CISC 372 Final Project

04.19.2021

Joseph Chan - [18chjc@queensu.ca](mailto:18chjc@queensu.ca) - 20120783

**Abstract**

The goal of the team this time is to predict the ESRB rating of a game. An ESRB rating is used to determine what age range the users of the game should be. This range is by a variety of binary attributes that contain information about the game. The predictions will be made through a variety of machine learning models.

**Introduction**

For our final assignment this semester we had to choose a problem to solve. The problem I decided to do for this assignment is to predict the ESRB ratings of a game. ESRB ratings for a game are used to let the audience know what age group the game was meant for. Currently in the gaming industry there are the big companies that make the big label games, and then there are the indie game designers where hidden gems can be found. The problem that occurs for these indie gamers is that there is a lot to learn. From game development to art, each one takes a lot of time. The purpose of building this model is to allow people who are new into game development to get an idea of what ESRB rating their game will have. They would just have to fill out a quick survey with a For this research assignment multiple models will be tested to determine the best one for this problem. The models will be built on a dataset found from kaggle called Video Games Rating By ‘ESRB’. There are two datasets: a training set and a testing set. The purpose of this assignment is to build the best model so it can be used for future indie developers.

**Problem statement**

The research problem is to define a model that can predict the ESRB rating of a game. ESRB ratings are split into four categories: Everyone, Everyone 10+, Teen, and Mature. There is a kaggle data set with 1895 games listed and 35 attributes. There is the y-attribute being the ESRB rating represented by E, ET, T, and M. Out of the 34 attributes left, 32 of them are correlated to the y-attribute. These 32 attributes are all binary attributes that consist of either one or zero. The purpose is to define a model which a user may use to figure out which rating their games belong to by filling out those attributes.

**Proposed Method**

**Data Processing**

The solution for the research problem will be in the form of a machine learning model. The first step I took was to design a neural network model. First I had created the program for the model using google colabatory. Next I downloaded the training and testing data from the kaggle dataset source. I then created a repository on github to upload the two datasets. Now the code in my google colaboratory can obtain the datasets. After obtaining the datasets I had to split the data into the training and testing sets. The two datasets were put into a pandas dataframe. The x datasets for training and testing sets are created by dropping the ‘title’ and ‘esrb\_rating’ columns. The y datasets for training and testing sets are created by selecting the ‘esrb\_rating’ column from the training and testing datasets. Next the y datasets are put in a label encoder to perform one hot encoding for the y values.

**Neural Network**

The next step was to design the neural network model. The model has an input layer with the shape of 32 nodes. The output layer has four output nodes, one for each class and a softmax activation function. There are five dense layers in between the input and output layer with each of them having 200 nodes. The second and fourth dense layers have a relu activation function. The model used categorical cross entropy for its loss, accuracy as it’s metrics, and used an adam optimizer. The model was put through 10 epochs with a batch size of 64. The validation split used was 70% training 30% testing. The model went through roughly five to six changes before this final version was chosen. Between models the performance was roughly 79% to 83% with the final model having 82.5%.

**XGBoost Classifier**

The next model I created used a randomized grid search xgboost classifier. The importing of data and dataset splitting process was the same except one hot encoding was not used. This model is able to use string values as class values so it was not needed. The model put the necessary values into categorical features. The categorical features will go through one hot encoding and the empty values would be filled with 0. The Pipeline uses an xgboost classifier with softmax classification. The grid search would go through four different numbers of estimators and four different max depths. The grid itself had 30 folds and 16 iterations for a total of 480 fits. These settings were found after 10 different trials, with the accuracy going between 75% to 85%, the final model having 84.8%.

**Birch Clustering**

The next model used was birch clustering. This model had all the same beginning steps, going with the one hot encoding option. The settings for the models was a threshold of 0.01 and 4 clusters. This model performed very badly even with a few changes at around 20% accuracy. Since the accuracy was so bad compared to the others the model was dropped.

**K-Nearest Neighbours**

The next model tested was K nearest neighbors classifier. This model used regular encoding rather than one hot encoding. The model was tested with a different amount of neighbours but, the one used in the end is 3. The accuracy obtained was 80.8%.

**Random Forest**

The final model tested was the random forest model. This model used a regular label encoder changing the letters into numbers. The number of parameters in this model after a certain number did not change the accuracy too much. The parameter chosen in the end is 300 estimators. The final accuracy changes every time it is run but, the accuracy is around 84%

**Experimental Results**

There were five models tested. Below is the accuracy and confusion matrix of the models.

|  |  |  |
| --- | --- | --- |
|  | Accuracy | Time |
| Neural Network | 85.6% | 3 seconds |
| K-Nearest Neighbour | 80.8% | instant |
| XGBoost | 83% | 2.4 minutes |
| Birch Cluster | 24.6% | instant |
| Random Forest | 84.4% | instant |

*Model Accuracy and Time*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| True/Predicted | E | ET | M | T |
| E | 90 | 1 | 0 | 9 |
| ET | 5 | 112 | 0 | 9 |
| M | 0 | 0 | 61 | 29 |
| T | 5 | 19 | 6 | 154 |

*XGBoost classifier*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| True/Predicted | E | ET | M | T |
| E | 95 | 4 | 0 | 1 |
| ET | 4 | 112 | 0 | 10 |
| M | 0 | 0 | 55 | 35 |
| T | 1 | 17 | 6 | 160 |

*Random Forest classifier*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| True/Predicted | E | ET | M | T |
| E | 95 | 4 | 0 | 1 |
| ET | 4 | 112 | 1 | 9 |
| M | 0 | 0 | 72 | 18 |
| T | 2 | 21 | 15 | 146 |

*Neural Network classifier*

**Lessons Learned/ Improvements**

Throughout the experiment there were many different stages of trial and error, where I learned of things that I could improve on. One of the many things learned is that longer is not always better. As seen simply by the xgboost classifier even though it runs almost 100 times longer than the other classifiers the accuracy is quite similar. I’ve also found that just running neural networks for more epochs does not increase the accuracy as it may overfit. Another lesson learned is that more is not always better. For example, while adding more layers to a neural network may make it stronger the increase will happen slowly regardless. It is actually better to start with a small model and figure out the hyperparameters that work with the model. Once the model has been perfected, adding the layers after will provide a lot more results than just using extra layers as a solution. One of the tricky parts was splitting the data and the preprocessing needed for the class labels. As I proceeded onto the later models the strategies used became easier to understand from previous experience. By the last model I already knew what to do and had not needed to research methods. One of the things not done that could have been added to increase accuracy was to create a voting ensemble with the models. This was a result of time constraints, however this is easily done in an excel file by taking the majority prediction. No further calculations would have been needed as all the accuracies were similar so the predictions all had the same weight. With a further understanding of the steps in data mining, the next project I will do will go much smoother.

**Summary**

At the end of the experiment five different models were tested. Out of five of them four of them had around 80% accuracy with one of them being around 20%. Since the birch clustering model performed significantly worse at 20% accuracy this model was dropped. Out of all the models only one took more than a few seconds and that one was the xgboost classifier. I decided to create a confusion matrix of the xgboost classifier, neural network method and the random forest method. I chose the xgboost classifier as this method took longer and may have found a better solution. The neural network method was chosen because it had the highest accuracy. The random foresting method was chosen as most of the attributes are binary making a good split for the trees. I have not chosen any of the methods by accuracy as they are all within a five percent range not making too much of a difference. In this esrb estimator model a prediction that predicts an label with stricter restrictions is better than the other way around. The order of the labels goes from E,ET,T, and M with E being the lowest. So if the actual label is M but, the prediction is E then everyone can play a mature game. Where as the other way around if a game that is E is predicted to M then only some people can play the everyone game. The xgboost method predicted 25 less strict labels, the neural network model predicted 30 less strict labels and the random forest method predicted 21 less strict labels. If this model was to be used for official games that were going to be released I would use the random forest model as even though it is slightly less accurate it has less serious mistakes. However, if the model is just used for people that want to see what rating their game would get then I suggest the neural network model as it is more accurate. It is important to note that Both the random foresting and neural network models will return different results every time it is run but, on average the neural network still outperforms the random foresting. In conclusion for people that want to see what rating they should release their game as, I would provide them the random foresting model to use.

**References**

1. <https://www.kaggle.com/imohtn/video-games-rating-by-esrb>
2. <https://scikit-learn.org/stable/>
3. https://www.tensorflow.org/