CSCI 5502-872

Group Project Report

On

Milestone 3 – Model Implementation

By:

Group Name: “Group 7”

Adwait Girish Mahajan: adma4717

Junsoo Jung: juju6944

Shreyash Ajay Sahare: shsa7246

**Introduction:**

In numerous fields, predictive modeling is a fundamental tool that provides predictions and insights based on the complex interactions between data. Using data from other columns to estimate values in a particular column is a basic component of predictive modeling, and it's a procedure that's essential for comprehending trends, coming to wise conclusions, and finding hidden patterns in datasets. Within the gaming industry, game ownership prediction is a crucial undertaking that provides insight into the level of popularity and appeal of various game titles, studios, and genres. Regression models of varying strengths have been used to tackle this job, with the goal of capturing the complex correlations between different characteristics and the target variable, game ownership. These models are chosen according to how well they can handle the complexity of the game dataset and produce precise predictions, ranging from the basic simplicity of linear regression to the intricate workings of neural networks.

Within this framework, we investigate the modeling strategy used to forecast game ownership. We seek to identify the complex web of factors impacting game ownership through the lens of regression approaches, including neural networks, k-nearest neighbors (KNN) regressor, random forest regressor, linear regression, and support vector regression (SVR). We want to leverage the combined capability of a wide range of regression models to find significant patterns and insights in the gaming industry.

This introduction acts as a starting point for our investigation, providing an overview of the modeling methods available to us, the reasoning behind their selection, and the next steps in putting these models into practice in order to use them to forecast game ownership based on the data in the dataset. We hope to solve the puzzles around game ownership prediction along the way and facilitate well-informed decision-making in the gaming sector.

**Modeling Approach:**

Regression models were utilized in order to forecast continuous numerical values, which is necessary for calculating the potential number of owners of a game based on different variables. We specifically used the following five regression models: neural networks, k-nearest neighbors (KNN) regressor, random forest regressor, support vector regression (SVR), and linear regression.

1. *Linear Regression Model*:

This traditional approach was selected because of its clarity and interpretability. Assuming a linear relationship between the features and the objective variable, linear regression is a useful tool for preliminary data exploration and interpretation.

2. *Support Vector Regression (SVR) Model*:

The flexibility of SVR in managing non-linear connections between features and the target variable led to its selection. SVR can identify intricate patterns in the data by mapping it into a higher-dimensional space using a kernel function.

3. *KNN Regressor Model*:

The KNN regressor was selected because to its capacity to identify certain patterns within the data. This model works well with datasets that include localized patterns or clusters because it computes the target variable by averaging the values of its k nearest neighbors.

4. *Random Forest Regressor Model*:

The random forest regressor was employed due to its robustness and capacity to manage big, highly dimensional datasets. This ensemble model lessens overfitting and produces precise predictions by combining the predictions of several decision trees.

5. *Neural Networks*:

Neural networks were utilized for their ability to capture complex and non-linear relationships in the data. With multiple hidden layers, neural networks can learn intricate patterns and interactions between features, making them suitable for modeling the complexities inherent in predicting game ownership.

The last preparation procedures were carried out prior to the models' implementation to make sure the dataset is clear, organized, and prepared for modeling. This laid the framework for precise forecasts and perceptive analysis of changes in video game ownership. The actions listed below are as follows:

1. Removing Missing Values:

The `dropna()` function is used to remove any rows in the dataset that have missing values. This guarantees that for modeling purposes, only complete observations are kept.

2. Dropping Columns with High Cardinality:

In order to make room for more relevant columns for modeling, columns like "Name," "Developers," "Publishers," and "Category," which have a large number of unique values, are removed.

3. Cleaning Data:

The "Followers" column is cleaned to guarantee correctness and consistency in displaying the number of followers for every game. Commas are removed, and the datatype is changed to integer (int64).

4. Extracting Month from Release Date:

After converting the "Release\_Date" column to datetime format, the month component is taken out and put into a new column called "Month." The analysis of release trends over many months is made possible by this change.

5. Encoding Categorical Variables:

Label Encoding converts categorical variables, such "Genre," into a numerical representation. This procedure makes it easier to incorporate categorical data into machine learning models by giving each category a unique number.

6. Renaming Columns:

The column "Ownership\_Midpoint" is finally renamed to "Num\_Owners" in order to provide the goal variable a more illustrative term.

**Model Implementation:**

Following the modeling process, the models were successfully fit to the transformed data and executed accordingly.

1. *Linear Regression Model*:

* Feature Selection:

The dataset is divided into features (X) and the target variable (y). The features are obtained by dropping the "Num\_Owners" column, which serves as the target variable to predict.

* Train-Test Split:

The dataset is split into training and testing sets using the `train\_test\_split()` function from scikit-learn. The training set (X\_train, y\_train) comprises 80% of the data, while the testing set (X\_test, y\_test) contains the remaining 20%. This ensures that the model's performance can be evaluated on unseen data.

* Feature Scaling:

Min-max scaling is applied to the features using the `MinMaxScaler()` to normalize the feature values between 0 and 1. This preprocessing step ensures that all features contribute equally to the model's learning process and prevents features with larger scales from dominating the model.

* Model Initialization and Training:

A linear regression model is initialized using the `LinearRegression()` class from scikit-learn. The model is trained on the training data (X\_train, y\_train) using the `fit()` method, which adjusts the model parameters to minimize the residual sum of squares between the observed and predicted values.

A screenshot of a computer program

Description automatically generated

Fig. 1: Fitting the linear regression model

2. *Support Vector Regressor (SVR) Model:*

* Train-Test Split:

The dataset is split into training and testing sets using the `train\_test\_split()` function from scikit-learn. The training set (X\_train, y\_train) comprises 80% of the data, while the testing set (X\_test, y\_test) contains the remaining 20%. This allows for the evaluation of the model's performance on unseen data.

* Feature Scaling:

Standardization is applied to the features using the `StandardScaler()` from scikit-learn. This scaling technique transforms the features to have a mean of 0 and a standard deviation of 1, ensuring that all features contribute equally to the model and preventing features with larger scales from dominating the learning process.

* Model Initialization and Training:

An SVR model is initialized with default hyperparameters using the `SVR()` class from scikit-learn. The model is then trained on the training data (X\_train, y\_train) using the `fit()` method, which adjusts the model parameters to minimize the error between the observed and predicted values.

A screenshot of a computer program

Description automatically generated

Fig. 2: Fitting the SVR model

3. *K-Nearest Neighbors (KNN) Regressor Model:*

* Feature Selection and Preprocessing:

The dataset is split into features (X) and the target variable (y), similar to the previous models. The features are standardized using the `StandardScaler()` from scikit-learn to ensure that all features have a mean of 0 and a variance of 1. This preprocessing step is essential for KNN models as they rely on the distance between data points, and scaling ensures that all features contribute equally to the distance computation.

* Train-Test Split:

The dataset is split into training and testing sets using the `train\_test\_split()` function from scikit-learn. The training set (X\_train, y\_train) comprises 80% of the data, while the testing set (X\_test, y\_test) contains the remaining 20%. This allows for the evaluation of the model's performance on unseen data.

* Model Initialization and Training:

A KNeighborsRegressor model is initialized with the desired number of neighbors (n\_neighbors=5) using the `KNeighborsRegressor()` class from scikit-learn. The model is then trained on the training data (X\_train, y\_train) using the `fit()` method, which stores the training data in memory for later use during prediction and the choice of the number of neighbors (n\_neighbors=5) allows for flexibility in capturing the underlying patterns in the data.

A screenshot of a computer program

Description automatically generated

Fig. 3: Fitting the KNN Regressor model

4. *Random Forest Regressor Model:*

* Feature Selection and Preprocessing:

The dataset is split into features (X) and the target variable (y), similar to the previous models. Additionally, feature scaling is applied using the `StandardScaler()` from scikit-learn to ensure that all features have a mean of 0 and a variance of 1. This preprocessing step is included in the pipeline to standardize the features before model training.

* Pipeline Definition:

A pipeline is defined to encapsulate both feature scaling and the random forest regressor model. The pipeline consists of two steps: scaling the features and fitting the random forest regressor.

* Hyperparameter Grid:

A parameter grid is defined to search for the best combination of hyperparameters for the random forest regressor. This grid includes various parameters such as the number of estimators, maximum depth of the trees, minimum samples split, minimum samples leaf, and maximum features to consider when looking for the best split.

* Hyperparameter Tuning:

GridSearchCV is employed to perform hyperparameter tuning using cross-validation. The `GridSearchCV()` function from scikit-learn is utilized, which exhaustively searches through the parameter grid to find the combination of hyperparameters that yields the best performance according to the specified scoring metric, in this case, negative mean squared error. Hyperparameter tuning ensures that the model's performance is optimized, and the pipeline structure facilitates efficient preprocessing and model training.

* Best Model Selection:

The best model obtained from the grid search is extracted using the `best\_estimator\_` attribute of the grid search object. This model represents the random forest regressor with the optimal hyperparameters found during the grid search process.

A screen shot of a computer program

Description automatically generated

Fig. 4: Fitting the Random Forest Regressor Model

5. *Artificial Neural Networks (ANN) Model:*

* Model Architecture:

An ANN model is constructed using the Sequential API from Keras. The model comprises multiple layers, including dense (fully connected) layers and dropout layers for regularization. The architecture consists of three dense layers: the first layer with 64 neurons and ReLU activation, the second layer with 32 neurons and ReLU activation, and the output layer with a single neuron and linear activation, which is suitable for regression tasks.

* Model Compilation:

The model is compiled using the `compile()` method, where the loss function is specified as MeanSquaredLogarithmicError (MSLE) from TensorFlow. The optimizer is set to 'adam', which is an efficient optimization algorithm commonly used in neural network training. Additionally, MSLE is also used as a metric for monitoring model performance during training.

* Early Stopping:

Early stopping is implemented using the `EarlyStopping()` callback from Keras. This callback monitors the validation loss during training and stops the training process if the validation loss stops decreasing for a specified number of epochs (patience). This prevents overfitting and improves generalization performance.

* Training and Validation:

The model is trained on the training set (X\_train, y\_train) using the `fit()` method. The training process is monitored using early stopping, and validation data (X\_test, y\_test) is provided to evaluate the model's performance on unseen data during training. The training continues for a maximum of 600 epochs or until early stopping criteria are met.

The model architecture and training process are designed to optimize performance and prevent overfitting, while early stopping ensures efficient training and prevents unnecessary computation.

A screenshot of a computer program

Description automatically generated

Fig. 5: Fitting the ANN Model

**Evaluation Results and Comparison of Models:**

During training, the models learn the underlying patterns in the data and adjust their parameters to minimize the chosen loss function, which, in this case, is the MSLE. The trained models were then used to make predictions on the test set (X\_test) using the predict() method. The predicted values (y\_pred) are obtained and stored for evaluation.

For evaluation purposes, the scikit-learn library was utilized to compute the Mean Squared Logarithmic Error (MSLE) between the true target values (y\_test) and the predicted target values (preds). The MSLE is calculated to evaluate the performance of the models. It is a measure of the ratio between the squared difference of the logarithms of the predicted and true values. The formula for MSLE is given below:

where,

n = number of samples

= true value of the target variable for sample i

= predicted value of the target variable for sample i

The discussion of the results obtained from evaluating each of the five models provides valuable insights into their performance and effectiveness in predicting game ownership based on the dataset features. Below are the results:

1. \*\*Linear Regression Model (MSLE: 2.54)\*\*:

- The linear regression model has the highest MSLE among all the models evaluated.

- This high MSLE indicates that the model's predictions have a significant level of error compared to the true target values.

- The linear regression model may be underfitting the data, as it assumes a linear relationship between the features and the target variable, which may not adequately capture the complexities in the dataset.

2. \*\*Support Vector Regression (SVR) Model (MSE: 1.09)\*\*:

- The SVR model performs better than the linear regression model, with a lower MSLE.

- However, the MSLE is still relatively high, suggesting that the SVR model's predictions have a considerable level of error.

- SVR is a powerful model for regression tasks, but it may struggle with complex relationships and large datasets.

3. \*\*K-Nearest Neighbors (KNN) Regressor Model (MSE: 0.75)\*\*:

- The KNN regressor model exhibits further improvement over the SVR model, with a lower MSLE.

- The lower MSLE suggests that the KNN model's predictions are closer to the true target values compared to the previous models.

- KNN is a non-parametric model that relies on similarity measures between data points, making it suitable for capturing complex patterns in the data.

4. \*\*Random Forest Regressor Model (MSLE: 0.67)\*\*:

- The random forest regressor model demonstrates a significant improvement over the previous models, with a notably lower MSLE.

- The lower MSLE indicates that the random forest model's predictions are even closer to the true target values, suggesting better performance.

- Random forest is an ensemble learning method that combines multiple decision trees, allowing it to capture complex relationships and reduce overfitting.

5. \*\*Artificial Neural Network (ANN) Model (MSLE: 0.48)\*\*:

- The ANN model outperforms all other models, with the lowest MSLE among the evaluated models.

- The significantly lower MSLE indicates that the ANN model's predictions are closest to the true target values, showcasing superior performance.

- ANNs are powerful models capable of learning complex patterns in data, making them well-suited for regression tasks with high-dimensional data.

Overall, the results suggest that more complex models such as Random Forest and ANN perform better than simpler models like Linear Regression and SVR for this particular dataset. The lower MSLE values obtained from Random Forest and ANN indicate their effectiveness in capturing the underlying patterns in the data and making accurate predictions. These findings highlight the importance of selecting appropriate models and techniques tailored to the specific characteristics of the dataset and problem domain.