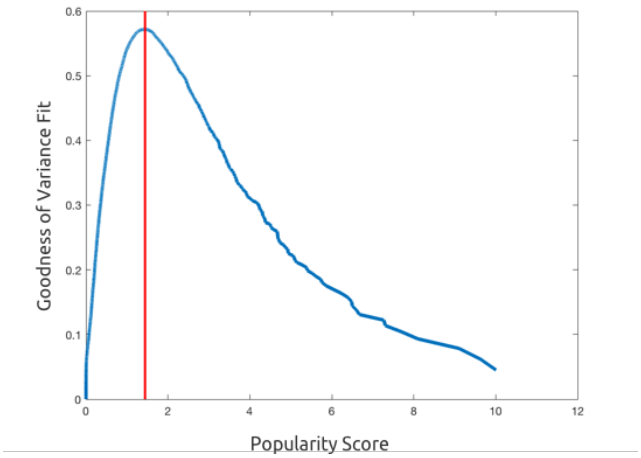

Figure 2: Jenks Natural Break Optimization

ral Breaks Optimization. It turns out that the optimal split point for two popularity classes is at the popularity score of 1.4405. This is illustrated in Figure 2 where we compared the Goodness of Variance Fit of using each popularity score in the training set as a split point.

To compare the different classification algorithms, we use F1 score which is the harmonic mean of precision and recall.

$$F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

It turns out that Gradient Boosting Classification has the best recall, and also has the best F_1 score, maintaining a good enough precision even with a higher recall. However, Support Vector Classification with R.B.F Kernel is able to out-perform Gradient Boosting Classification in terms of precision.

Algorithm	F1 Score	Precision	Recall
Decision Stump Boosting	0.5463	0.6571	0.4674
Support Vector Classifier with R.B.F Kernel	0.5152	0.8086	0.3781
Random Forests Classifier	0.4081	0.7216	0.2846
Gradient Boosting Classification	0.5797	0.7143	0.4878

Table 4: Classification algorithm performance, using F1 score.

		Predicted		
		Unpopular	Popular	Total
Actual	Unpopular	1775	48	1823
	Popular	126	120	246
	Total	1901	168	2069

Table 5: Confusion matrix for Gradient Boosting Trees Classification

Comparing Random Forests and Gradient Boosting Classification, the number of optimal number of trees is much lower than gradient boosting. 7 trees for Random Forests vs 400 for Gradient Boosting.

It was unexpected that Random Forests perform pretty badly in classification, while it performs almost as well as Support Vector Regression in regression.

5 Conclusion

The main purpose of this research is to determine a correlation between movie genome tags and popularity score. Through this research, we have achieved our goal of predicting movie scores with a good enough error rate.

It turns out that the best performing regression algorithm is Kernelized Ridge Regression with Poly-3 kernel, and the best performing classification algorithm is Gradient Boosting Classification. Both algorithm proves that there is little linear relationship between the genome tags and popularity score. It requires a combination of tags in order to successfully predict a movie's popularity.

6 Future Work

Given more time, we would use our research to implement a movie recommendation system that makes use of the genome tag scores, and other possible features in the dataset such as movie ratings, and the movie casts.

References

Harper, F., & Konstan, J. (2015). The MovieLens Datasets: History and Context. *ACM Transactions on Interactive Intelligent Systems (TiiS) 5*, 4, Article 19 (December 2015), 19 pages. DOI=http://dx.doi.org/10.1145/2827872

Smola, A., & Vapnik, V. (1997). Support vector regression machines. *Advances in neural information processing systems, 9*, 155-161. Chicago

A geospatial model of ambient sound pressure levels in the contiguous United States - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/278677730_fig2_Conceptual-diagram-of-the-RANDOM-FOREST-algorithm-On-the-left-trees-are-trained [accessed 14 Dec, 2016]

Jenks Natural Breaks Explained https://www.ehdp.com/methods/jenks-natural-breaks-1.htm

Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011. http://jmlr.csail.mit.edu/papers/v12/pedregosa11a.html