ESRB Classifier

Testing Design Document

## Dataset

### Data Sources

The classification relies on previously recorded official ESRB ratings to make a prediction about a hypothetical game’s content rating. Although this data isn’t made available by the ESRB itself, many unaffiliated and hobbyist projects have compiled data sets through publicly available sources for use in applications like ours. The data set used in the initial training for the classification contains what we believe is a good cross-section of gaming titles across the parental advisory categories, platform and genre. The data is provided and ingested by the software in a comma separated value (.csv) formatted file which contains:

* A record containing the headings for each column.
* 2396 records of past official ESRB game ratings.
* 34 data points per entry.
* Each game’s title.
* Whether the game released on a console platform.
* Whether the official list of the content descriptors for the title was not available.
* 31 content descriptors indicated by the ESRB.

#### Testing & Validation Data

The original data, once read from the original file, will be stored in memory as a simple java string object and then passed into a custom data set object that can split itself into randomly selected disjoint sets. In the initial read, we will split the entire data set into three parts:

* 70% for use in bootstrapping working data partitions to generate decision trees.
* 20% for sample data used in testing.
* 10% for validation of testing and tuning model parameters.

30% of the data is reserved for our tests and further refining the system to achieve high classification accuracy. To create the most representative sample data set for testing, we use a subset of our initial records as opposed to similar data from another source. We also use a programmatic approach to selection to avoid any bias or human error.

## Unit Testing

### Class – DataSet

#### Test Case 1

Check DataSet constructor with a valid input: (function: DataSet constructor).

Valid Inputs: File Text Object and character delimiter.

Expected Output: Data Set Object with a list of Datapoints otherwise input input is invalid. Error(“Object classified as File Text Object and/or wrong delimiter argument”).

#### Test Case 2

Ensure the function to handle missing data works to fill any absent fields in the input.

Valid Inputs: File Text Object with > 75% of filled in fields (function: handleMissingData() ).

Expected Output: Data Set Object with list of Datapoints with no missing Fields in List of Datapoints.

#### Test Case 3

Test the split function For Dataset Class. The class should be able to split itself to produce subsets as an array list for training, testing, and validation.

Valid Inputs: Instance of Data Set object.

Expected Output: ArrayList of Datasets of size three with no overlap or duplicates between each DataSet. The number of entries in each DataSet should be a 70/20/10 split, otherwise Error(“Split cannot be performed”).

### Class – RandomForest

#### Test Case 1

Random Forest can generate bootstrapped datasets from original Data Set (function: generate bootstrapped DataSet()’s )

Valid Inputs: DataSet.

Expected Output: 2D Array of Datasets, with duplicates(replacement), otherwise handle Error (“Bootstrapped Sets cannot be created”).

#### Test Case 2

Random Forest train based on a data set by generating a list of Decision Trees (function: train)

Valid Inputs: DataSet.

Expected Output: List of Decision Trees otherwise log Error (“Cannot generate list of Decision Trees”)

#### Test Case 3

Random Forest can perform testing on a Dataset and generate an accuracy number.

Valid Inputs: Dataset not seen by training.

Expected Output: An accuracy number and a valid count of tested datapoints equaling the size of the testing set otherwise, log (“Error in training” or “Datapoint used by Training”)

#### Test Case 4

Random Forest should retrain model based on user reinforcement (function: recalculate Decision Trees)

Valid Inputs: Object that contains features contained in recommendations along with Boolean value for like and dislike.

Expected Output: A retrained model, otherwise log Error(“Error in retraining”).

### Class – DecisionTree

#### Test Case 1

Decision Tree is created with correct termination criteria (function: Decision Tree Constructor)

Valid Inputs: Bootstrapped dataset with minSamples and maxDepth criteria.

Expected Output: Tree Object that calls buildTree(), otherwise Error(“Tree could not be generated”).

#### Test Case 2

Decision Tree can generate adhering to termination criteria. (function: build Tree)

Valid Inputs: List of DataPoint.

Expected Output: Tree of depth less than or equal to max depth with no nodes having datapoints that exceed the minimum number of samples otherwise, otherwise Log Error (“Error: Build Tree”)

### Class – Node

#### Test Case 1

Node can be constructed from the set of datapoints (function: Node Constructor).

Expected Output: Node consisting of datapoints otherwise Error(“Node could not be constructed”).

#### Test Case 2

Node can calculate Gini Index and Gini Impurity of Self and child nodes (function: calculate Gini Impurity).

Expected Output: Node consisting of datapoints with Gini Index and Impurity of itself otherwise Error(“Gini Impurity cannot be generated”).

#### Test Case 3

Node can get the best split based on features set and Gini Impurity of children (function: calculate Gini Impurity).

Expected Output: Node can get the best split based on features set and Gini Impurity of children (function: calculate Gini Impurity ).

#### Test Case 4

Each node has a label according to the datapoints with the greatest majority in the node otherwise Error(“No majority label”).

Expected Output: A retrained model, otherwise log Error(“Error in retraining”).

## System Testing

### Classify A Given Game

#### Test Case 1

Positive Classification: (System correctly identifies when a game belongs to a particular category or genre).

Test Data:

* Game Title: "The Legend of Zelda: Breath of the Wild"
* Game Genre: Action-Adventure
* Platform: Nintendo Switch

Expected Outcome:

* The system should correctly classify the game as an Action-Adventure genre for the given platform.

#### Test Case 2

Negative Classification (System does not correctly identify when a game does not belong to a particular category or genre).

Test Data:

* Game Title: "FIFA 22"
* Game Genre: Sports
* Platform: PlayStation 4

Expected Outcome:

* The system should correctly classify the game as a Sports genre for the given platform.

### Recommend Similar and Dissimilar Games

#### Test Case 1

Recommend Similar Games.

Test Data:

* User's Liked Game: "Overwatch"
* User's Disliked Game: "Call of Duty: Warzone"

Expected Outcome:

* The system should recommend games like "Overwatch" based on the user's preference.

#### Test Case 2

Recommend Dissimilar Games.

Test Data:

* User's Liked Game: "Stardew Valley".
* User's Disliked Game: "Dark Souls III".

Expected Outcome:

* The system should recommend games dissimilar to "Stardew Valley" based on the user's preference.

### Training and Retraining Random Forest based on CSV and User Like/Dislike

#### Test Case 1

Training Random Forest with CSV Data.

Test Data:

* Training CSV File: Contains historical user preferences and game features.
* Number of Trees: 100

Expected Outcome:

* The system should successfully train the random forest model using the provided CSV data with 100 decision trees.

#### Test Case 2

Retraining Random Forest with User Like/Dislike Feedback.

Test Data:

* User Liked Game: "Minecraft"
* User Disliked Game: "Assassin's Creed Valhalla"
* Number of Trees: 50

Expected Outcome:

* The system should retrain the random forest model incorporating the user’s like/dislike feedback with 50 additional decision trees.

## Acceptance Testing for Non-Functional Requirements

### Transparency and User Expectation

#### Test Scenario - Ensure Transparency in User Interface

Test Steps:

1. Interact with the system's user interface.
2. Verify that information and actions are presented in a clear and understandable manner.

#### Test Scenario: Meet User Expectations for Response Time

Test Steps:

1. Execute actions required for classification.
2. Measure response time so that it is less than in a reasonable time (Goal < 5 seconds).

### Error Handling with Logging/Reliability

#### Test Scenario - Error Handling in Data Input

Test Steps:

1. Intentionally provide invalid or incomplete data during system interactions.
2. Verify that the system detects and handles errors well.
3. Check if error messages are clear and provide guidance on corrective actions.

#### Test Scenario: Logging for Troubleshooting

Test Steps:

1. Introduce simulated errors into the system.
2. Inspect system logs to ensure that errors are appropriately logged, and information is sufficient for troubleshooting.

### Handle Missing Data Fields

#### Test Scenario: Missing Data in Input Fields

Test Steps:

1. Upload data from csv with missing Data
2. Ensure the system provides appropriate feedback about the missing data and/or fills the data field with default case “1”. If too many fields are missing, make sure that user cannot proceed without providing enough fields in dataset.

### Scalability

#### Test Scenario: Evaluate System Scalability

Test Steps:

1. Simulate increased Data set size.
2. Monitor system response time.
3. Verify that the system scales with similar response times even with larger datasets.