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 Bootsrapped 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”)