Bilibili Video Popularity Prediction Using Neural Network

Yufeng Wu sw20@williams.edu
Williams College Computer Science

1. Introduction

Bilibili is one of the most popular video platforms in China. Until Q3 2022, its average monthly active users has reached 332.6 million and is still predicted to rise further (CIW Team, 2022). With such huge amount of internet traffic, Bilibili is not only a site for entertainment, but it is also where many businesses market their products.

Therefore, it is crucial to be able to predict the popularity of videos. For those who uses videos for marketing, being able to accurately make such prediction means opportunities for better stock and shipment managements. For Bilibili, it can help them better allocate network resources by pre-loading predicted trendy videos to local data centers. For many full-time video creators, it means getting a sense of their next month's income beforehand.

The goal of this project is to use machine learning to predict a video's view on the 8th day of publish, using its metadata on the first day and information of the author. The video data are scraped directly from Bilibili using a web scraper that I built, and the final dataset contains 12,177 videos published between 8:33 to 9:33 A.M. EST on Oct.28, 2022. I tracked these videos only for 8 days due to the time constraints of this project, but the machine learning pipeline used in this project can be directly transferred to tasks that predict view counts longer or shorter than 8 days after publish.

I tried random forests and neural network, which achieved a validation score of $R^2 = 0.894$ and $R^2 = 0.939$ respectively. My final model is a neural network with two hidden layers of 352 and 224 nodes each, a dropout probability of 0.2, and a learning rate of 0.00558. It accomplished $R^2 = 0.742$ on the testing set. Although the model achieves moderately strong accuracy, it is not ready for deployment as there are many issues with distribution shift that the model currently does not consider.

2. Preliminaries

This project primarily uses random forest and neural networks, and they are compared with a baseline model trained using linear regression. Below are details about each model:

Linear regression produces predictions for an outcome variable Y as a linear combination of learned parameters θ and input features X. That is, for a given example, we have the following equation that determines how we obtain a prediction $\widehat{y}$ as a function of inputs $x_1, \ldots, x_d$,

$$\widehat{y} = \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_3 x_d.$$

The parameters θ can be learned by minimizing a suitable loss function L using gradient descent. This project uses the mean squared error as the loss function, which is defined for n samples of data as follows: $L(\theta) \equiv \frac{1}{n}\sum_{i=1}^{n}(y_i - \widehat{y}_i)^2$. Although it is common to use regularization to prevent overfitting, my baseline model uses an unregularized linear regression. The goodness-of-fit for linear regression can be measured by the R^2 metric, which is computed as follows:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y}_i)^2}$$

The value of R^2 usually lies between 0 and 1, although it can be negative when the model is worse than simply always predict $\bar{y}$, and higher R^2 value indicates better model.

This project uses the Sklearn package for linear regression.

Random forest (Ho, 1998; Breiman, 2001) is an ensemble learning algorithm that contains a large number of decision trees, each trained on a random subset of data using a random subset of features. A **decision tree** (Breiman et al., 2017) is trained by repeatedly splitting the training data into smaller subsets according to some user-specified criterion. The process continues until we can no longer minimize the loss by splitting the leaves further or until the tree reaches some maximum depth. Since we have a regression task, we will use the same mean squared error (MSE) used by linear regression as our splitting criterion. Specifically, for each node, the algorithm splits its data $[D]$ into two subsets, $[D]^{s1}$ and $[D]^{s2}$, that minimizes the weighted MSE, which is computed as follows:

$$\text{WeightedMSE}([D]^{s1}, [D]^{s2}) = \frac{\text{len}([D]^{s1}) \cdot \text{MSE}([D]^{s1}) + \text{len}([D]^{s2}) \cdot \text{MSE}([D]^{s2})}{\text{len}([D]^{s1}) + \text{len}([D]^{s2})}$$

where len() returns the number of rows in a certain dataset.

During prediction time, the algorithm sends the input to the trained decision tree from the root node to a leaf node. At each node, the algorithm uses the decision rule at that node to decide which child to visit next. When the algorithm reaches a leaf node, it outputs the predicted value associated with that leaf, which is the mean of the Y values for all data points in that leaf.

Given a dataset $[D]$ with n rows and d features, the random forest algorithm is trained as follows:

1. Sample m datasets $[D]_1$, ... , $[D]_m$, each of size n, from $[D]$ with replacement.

2. For each $[D]_i$, subsample $k \leq d$ features without replacement, and train a depth-h decision tree that only considers these k features.

During prediction time, the algorithm applies the input data to each decision tree and returns the mean of the outputs from m individual trees. Hyperparameters m, k, and h will be tuned using a validation set. Accuracy for this random forest is also measured by the R^2 metric.

This project uses RandomForestRegressor from sklearn for random forest training.

Finally, a **neural network** (Rosenblatt, 1958) can be used to predict Y using input data X_i and some learned parameters θ through some non-linear transformations. As explained in Rohit's slides, a neural network is a directed acyclic graph (DAG) partitioned into k layers, $\mathcal{L}_1$ through $\mathcal{L}_k$, where the first layer $\mathcal{L}_1$ contains one node per feature, and each node contains the value of each of these features. Nodes in subsequent layers $\mathcal{L}_i$ are determined by non-linear transformations of linear combination of nodes in $\mathcal{L}_{i-1}$. The exact form of linear combination is determined by the weight of edges pointing from $\mathcal{L}_{i-1}$ to $\mathcal{L}_i$, and the non-linear transformation is done using an activation function. This project uses ReLU as the activation function in order to ensure stability of the gradient descent process. The ReLU function is defined as follows:

$$f(x) = \max(0,\ x)$$

The values of the nodes in the final layer $\mathcal{L}_k$ is treated as the final predictions of the neural net for the given inputs in $\mathcal{L}_1$. For this project, there will be exactly 1 node in $\mathcal{L}_k$ representing the neural net's prediction of a video's views. To introduce an intercept term, we also add a node that just contains the value 1 to every layer except for the output layer $\mathcal{L}_k$.

The parameters of the DAG (i.e., weight of the edges) are learned using gradient descent to minimize a suitable loss function. We will again be using the MSE loss function here. To reduce computational time and produce models with better generalization error, this project uses mini-batch gradient descent (Hinton et al., 2012), which is a variant of gradient descent except that we update the parameters based on gradients calculated using a batch of w samples randomly selected from the training data, where w is a tunable parameter. To stablize the training process, we will use gradient clipping during gradient descent updates, which truncates the value of gradients to some maximum constant $G > 0$ whenever the gradient is greater than G.

Finally, we use dropout (Srivastava et al., 2014) to prevent overfitting. Dropout is a technique that randomly shut off some connections in the network with probability p at each iteration. Then, at testing time, we do not randomly shut off the connections. Instead, we weight the parameters in the network by $1 - p$. The accuracy of the neural net is also evaluated using the R^2 metric.

This project uses the Keras API to implement neural nets.

3. Data

The task of this project is to predict a video's views 8 days after its publish based on its 1st day statistics. The dataset that I used scraped directly from Bilibili using a web scraper that I built, because there are no publicly available dataset that records the growth of videos' statistics over time, from its moment of publish.

The dataset contains 12,177 videos published between 8:33 to 9:33 A.M. EST on Oct.28, 2022 and 47 raw features are collected from each video. Features that are not suitable for our and features with the exact same label for all videos in the dataset are discarded. Some features like text data are further processed to extract features that may be helpful to predict video popularity. Most categorical variables are converted into either one-hot encoding or indicator variable, and ordinal encodings are used when the categories have a natural ordering.

The final dataset contains the following 29 features for each video, and all the time-variant features (e.g. View count) are collected exactly 1 day after the video was published:

- Length of the video in seconds
- Is the video a short-form story (similar to Instagram Story)? (0 = No, 1 = Yes)
- Is the video original? (0 = No - it's reposted, 1 = Yes)
- Does the video prohibit reprint? (0 = No, 1 = Yes)
- Is the video quality HD? (0 = No, 1 = Yes)
- View count
- Virtual coins count
- Danmu count
- Favorite count
- Like count
- Share count
- Comment count
- Number of videos in this collection (1 if it's just a single video)
- Number of tags associated with the video
- Total number of videos published by this creator
- Creator's follower count
- Creator's like count
- Creator's following count
- Is the creator verified by Bilibili? (0 = No, 1 = Yes)
- Level of the creator (ordinal scale from 0 to 6)
- Is the video frame horizontal? (0 = No, 1 = Yes)
- Title length
- Title excitedness (count of '?' and '!' characters in the title)
- Title memeness[1] (count of white spaces in the title)

[1]Meme videos on Bilibili usually have white spaces between every character

- Average number of subscribers of all tags in this video
- Average number of videos achieved under all tags in this video
- Is the creator male? ($0 =$ No, $1 =$ Yes)
- Is the creator female? ($0 =$ No, $1 =$ Yes)
- Is the creator's sex hidden? ($0 =$ No, $1 =$ Yes)

The response variable is the video's view count 8 days after publish. I used a 60%-20%-20% split for my training, validation, and testing set. Next, I standardized the continuous variables in all three sets using the means and standard deviations of these variables from the training set. In this way, I prevent information leaking from testing and validation set into the training set.

4. Training And Validation Of Models

I began by setting up an unregularized linear regression model as our baseline. It achieved $R^2 = 0.867$ on training set and $R^2 = 0.757$ on validation set. The gap between training and validation score indicates a moderate degree of overfitting, but it is still a suitable baseline model as it achieved relatively strong scores on both sets.

Next, I built a random forest model and tuned hyperparameters on the validation set described in Section 3 using a grid search through all combinations of the following options:

- k (the number of features that each tree considers): all features or $\lceil \sqrt{29} \rceil = 6$ features
- m (the number of trees in random forest): 100, 480, 860, 1240, 1620, or 2000
- h (maximum depth of each tree): 10 to 100 with step size 10, and unlimited depth

I only tried two options for k because these two are the most common choices based on rule of thumbs. Other hyperparameters like the minimum number of samples required to split a node and the minimum number of samples required to be at a leaf node are held at their default values at 2 and 1, respectively, to make the grid search computationally feasible.

The validation scores of a total of $2 \times 6 \times 11 = 132$ random forests shown in Figure 1, where the curves are formed by connecting consecutive points with a straight line. The model with the highest validation score is the one with $k =$ all features, $m = 100$, $h = 30$. It produced a training score of $R^2 = 0.953$ and a validation score of $R^2 = 0.894$, which is 0.137 higher than the validation score of the baseline model.

The next model I tried is neural network. I first experimented several different network architectures, ranging from 2 to 4 hidden layers. It seems that a network with 2 hidden layers is enough for this task, and too many layers seems to over-complicate the model, making it both difficult to compute and regularize. So, I proceed by setting the architecture fixed for 2 hidden layers and tuning the model using a random search on the following hyperparameter distributions:

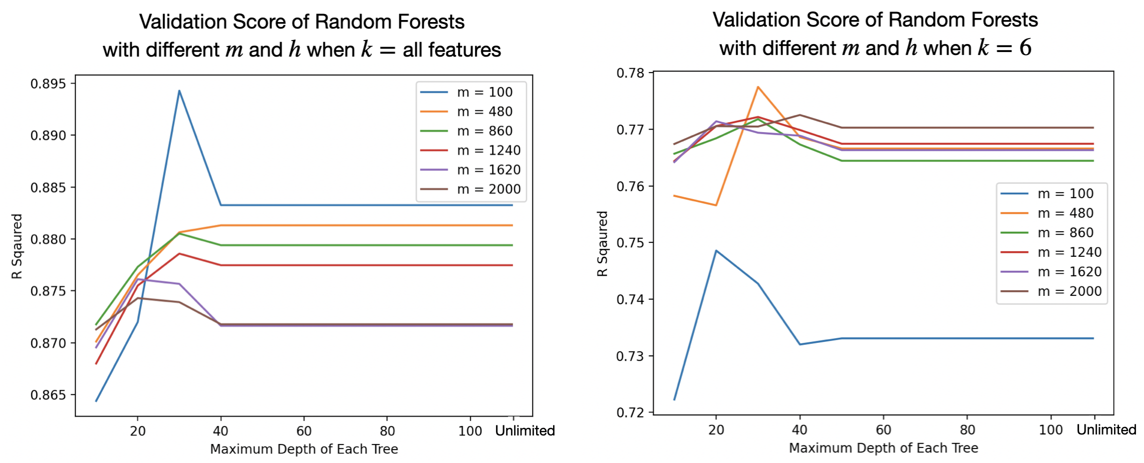

Figure 1: Validation scores of random forests with different hyperparameters

- dropout: equal probability for either TRUE or FALSE
- dropout probability: equal probability for either 0.1, 0.2, 0.3, 0.4, or 0.5
- first hidden layer size: equal probability for values between 32 and 512 (inclusive) with step size 32
- second hidden layer size: equal probability for values between 16 and 256 (inclusive) with step size 16
- learning rate: equal probability for values between 0.0001 and 0.01

I used the RandomSearch module in Keras Tuner and ran 100 samples from the above distribution. I selected the batch size to be 64 in order to stablize the training process while making the training process relatively fast, and I trained 1,000 epochs for all trials in order to allow models with small learning rates to have enough time to converge. The gradient clipping value G is set to be 0.5, as it well balances the speed and stability of the training process. The 100 models are ranked based on validation loss (MSE, as mentioned in Section 2), and Table 1 below summarizes the best five.

rank	dropout	dropout prob	1st hidden layer	2nd hidden layer	learning rate	validation loss
1	TRUE	0.2	352	224	0.00558	9914789.5
2	FALSE	n/a	416	160	0.00946	10588395.0
3	TRUE	0.2	256	80	0.00815	11182344.0
4	TRUE	0.2	320	160	0.00327	11437768.0
5	FALSE	n/a	352	80	0.00981	11582460.5

Table 1: Summary of the best 5 models from random search

The best model has a validation score of $R^2 = 0.939$. Figure 2 below shows the training and validation loss as well as R^2 for different epochs. As we can see, both training and validation R^2 increases very fast in the first 50 epochs and reaches the plateau. The model hits its best validation R^2 in epoch 337, and it slightly decreases after epoch 400 as training score slightly rises, indicating that the model starts to overfit to the training set. The zig-zagging shape in all curves reflect the variance of gradients between different batches, which is a reasonable shape since I am using mini-batch gradient descent.

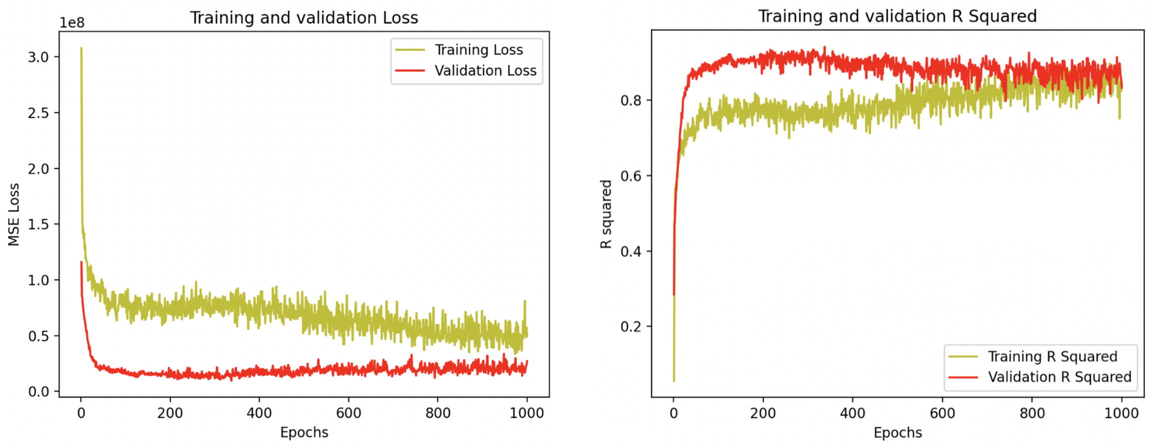

Figure 2: Training and validation loss & R^2 for the best neural network

This neural network will be our final model as it produces the highest validation R^2 with a reasonable computational complexity. To summarize, the neural net has two hidden layers with 352 and 224 nodes, and it is trained with batch size $= 64$, learning rate $= 0.00558$, and dropout probability $= 0.2$ for 337 epochs.

5. Results

The final model achieved a test score of $R^2 = 0.742$. Although it is significantly lower than its validation score, the model shows a moderately strong ability to predict the video's views on day 8. To evaluate the model in context, Table 2 shows a comparison between the predicted and actual views of the top 10 popular videos in the testing set. Among the actual top 10 videos, 8 of them are still predicted to be the top 10, but the order of the actual popularity is not correctly predicted by the model. The absolute error for these 10 predictions spread across several magnitudes, indicating relatively high variation of errors. The largest absolute error among the 2,435 data points in the testing set is $282,772.5$, where the actual view is $80,088$ but the predicted view is $362,860.5$.

ranking	view	predicted ranking	predicted view	absolute error
1	546117	2	457527.25	88589.75
2	404098	1	548891.87	144793.88
3	348092	5	192310.64	155781.36
4	190281	9	145615.79	44665.21
5	177556	10	144551.45	33004.55
6	162496	7	166619.57	4123.57
7	153591	6	167011.14	13420.14
8	126804	11	139134.42	12330.42
9	123520	8	158837.70	35317.70
10	94955	15	101796.35	6841.35

Table 2: Comparison between actual and predicted views for 10 most popular videos in testing set

6. Ablation Study

In order to gain more understanding of the inner working of the final model, I first examined the coefficients of the linear regression baseline model. Among the 29 input features, view count on day 1 is the single most important predictor. I would like to see whether my final model has the ability to predict day 8 view count without using day 1 view count. If the model turns out to remain moderately accurate without using this key input feature, then it may suggest that we can potentially build a model that predicts video popularity before it even publishes.

So, I dropped this feature in my dataset and trained my final model using this trimmed dataset. Figure 3 shows the training and validation curves using my final model, except that I trained it for 1,000 epochs instead of 337 (which is the setup in my final model) in order to see a more complete picture of the training process. As shown by the plots, the validation R^2 can only achieve around 0.67 at its best, which is 0.27 lower than it is on the original dataset. The testing R^2 is 0.382 on the epoch with the best validation score, which is 0.36 lower than before.

This experiment shows that view count on day 1 is indeed a very important factor that the final model heavily relies on, as its accuracy significantly declines after removing this feature.

7. Discussion and Conclusion

This project uses random forest and neural network to predict Bilibili videos' view count on day 8 using the author's information and the video's metadata on the 1st day of publish. Although my final model achieves $R^2 = 0.939$ on validation, it only scores $R^2 = 0.742$ during testing.

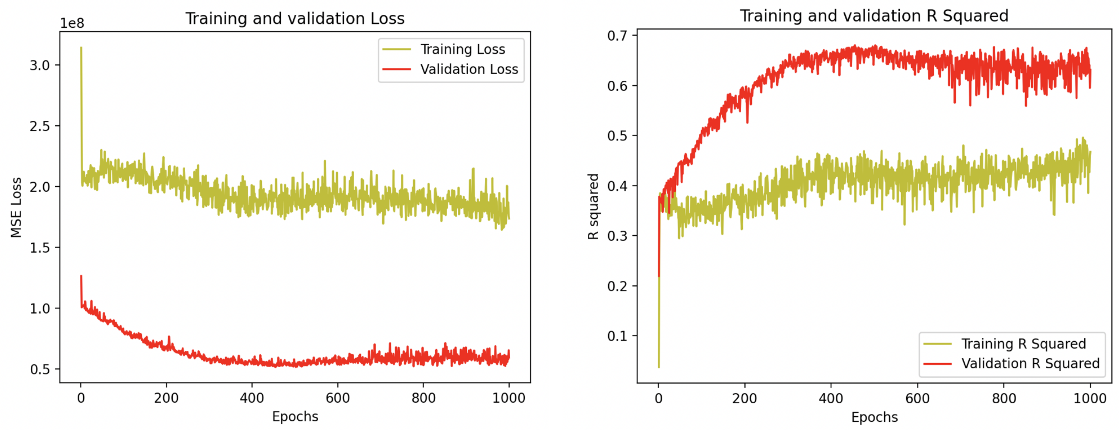

Figure 3: Training and validation curves after dropping ‘View count’

One possible reason for this gap is the possible distributional differences between the validation and testing set. In my dataset, the majority of videos are unpopular whereas a few videos have very high views. Specifically, 88.7% of the videos have less than 1,000 views on day 8, and 96.8% have less than 10,000. Thus, even though the train-validation-test split in my project is randomized, there could still be proportionally more videos with very high number of views in my validation set than testing set, or vise versa. This would result in my model overfitting to a specific outlier pattern in the validation set, resulting in a poor predictive power during testing.

In Ablation study, the model shows strong sensitivity towards the most important predictor: day 1 view count. Removing this feature from our data significantly decreased both the validation and test accuracy of the model.

Given these observations, I conclude that, although the final model achieves moderately strong testing score, it is still far from ready for deployment. In the future, possible directions to continue this topic include: first, collecting more data to potentially reduce the effect of extreme outliers; second, considering building several models that are specialized for different kinds of authors (e.g., one model for authors with less than 1,000 followers, another model for those with over 1,000); and finally, conducting in-depth ablation study, especially analyzing how the model responds to distribution shifts (e.g., different user behavior on major holidays, emergence of new pop culture, etc).

References

Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

Leo Breiman, Jerome H Friedman, Richard A Olshen, and Charles J Stone. *Classification and regression trees*. Routledge, 2017.

Unknown CIW Team. Chinese video platform bilibili mau up 25% in q3 2022. *China Internet Watch*, Nov 2022. URL `https://www.chinainternetwatch.com/31131/bilibili-quarterly/`.

Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. *Cited on*, 14(8):2, 2012.

Tin Kam Ho. The random subspace method for constructing decision forests. *IEEE transactions on pattern analysis and machine intelligence*, 20(8):832–844, 1998.

Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.