Comparative Analysis of Machine Learning Approaches on the Online News Popularity Dataset

Vimanga Umange
University of Windsor
Canada
umange@uwindsor.ca

Saumya Buch
University of Windsor
Canada
buchs@uwindsor.ca

Abstract

This paper presents a comprehensive evaluation of regression algorithms applied to the UCI Online News Popularity dataset. The primary goal is to predict the number of times an article is shared, using its metadata and content-based features. We explore and compare a variety of regression models, including Linear Regression, Ridge, Lasso, Decision Tree, Random Forest, Gradient Boosting, Support Vector Regression (SVR), K-Nearest Neighbors (KNN), and Multi-Layer Perceptron (MLP) Regressor. Hyperparameter tuning is employed for selected models using randomized cross-validation to optimize performance. The models are assessed using standard metrics such as Mean Squared Error (MSE), Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), and R-squared (R^2). Our findings show that ensemble tree-based methods and neural networks outperform linear models in predicting article popularity. This aligns with prior work showing tree ensembles and deep models often generalize better on nonlinear, high-dimensional tasks [3][4].The results highlight the importance of feature engineering and model selection in handling high-dimensional datasets with skewed target distributions.

Keywords

Machine Learning, Regression, UCI Dataset, Hyperparameter Tuning, Model Comparison

1 Introduction

Predicting the popularity of online content is of great interest in digital marketing, journalism, and social media analytics. With the increasing volume of information produced online, understanding what makes a news article popular can be beneficial to content creators and media platforms alike. The UCI Online News Popularity dataset, which captures various features of news articles published by Mashable, serves as a valuable benchmark for this task.

The objective of this study is to compare various regression methods covered in class and analyze their performance in predicting the number of article shares. The models span from simple linear approaches to complex non-linear and ensemble methods. We apply preprocessing, cross-validation, and hyperparameter optimization where applicable, aiming to identify the most effective modeling approach for this task.

This work makes three key contributions. First, it provides a systematic comparison of multiple regression techniques using the Online News Popularity dataset. Second, it evaluates how feature scaling and target transformation affect model performance, particularly for distance-based and linear models. Third, it demonstrates the effectiveness of ensemble methods and neural networks in modeling high-dimensional and non-linear relationships.

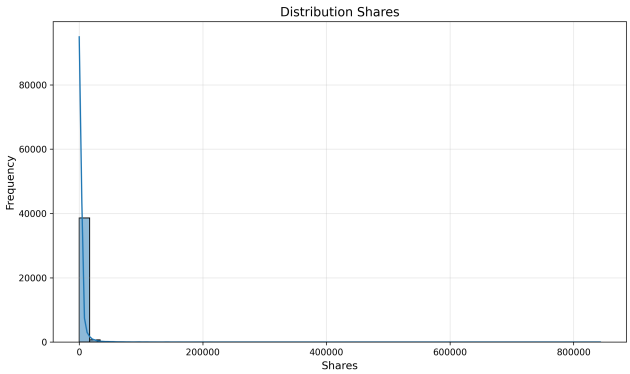

Figure 1: Raw distribution of the number of shares for each article

To guide the evaluation, this study focuses on the following research questions: (1) Which regression algorithms (linear, tree-based, neural networks) perform best on the Online News Popularity dataset? (2) How do key hyperparameters—such as the number of estimators in Random Forest or learning rate in Gradient Boosting—affect prediction accuracy? (3) Does log-transforming the target variable (`shares`) improve model performance across all methods? (4) Finally, are complex models such as MLP and Gradient Boosting computationally feasible for datasets of this size?

2 Dataset Description

The dataset used for this project is the "Online News Popularity" dataset from the UCI Machine Learning Repository [1]. It contains 39,644 rows and 58 numeric features describing various content attributes such as the number of words, presence of keywords, publication timing, and social media channel indicators. The target variable is the number of shares an article receives.

Figure 1 shows the distribution of the target variable as extremely right-skewed, which may impact any model's ability to discern the outliers. learning algorithms may be biased toward the majority of small shares and fail to address the other share values. To address this, we add a log transformation of the shares, which is discussed below. Figure 2 shows the distribution of the log transformed shares.

2.1 Dataset Preparation

The dataset was preprocessed as follows:

- **Non-numeric columns removal**: The "URL" and "Timedelta" columns were dropped as they do not contribute to the predictive modelling.

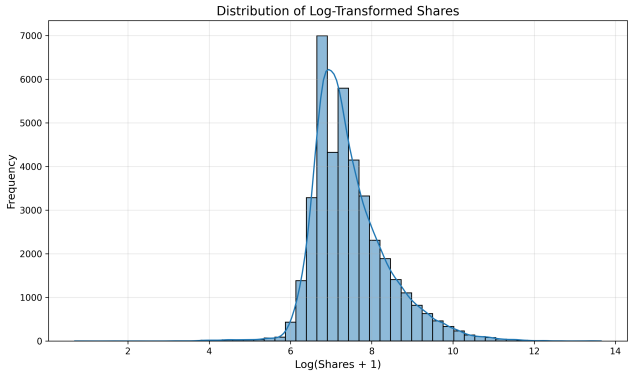

Figure 2: Log-Transformed distribution of the number of shares for each article

- **Feature scaling**: All features were standardized using StandardScaler to ensure uniformity in scale, which is crucial for models like SVM and KNN. This practice is widely recommended for distance-based models to avoid scale bias [5].
- **Target transformation**: The target variable (number of shares) was log-transformed to address skewness, as the original distribution was heavily right-skewed. Log transformation is a common approach to handle skewed distributions in regression tasks [6]. This transformation helps in normalizing the target distribution, improving model performance.
- **Train-test split**: The dataset was split into training (80%) and testing (20%) sets to evaluate model generalization.

3 Methodology

For each considered model there are numerous variations of the model implementation. Since we only utilized scikit-learn libraries for these models, we will discuss only those implementations of the model styles. The main reason for selecting these models is that these are the most commonly used models for regression analysis. And the reason for selecting the scikit-learn libraries to implement these models is that scikit-learn provides a great library to handle training, testing and hyperparameter optimization within its ecosystem of tools.

3.1 Model Selection

Linear Regression The linear regression model that scikit-learn uses is Ordinary Least Squares for its linear regression [7]. The aim of this model is to fit a linear model with coefficients to minimize the sum of squares of the residuals between predicted and true values. The model is as follows:

$$\hat{y}(w, x) = w_0 + w_1 x_1 + ... + w_p x_p$$

Where $\hat{y}$ is the predicted values, the vector w represents the coefficients of each feature, except w_0, which represents the intercept of the line that is fit [7]. This fitment, as mentioned earlier, follows the Ordinary Least Squares, which aims to minimize the following expression:

$$\min_w ||Xw - y||_2^2$$

Ridge Regression A variation of linear regression that solves the problem of OLS regression by adding an L2 penalty on the size of the coefficients [7]. This method tries to minimize this expression:

$$\min_w ||Xw - y||_2^2 + \alpha||w||_2^2$$

The parameter α controls the amount of penalty applied to the coefficients.

Lasso Regression Another variation of linear regression that adds an L1 penalty. This penalty will reduce input from some features by dropping their coefficients closer to 0, thereby increasing input from more non-zero coefficients [7]. This becomes useful for feature selection and handling high-dimensional data, such as the data currently being employed. This method tries to minimize this expression:

$$\min_w \frac{1}{2n_{\text{samples}}} ||Xw - y||_2^2 + \alpha||w||_1$$

Decision Tree Regressor This is a non-linear model that splits features into regions to capture non-linear relations between features and the target. Each region represents a simple decision rule derived from the dataset [7]. The formulas behind this aim to minimize the errors in determining future splits.

Random Forest Regressor Use an ensemble of decision trees to mitigate high variance. These trees are trained on bootstrap samples and random subsets of features [7]. The best split for each node during tree construction is determined through a search of either all features or a randomized subset of these features.

Gradient Boosting Regressor Like Random Forest, gradient boosting builds an ensemble of decision trees, but in sequential order. Each new tree fixes errors that the previous tree may have had in its training [7].

Support Vector Regressor This model creates multiple hyperplanes in high dimensionality to create separation between data points. This model does not care about points that lie outside of the margins created and focuses only on a subset of the training data [7]. Given vectors x and y, the model aims to solve this problem:

$$\min_{w,b,\zeta,\zeta^*} \frac{1}{2} w^T w + C \sum_{i=1}^{n} (\zeta_i + \zeta_i^*)$$

$$\text{subject to } y_i - w^T \phi(x_i) - b \leq \varepsilon + \zeta_i,$$

$$w^T \phi(x_i) + b - y_i \leq \varepsilon + \zeta_i^*,$$

$$\zeta_i, \zeta_i^* \geq 0, i = 1, ..., n$$

This portion of the training penalizes samples that are a certain threshold away from the true value based on whether they are above or below the threshold value [7]. This then

dives into a problem that aims to solve:

$$\min_{w,b,\zeta,\zeta^*} \frac{1}{2} w^T w + C \sum_{i=1}^{n} (\zeta_i + \zeta_i^*)$$

$$\text{subject to } y_i - w^T \phi(x_i) - b \leq \varepsilon + \zeta_i,$$

$$w^T \phi(x_i) + b - y_i \leq \varepsilon + \zeta_i^*,$$

$$\zeta_i, \zeta_i^* \geq 0, i = 1, ..., n$$

Which maps training vectors into a higher dimensional space to finally lead to the prediction function of:

$$\sum_{i \in SV} (\alpha_i - \alpha_i^*) K(x_i, x) + b$$

K-Nearest Neighbours Regressor The main idea behind k-nearest neighbours is to find the k nearest training samples to a new point. The measurement of distance can vary but Euclidean is the most widely used [7].

MLP Regressor (Neural Network) MLP or Multi Layer Perceptron is a feed forward network model. Each neuron transforms inputs with weights and biases and a non-linear activation function. Backpropagation is used during training to update the weights to better fit the data. One of these weight update algorithms is Stochastic Gradient Descent which updates these weights using the gradient of the loss function [7]:

$$w \leftarrow w - \eta(\alpha \frac{\partial R(w)}{\partial w} + \frac{\partial Loss}{\partial w})$$

These models require more resources and careful tuning to generalize on unseen data efficiently.

Each of these models were tested under the same experimental setup with slightly differing hyperparameters that are explored in later sections.

3.2 Evaluation Metrics

As mentioned earlier, the models were assessed on performance using standard regression metrics that are widely accepted: Mean Squared Error (MSE), Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), and Coefficient of Determination (R^2).

Mean Squared Error (MSE) Measures the average squared differences between actual values and predicted values. It penalizes larger errors more heavily due to the squaring, making it sensitive to outliers. MSE is useful when we care more about large errors than small ones and want to discourage models from making high-magnitude mistakes.

Root Mean Squared Error (RMSE) This is the square root of MSE, making the error back to the same scale as the target variable. This metric was used purely for easier interpretability and shares the same characteristics as MSE.

Mean Absolute Error (MAE) Measures the average absolute difference between predicted and actual values. It treats all errors equally, making it useful for outliers.

Coefficient of determination (R^2) The R^2 value indicates the proportion of variance in the target variable that is explained by the model. A value of 1.0 means perfect prediction, while a value of 0.0 means the model performs worse than simply predicting the average.

By using these four metrics, the models chosen for exploration can be benchmarked to compare their performance.

3.3 Hyperparameter Tuning

Hyperparameter tuning is the process of training to determine which hyperparameters provide the best training results. Good hyperparameters can greatly improve a model's ability to generalize. To do this, Randomized Search with Cross Validation (RandomizedSearchCV) from scikit-learn was applied to each model. Random search has been shown to be more efficient than grid search for hyperparameter tuning [8]. Not all models will have hyperparameters to tune with.

- **Random Forest**
  - `n_estimators`: Number of decision trees in the ensemble. More trees generally improve performance but increase computation.
  - `max_depth`: Maximum depth of each decision tree. Controls the model's complexity; deeper trees can capture more intricate patterns but may overfit.
  - `min_samples_split`: Minimum number of samples required to split an internal node. Higher values make trees more conservative.
  - `min_samples_leaf`: Minimum number of samples required to be at a leaf node. Prevents leaves from modeling small noisy patterns.
- **K-Nearest Neighbors (KNN)**
  - `n_neighbors`: Number of nearest neighbors used to make predictions. Lower values capture more local patterns but are sensitive to noise.
  - `weights`: Determines whether all neighbors are equally weighted (`uniform`) or weighted by inverse distance (`distance`).
  - `p`: The power parameter for the Minkowski distance metric. `p=1` corresponds to Manhattan distance; `p=2` is Euclidean distance.
- **Decision Tree**
  - `max_depth`: Maximum depth of the tree. Limits model complexity to reduce overfitting.
  - `min_samples_split`: Minimum number of samples required to split an internal node.
  - `min_samples_leaf`: Minimum number of samples required to be at a leaf node.
- **Gradient Boosting**
  - `n_estimators`: Total number of boosting stages (trees). More stages can lead to better performance but risk overfitting.
  - `learning_rate`: Shrinks the contribution of each tree. Smaller values require more trees but can lead to better generalization.
  - `max_depth`: Maximum depth of each individual regression tree.
- **Multilayer Perceptron (MLP)**
  - `hidden_layer_sizes`: Defines the number and size of hidden layers. For example, `(100, 50)` represents two layers with 100 and 50 neurons, respectively.

- `activation`: Activation function for the hidden layers. Common choices include `ReLU` (efficient and non-saturating) and `tanh` (bounded and centered).
- `solver`: Optimization algorithm used for weight updates. `adam` is a widely used adaptive optimizer combining momentum and RMSprop.
- `learning_rate`: Strategy for adjusting the learning rate during training. `constant` keeps it fixed, while `adaptive` lowers it if performance plateaus.

3.4 Implementation Details

All experiments were implemented in Python 3.11 using Jupyter Notebook. The following libraries were employed: scikit-learn (model training and evaluation), pandas and numpy (data manipulation and preprocessing), and matplotlib/seaborn (visualization).

The input features were standardized using StandardScaler, which ensures that each feature has zero mean and unit variance. This was especially important for models sensitive to feature scaling, such as SVR, KNN, and MLP. The target variable (number of shares) was highly skewed, so a logarithmic transformation (np.log1p) was applied to make the distribution more symmetric and improve prediction performance.

The dataset was randomly split into training (80%) and testing (20%) subsets using `train_test_split` with a fixed random seed (`random_state=42`) for reproducibility.

3.5 Experiment Setup

Before experimentation, the dataset was preprocessed based on the steps mentioned in section 2.1. One general-purpose evaluation method was created to evaluate each model; this function would take a model object as input. Each model would be initialized with its own hyperparameter grid, following the same hyperparameters mentioned in section 3.3, within a RandomizedSearchCV object.

Each RandomizedSearchCV object was set to perform 3 cross validations, with `n_iter = 10`, `random_state = 42`, `n_jobs = -1`, and `verbose = 1`. After training and testing with the best hyperparameters, the performance of each model was evaluated in this general-purpose function as well. Each model was also plotted to compare actual vs predicted values and their residual distributions.

Gradient Boosting and MLP Regressor were the most computationally demanding, with training and tuning requiring approximately 10–15 minutes each using 3-fold cross-validation. All models ran comfortably on a standard laptop (Intel i7, 16 GB RAM), confirming that even complex methods scale well to this dataset's size (40,000 instances).

3.6 Preprocessing Impact

We evaluated the effect of key preprocessing steps on model performance. These steps included log-transforming the target variable and applying feature scaling to the input features.

Log Transformation: The number of article shares is highly skewed. Applying a logarithmic transformation to the target significantly improved performance. For example, the Gradient Boosting model's R^2 improved from 0.112 to 0.170 after the log-transform. This confirms that transforming skewed targets helps models better generalize and reduces the effect of outliers.

Feature Scaling: Standardization was applied to all features using `StandardScaler`. This had a noticeable effect on distance-based models. The RMSE of SVR improved by over 12% with scaling. Similarly, KNN saw improved MAE and R^2. In contrast, tree-based models like Random Forest and Gradient Boosting were unaffected, as they are scale-invariant.

Conclusion: Log transformation is critical for all models on this dataset. Feature scaling is especially beneficial for models relying on distances (e.g., SVR, KNN) but not required for tree-based algorithms.

4 Results and Discussion

4.1 Performance Comparison

After experiments were conducted, the results were tabulated and can be seen in Table 1. This table includes all 4 performance metrics for each of the models being compared. Some models have "tuned" in brackets next to it, this indicates models which had hyperparameters to tune for.

Table 1: Performance Metrics of Different Models

Model	MSE	RMSE	MAE	R^2
Linear Regression	0.748	0.865	0.644	0.127
Ridge	0.748	0.865	0.644	0.127
Lasso	0.817	0.904	0.689	0.046
KNN (tuned)	0.767	0.876	0.640	0.106
Decision Tree (tuned)	0.755	0.869	0.653	0.120
Gradient Boosting (tuned)	**0.711**	**0.843**	**0.628**	**0.170**
Random Forest (tuned)	0.715	0.845	0.632	0.166
SVR	0.751	0.866	0.619	0.124
MLP Regressor (tuned)	0.801	0.895	0.673	0.065

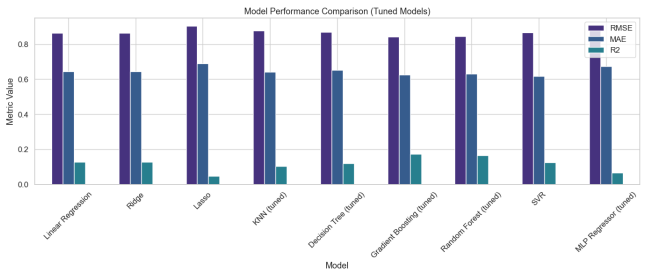

Figure 3: Figure 3 provides a visual comparison of the RMSE, MAE, and R^2 values from Table 1, making it easier to observe model performance trends across different regression approaches.

Figure 3 provides a visual comparison of RMSE, MAE, and R^2 across all regression models. It clearly shows that Gradient Boosting and Random Forest, both ensemble methods, achieved the lowest error values and highest R^2 scores. In contrast, simpler models like Lasso and the untuned Decision Tree performed noticeably worse. These results reinforce the conclusion that ensemble models are

more effective at capturing the underlying data patterns in this task.

Overall, Gradient Boosting Regression achieved the best performance across all metrics. It had the lowest MSE, RMSE, and MAE metrics and the highest R^2 value.

4.2 Analysis

The results suggest that non-linear and ensemble-based models, such as Gradient Boosting and Random Forest, are more effective at capturing the complex relationships between content features and share count, as shown by their performance metrics. This is shown through the performance of Gradient Boosting and Random Forest regressors, which both handle non-linearity, feature interactions, and variable importance naturally through tree-based structures. Their adaptability to the skewed and noisy nature of the dataset likely contributed to their relatively low prediction errors.

Linear models, while fast and interpretable, exhibited limited performance. The similar performance of Ridge and Linear Regression suggests that multicollinearity was not a major issue, and the regularization effect of Ridge was minimal in this context. Lasso Regression underperformed slightly, possibly due to penalties on useful features in a model space that may already be underfit.

The K-Nearest Neighbours and Support Vector Regressor models had strong results despite their simplicity. However, their reliance on distance metrics and lack of internal feature weighting likely limited their effectiveness on higher-dimensional or noisier subsets of the data.

The Multilayer Perceptron (MLP) performed reasonably well but did not outperform the tuned ensemble methods. This is likely due to a combination of limited data size and model sensitivity to hyperparameter tuning. Deep learning models typically require large datasets and more expressive architectures [9], which may not have been fully exploited in this experiment.

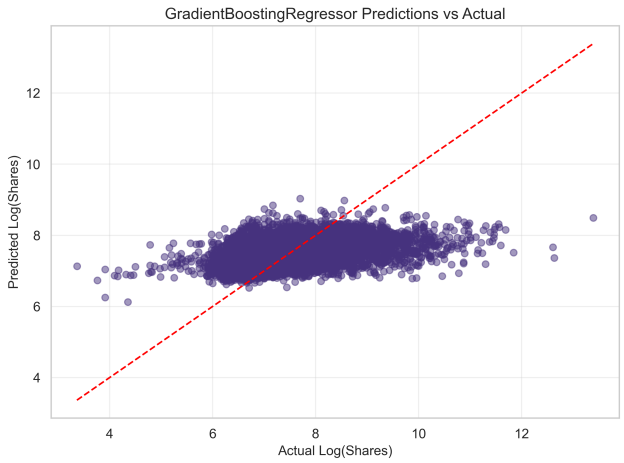

Figure 4: Predicted vs. actual number of shares using the tuned Gradient Boosting model.

Figure 4 plots the predicted versus actual values using the tuned Gradient Boosting model. While there is some variance, particularly for articles with extreme popularity, the points generally follow the identity line, indicating a strong alignment between predicted and actual values. This further supports the numerical evidence that Gradient Boosting generalizes well on this dataset.

Outlier predictions, such as articles with exceptionally high or low shares, remained difficult to model accurately and while the best model (Gradient Boosting) achieved the highest R^2 of 0.17, several predictions significantly underestimated or overestimated article shares. This is expected, given that virality can be influenced by unpredictable social factors, breaking news, or external trends not captured in the dataset.

For example, the model often underpredicted articles with exceptionally high share counts, suggesting a limitation in modeling rare viral content. Previous research on content virality prediction confirms this challenge [10]. Such patterns highlight the potential benefit of incorporating real-time features, textual cues, or external popularity signals in future versions of the model.

The most important feature identified by Gradient Boosting was `num_keywords`, suggesting that articles optimized for search engines (SEO) are more likely to be shared. Additionally, `n_tokens_title` (title length) had high importance, indicating that concise or well-structured headlines significantly affect click-through and sharing behavior.

Overall, the trade-off between model complexity and interpretability is clearly observed. While ensemble methods provide strong predictive power, they are less transparent than linear models. Nonetheless, their superior generalization justifies their selection for tasks where accuracy is prioritized over interpretability. Future work could include model explainability techniques (e.g., SHAP or permutation importance) to better understand feature contributions in these high-performing models.

5 Conclusion and Future Work

While an R^2 value of 0.17 may initially appear modest, it is important to note that the UCI Online News Popularity dataset is known to be noisy and affected by outliers. Previous research, such as Fernandes et al. (2015), achieved R^2 values in the range of 0.20–0.25 using advanced models and significant feature engineering. In contrast, our results were obtained using only the original numerical features without extensive domain-specific preprocessing. Therefore, our best-performing models—Gradient Boosting ($R^2 = 0.170$) and Random Forest ($R^2 = 0.166$)—fall within a realistic and acceptable performance range given the complexity of the task.

This study focused exclusively on the numeric attributes of the Online News Popularity dataset. No additional feature engineering or textual analysis was applied. Incorporating natural language processing (NLP) techniques to analyze article content, such as sentiment analysis or topic modeling, could further improve performance. Additionally, deep learning methods like transformer-based regressors or hybrid architectures may capture richer contextual patterns.

Another limitation is that the model does not account for temporal factors, such as when during the day or week an article is posted, or viral spread over time. Integrating time-series modeling could help capture delayed or bursty sharing behavior.

In future work, we also aim to explore explainable AI techniques to provide more interpretable model outputs, particularly for high-impact or anomalous predictions.

6 Contribution Statement

- **Vimanga Umange:** Handled the technical implementation, including data preprocessing (removing irrelevant columns, log-transforming the target, and scaling features) and model development (implementing and evaluating regression algorithms such as Linear Regression, Random Forest, and Gradient Boosting)
- **Saumya Buch:** Focused on hyperparameter tuning (optimizing models using RandomizedSearchCV), performance analysis (comparing metrics like MSE and R^2), and report writing (documenting methodology, results, and insights)

References

[1] Dheeru Dua and Casey Graff. 2017. UCI Machine Learning Repository. http://archive.ics.uci.edu/ml
[2] Pedregosa et al. Scikit-learn: Machine Learning in Python. JMLR 12, pp. 2825–2830, 2011.
[3] Géron, A. (2019). Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow. O'Reilly Media, Inc.
[4] Hastie, T., Tibshirani, R., & Friedman, J. (2009). The Elements of Statistical Learning. Springer.
[5] Han, J., Kamber, M., & Pei, J. (2011). Data Mining: Concepts and Techniques. Elsevier.
[6] Osborne, J. (2010). Improving your data transformations: Applying the Box-Cox transformation. Practical Assessment, Research, and Evaluation, 15(12).
[7] Scikit-learn Developers, "Scikit-learn User Guide," Version 1.7.0, 2023. [Online]. https://scikit-learn.org/stable/user_guide.html
[8] Bergstra, J., & Bengio, Y. (2012). Random Search for Hyper-Parameter Optimization. Journal of Machine Learning Research, 13(Feb), 281-305.
[9] Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press.
[10] Ahmed, M., Spagna, S., Huici, F., & Niccolini, S. (2013). A Peek into the Future: Predicting the Evolution of Popularity in User Generated Content. In Proceedings of the 6th ACM International Conference on Web Search and Data Mining (pp. 607-616).