An analysis of the game industry over time - a data mining perspective -

Lisa Denzer, Vlad Limbean, Ivan Mladenov, and Joachim Sogn

***Abstract*—A breakdown of historical video-game releases. This paper focuses on clustering of the data, classification of game titles and implementation of an artificial neural network tasked with sales value prediction. Data cleaning and pre-processing also take a titular position.**

I. Introduction

The game industry is to a large extent defined by hardware life-cycles. What do you run your games on? Which is the subjectively better platform? The first generation of consoles started back in the days of 1972 with the progenitor of video games, ”Pong” released for ”Color TV-Game” console.

For this project we decided to work with a dataset from Kaggle featuring video game sales. We chose this dataset because most group members were familiar with the overall topic and shared an interest in working with the provided data. Above that the dataset seemed to provide a reasonable number of attributes and entries.

The dataset starts off with machines and game titles from the 2nd console generation. For example, the Atari 2600 console and titles like E.T. the Extra-Terrestrial (the first worst game of all time) and Pac-Mac (an instant classic). Entries continue to present day.

All cleanup, pre-processing and algorithm implementation is done in Python. Throughout our group, we used different setups of Python; some of us used Anaconda with Spyder as the IDE. A short outline of these frameworks and tools is given in the individual chapters.

II. Dataset

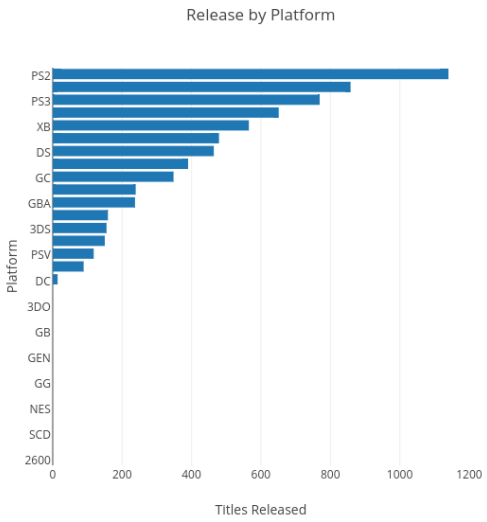
A. Overall information

The dataset contains information on 16,719 games titles, platforms on which they are available, game genre, year of release, critic and user information, parental rating and sales data by region.

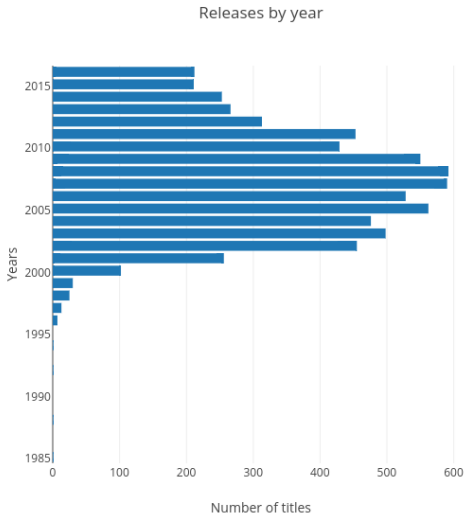
- Dataset is extracted from Kaggle.com
- Based on information from Metacritic.com and VGChartz.com
- Roughly 6000 games from earlier generations, lack Year of Release, Sales and Score information. Heavy pre-processing efforts will be required.
- The remaining entries in the dataset do not lack information. This is due to the emergence of game journalism in the late 1990s, thus providing us with ample information on the market’s progress over time.

B. Descriptive Statistics

The dataset consistently contains information concerning game titles, platforms for which they have been released and genre. The plot below describes the number of games released for all platforms across all generations.



The medium of video games becomes a fully fledged global market force by 1995. This is described by the chart bellow illustrating title releases by year.



C. Missing data visualized

As previously mentioned, our dataset goes back to the early '80s. Much on the information concerning critic and user scores and regional sales is missing from the early days of the industry.

The white gap in the image bellow describes the sales data missing for the early '80s. This offers a rough indication of the overall data missing throughout the data set. The X axis details the Critic Score, the Y axis describes the year of release for a given title and the Z axis represents global unit sales in millions.

III. Pre-Processing and Missing Value Replacement

As described earlier, the dataset has a fair number of missing values. For the purpose of using the full dataset in our classification algorithm and artificial neural network we decided to clean the data with as much accuracy as possible.

A. Missing data

In terms of raw input, the dataset is missing the following number of attributes:

Attribute	Number of values missing
Name	2
Platform	0
Year of release	269
Genre	2
Publisher	54
North America Sales	0
EU Sales	0
Japan Sales	0
Other Sales	0
Global Sales	0
Critic score	8582
Critic count	8582
User score	6704
User count	9129
Developer	6623
Rating	6769
Total games: 16,719	

B. General clean-up

- We start off by importing the .csv file in a python data frame which allows for easy manipulation.
- We remove every entry in the dataset that does not have a title for the respective video game. Additionally, we remove any entry where the year of release later than 2017. This is done because these entries are missing most of the attributes.
- Outliers are removed using zScore measurement on all of the Sales attributes. This process is not applied to the Score attributes, as they have predefined range.
- Since the User Score column contains entries labeled "to be discovered", we replace those with NaN values. This is done so we can convert the column to a numerical attribute. Furthermore, every entry is normalized to match the range of its Critic Score counterpart (by multiplying every entry by 10).
- Finally, we bin all the data by the given hardware generation. This metric refers to the life-cycle of a given console generation by roughly 6-year-intervals. Distinct generations are represented by numbers from 2 to 8. This range is due to our dataset's earliest console being of the 2nd generation and the latest of the 8th.

C. Missing value replacement

- Missing year of release values are replaced by the median after grouping the dataset by platform. That way, entries get filled in with a year that is close to the life cycle of the platform they were on, instead of the median of the whole period.
- Missing information concerning Developer teams is replaced by the median of the grouped game titles. That way, multiple entries for the same game get matching developers. If any missing values persist past this point, we replace them by the median of the grouped Publisher companies. This is done because developers usually work with the same publishers. If any null values remain, we fill them in with the publisher. We consider this last step to be a last resort as it may drastically skew the dataset. It is applied because most entries that remain null are older games, and based on our knowledge of the industry, companies acted as both developers and publishers.
- For Critic and User score and Count values we fill in the mean values that result from grouping by Genre and Publisher. If any entries still have missing values, we repeat the process but instead group by Platform and Publisher.

- The missing parental advisory values (i.e. Rating column) are replaced with the median after grouping by Genre. The reasoning is that specific genres are usually targeted towards specific groups of people (for example, sports games usually receive an E rating). If any values are missing past this point we replace with the median after grouping by Platform, then by Developer, after which we group by Publisher. Finally, if there are any remaining missing values we replace with the median of the Rating itself.

- Games that have a year of release after 2015 are removed. These games were not yet released when the dataset was published (December 2016) or were just recently put on the market, so their attributes are either incomplete, or are in the process of evaluation.

D. Generating new properties and normalization

- We start by creating the boolean attribute "has great critic score" thus splitting games with scores above and bellow 80. The same is performed for "has great user score".

- We generate the average score a game has. It is computed using the following formula:

$$\frac{(criticscore * criticcount) + (userscore * criticcount)}{(criticcount + usercount)}$$

- We create the "has great average score" attribute after computing the average scores using the same threshold value as the critic and user score booleans mentioned above.

- We create a best sales region column. It is a categorical attribute and reflects which region had the largest proportion of sales.

- We create bins to separate sales performance. We tested our classification algorithms with different intervals until we found some that are both concise and balanced in terms of number of elements in them. The final bins we decided upon are as follows: 0 to 0.05, 0.05 to 0.1, 0.1 to 0.2, 0.2 to 0.4, 0.4 to 0.7, 0.7 to 1.25 and over 1.25. These values refer to millions of units sold.

- Game scores (all of Critic, User and Average) are also binned by interval: 0 to 35, 35 to 50, 50 to 65, 65 to 85, and 85 or above.

- A categorical representation of the publisher of the game is generated. This representation reflects on the results from the clustering algorithm that was implemented. More on that in section IV.

- Finally, we normalize all values to a scale of 0 to 1. This is done for all numerical attributes using the following formula:

$$\frac{(attributevalue - min(attribute)}{(max(attribute) - min(attribute)}$$

  The result is always saved in a new column with a postfix "_Normalized" so we don't overwrite the original value.

- Save the resulting data frame into a new .csv so as to not re-compute it on each run.

E. Results

At the end of the whole process, the number of null elements in the dataset looks as follows:

Attribute	Number of values missing
Name	0
Platform	0
Year of release	0
Genre	0
Publisher	0
North America Sales	0
EU Sales	0
Japan Sales	0
Other Sales	0
Global Sales	0
Critic score	0
Critic count	0
User score	0
User count	0
Developer	5036
Rating	5141
Total games: 11,089	

As a reflection, we also experimented with filling in all of the developers and ratings, but the results were skewed and the two attributes proved to not be trustworthy for the clustering and the classification algorithm.

F. Segregating the data

After examining the resulting dataset, we found out that segregation is something that can improve results. Out of the 11,089 games, only 486 were released before 1995. Furthermore, only 7.41% of the sales happened in that time frame. Since the industry clearly changed after that, we decided it would be irrelevant to use old games when predicting sales. It also makes sense given the fact that the first major platforms (Playstation, Nintendo 64) came on the market around that time and completely turned the market around.

IV. Clustering

A. Introduction

One of the properties used for sales prediction is the company that publishes the game. However, automatically transforming the Publisher attribute into a category or using

pure statistical data of the publisher is not enough for an accurate prediction. The reason being that we have no idea whether a specific publisher is running a good business venture. If in fact we do, then there is no classification of exactly how good it is. Therefore, we implemented a clustering algorithm to analyze the different publishers and find similarities between them.

B. K-Means

The K-Means algorithm uses the center of a cluster to calculate the distances (or similarities) between different data entries. It creates K number of clusters, and puts arbitrarily chosen data entries into those clusters. These objects act as initial cluster centers. Then, each object in the data is assigned to its closest cluster (in terms of distance). After the process is complete, new cluster centers are assigned by calculating the mean points of all objects currently in the cluster. The process is repeated, and objects are again assigned to their closest cluster. This is done until objects no longer change their cluster.

C. Implementation

For running the algorithm, we used the scikit library in python. It has several options for clustering. Of particular note is the possibility to choose the degree of randomness employed by the algorithm in selecting cluster data. We decided to use ”k-means++”, which guarantees partial randomness while also carefully choosing the starting clusters far apart from each other. Using ”full” would ensure total randomness. We tested the option but resulting clusters were marginally less accurate relative to using ”k-means++”.

We chose attribute which made sense for defining publishers and their level of market performance: number of games published, sales, public perception (in terms of critic and user scores). Some of the other attributes were used experimentally, but to mixed results. We avoided using any non-numeric attributes, as the algorithm only works with attributes that can have a definable mean.

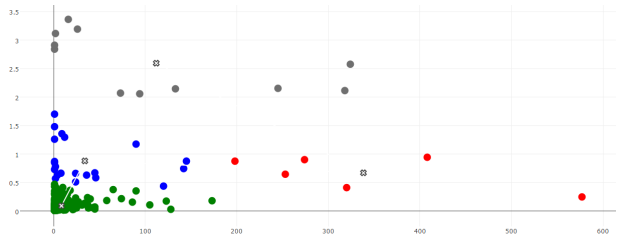
The aforementioned attributes do not exist in the original data, but are calculated at the beginning of the process. These attributes are normalized to make sure the clustering doesn’t weigh one more than another. Once the KMeans algorithm is performed the results are visualized on a scatter plot.

D. Results

To understand the overall historical performance of publishers we defined limits for each attribute and segregated them into different categories. We pre-defined the publishers by the following categories:

- **Green Underdogs:** Publishers with low amount of games and low sales
- **Silver Surfers:** Publishers with low amount of games and high sales
- **Red Dwarfs:** Publishers with many games and low sales
- **Blue Ribbons:** Publishers with many games and high sales

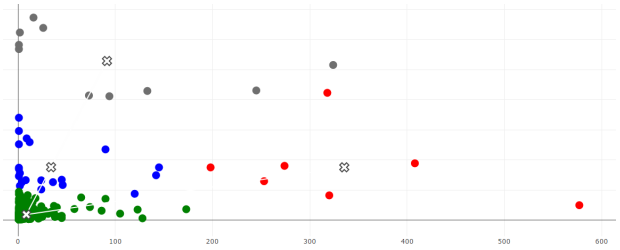
The graph below shows the generated distribution of companies from the cleaned dataset. The Y axis represents the average sales a company achieves (in millions of units), while the X axis describes the number of games put to market by that company.



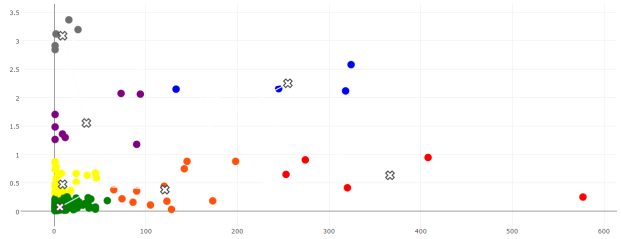
The issue with this distribution is that it relies on the author’s expert knowledge of the game industry. Consequently, we divided the four types of publishers by a threshold value for number of sales: 1.5 million units; and a threshold value for number of games put to market: 100 titles.

If these values seem like a rough approximation, **it’s because they are**. In order to determine a more realistic understanding of how publishers can be accurately classified we relied on something more accurate and less biased than our understanding of the data - clustering algorithms.

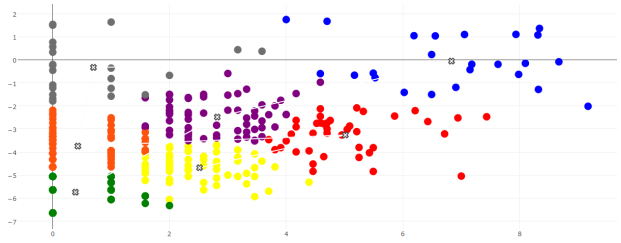
The conclusions after running K-Means are that the threshold values are in fact not at 100 titles and 1.5 million units sold. Furthermore, simplifying the clustering to two threshold values will not provide us with an accurate representation of the market. The graph bellow illustrates the initial results of running the K-Means clustering. Clusters are identified by the white crosses indicating centroids.



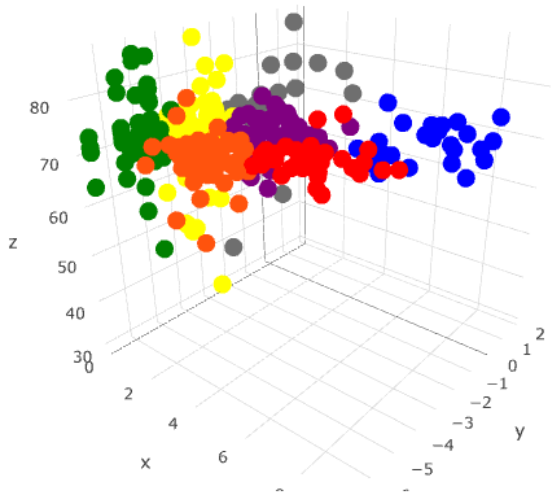
The distinction between publishers is clearly visible. If an invisible cut were to be made on 200 games and 2 cuts and on 500,000 and 1 million sales, the clusters from these cuts will be exactly the same. The obvious problem is that a big chunk of companies are in the ”Green underdogs” section, which might skew any prediction algorithm. For this reason, we tried running KMeans with a larger number of clusters. We believe the best results came for $K = 7$. The companies are distributed in a more balanced way and the groups show clear distinctions between each other in terms of market significance.



However, a small problem still remains. Even though the two attributes are normalized, the data seems to be compressed into the left part of the scatter plot. This might skew the results of the clustering and therefore a fix is needed. A typical technique to fix this problem is to apply logarithmic scales. They allow for a wider spread of most companies by valuing small distances in small numbers in the same manner as big distances between big numbers. Every entry in the data frame has a log_2 applied to both attributes (for example, Namco Bandai's 577 games are replaced with 9.17 and it's average sales of 0.2476 replaced with -2.0139). Thus the distinction between companies is even greater and the data is yet again more balanced.



Finally, we are interested in finding out if the opinion of game critics and users has any significant bearing on the publisher's type. The 3-dimensional graph integrates the average critic score on the z axis.



We notice the centroids are still mostly spread over the X and Y plane and have little to no variation on the z axis. This shows that critic opinion does not have significant weight when trying to define the type of company.

After carefully observing the resulting scatter plots, we believe the following clusters accurately represent publisher companies. We have defined the following color codes for them:

- **Green Underdogs:** Publishers with low amount of games and low sales. More specifically, 1 or 2 games published, and less than 100,000 units sold.
  Example: Stainless Games
- **Orange Donalds:** Publishers with low amount of games and slightly better sales - more than 100,000 but less than 300,000 units
  Example: Bohemia Interactive
- **Silver Surfers:** Publishers with low amount of games and high sales figures (300,000 or more)
  Example: Mojang
- **Yellow Submarines:** Publishers with a respectable number of games on the market, but low sales (200,000 or less)
  Example: Taito
- **Deep Purple:** Publishers with recognizable game titles released (between 4 and 16) and solid sales (above 200,000 but less than 1 million)
  Example: Telltale Games
- **Red Dwarfs:** Publishers with many games and disappointing sales (less than 200,000 units sold)
  Example: Paradox Interactive
- **Blue Ribbons:** Publishers with many games and high sales
  Example: Nintendo, Electronic Arts, Take-Two Interactive

This classification is applied to every entry in the dataset and will be used in predicting game sales.

V. Classification - Decision Tree

We used a decision tree to find attributes which influence the number of games sold of a specific game. The team used the cleaned data from python and the data science platform RapidMiner to execute the decision tree.

A. Attributes

After using multiple different attributes in the tool, we got the best results with the following ones:

- **Average publisher sales**
- **Average score**
- **Year of release**
- **Publisher type**

The first three attributes are numeric and binned by entropy. We use entropy to order the attributes in homogeneous rather than random bins. We thus get more information about the attributes prior to executing the decision tree. For example, the attribute year_of_release was divided into two bins where bin 1 contained games released prior to 2015 and bin 2 games released after. Similarly the attribute average_score was divided into 5 bins, the attribute average_score into 5 bins and averagepublisher_score into 6 bins.

B. The Decision Tree

The creation of a decision tree in RapidMiner depends on a number of criteria. The first is to select a gain ratio. Here we had the choice between the information gain ratio or the gain ratio. The difference is that the gain ratio penalizes attributes with many subsets while the information gain ratio favors them. We found it best to use the gain ratio as it narrowed down the tree and gave us a more clear and readable result. Our final minimal gain ratio was set to 0.2. After testing with the minimum leaf size and the minimal size for split we selected them to 25 and 4.

See the appendix under decision tree for our solution.

C. Analysis of Decision Tree result

The decision tree first splits on the publisher type. By looking at the tree, it is difficult to find a clear tendency of whether most publishers sell a lot of games based on the type they were assigned by the clustering algorithm. The decision tree does however find a tendency for the third publisher type (Blue ribbons), which are large companies with high average sales. In this case we found the result below. Note that year_of_release is split into years before 2015 and after 2015. This may be caused by reasons discussed earlier such as that many games after 2015 not yet have received a score.

- The game is likely to have less than **0.5 million sales** if the game has a score **less than 74.2.**
- If the score is **between 74.2 and 79.2**, it is **equally likely** whether a game will sell more or less than 0.5 million.
- If the score is **larger than 79.2** then 70% of the games has **more than 0.5 million sales**.

VI. Supervised learning

Looking at our dataset we conclude that sales figures play an important role and are of great impact to publishers and developers. Consequently, we aim to predict the financial success of a game. Thus, we came up with the idea of training an **artificial neural network** to predict the sales of future game releases.

Artificial Neural Networks have been inspired by the human brain and its capability to learn and make predictions based on past experiences. At the core of this concept lies the neuron, which is responsible for processing incoming signals and deriving predictions. The basic concept revolves around the principle of passing input through a sequence of layers which determine the ultimate prediction.

There are several types of activation functions available suiting different requirements, e.g. the rectifier function. The input is taken in and processed through the entire sequence of layers for each data entry. This is done multiple times and for each iteration. The prediction is checked against the actual outcome using an error-function. The result is then ”back-propagated” through the neural network and the weights are adjusted accordingly. The learning rate of the algorithm can be used to define how much the weights are adjusted throughout the process. Consequently, the error rate should decrease while the accuracy of predictions should increase over the training phase. Once training is complete, the neural network uses test-data and a confusion matrix to evaluate performance of predictions.

The advantage of artificial neural networks is the ability to process large amounts of data and to make predictions that are less influenced by assumptions.

A. Tools for Neural Network implementation

There are several libraries and tools available which we used: Theano, Tensorflow and Keras. **Theano** is a Python library which enables us to ”define, optimize, and evaluate mathematical expressions involving multi-dimensional arrays efficiently”

Tensorflow is a Google open source software library for numerical computation using data flow graphs. Nodes in the graph represent mathematical operations, the graph edges represent multidimensional data arrays (tensors). The system is applicable for a wide variety of other domains.

Keras is a high-level neural network API, written in Python and running on top of TensorFlow or Theano.

B. Implementation

Input values are referred to as independent variables and output values as dependent variables. An output value can be binary, continuous or categorical. Before the actual implementation of the neural network, these variables are chosen and pre-processed. Our interest lies in the sales figures. Consequently, sales figures are the dependent variable we aim to predict. Factors that influence sales are year of release, genre, publisher and platform, which we chose as independent variables.

An important detail here is that some of our variables are originally of categorical type. We take care of the transformation in our data cleaning class. Furthermore, the independent variables need to be standardized (mean = 0, variance = 1) or normalized (values between 0 and 1).

The first step of the implementation is to split the dataset into a training set and a test set. For the splitting we use the sklearn.model_selection.train_test_split method from sci-kit learn to split arrays into random train and test sets. The size of the test set is set to 30%.

To initialize the neural network, we use the Keras Sequential model and start with defining a sequence of layers. Our first layer is a fully connected neural network layer which must have a defined input shape. We therefore pass an input_shape argument to the first layer. Following this we specify output dimension, initialization-function and activation function: $model.add(Dense(input\_dim = len(properties\_to\_use), output\_dim = 32, init ='$

$uniform', activation =' relu'))$.

The setup of the first two layers at once is a feature of the Keras sequential model we are using. The dense function from Keras randomly initializes the weights of the nodes to small numbers close to zero. We chose the uniform distribution here.

We start forward-propagation by applying the rectifier activation function. The nodes are activated (from left to right). The weight of the node limits the impact of the activation. Consequently, the higher the activation function of a node (how close is it to "1") the higher the impact the node will have on the neural network. Propagation takes place until the predicted result is achieved.

There is no general rule of thumb on how many nodes are optimal for the hidden layer. We decided to take the average of the number of nodes in the input layer and the number of nodes in the output layer.

The output layer has 7 nodes based on the number of bins we are trying to predict.

VII. Results

Throughout the process of implementing the neural network we experimented with different input attributes. Setting up the framework with Keras first seemed surprisingly simple but as we went on we encountered a number of complex issues. Depending on what we wanted to predict, we needed to carefully investigate which functions to use. We first decided to use the "had_great_sales" attribute we generated in the cleaning process, which is a boolean based on whether the global sales figures were great or not. We started running the algorithm at a batch-size of 32 with 100 epochs and witnessed an increase of accuracy for the first 15-20 epochs about 10%. The accuracy stabilized around 70%. However, when applying the prediction, the neural network almost always predicted bad sales figures.

We continued experimenting with the neural network, by changing the threshold for great sales and feeding it different attributes, changing the batch-size as well as number of epochs. During this process we have seen many iterations without satisfying results, which were often connected to the way we had pre-processed our data. However, with our initial sample run of boolean predictions, we have been able to demonstrate that using more effort and time, it would most likely be possible to use an artificial neural network to predict the success of a game in terms of sales based on attributes such as platform, publisher, year or genre.

Finally, we decided to try and guess the sales figure interval instead of a simple boolean. This reflects the global sales bin attribute we created in the data cleaning and normalization process. We also make use of the publisher type, best sales region, and score bins we segregated the games in. The learning curve of the network is seen below.

Even though the final accuracy isn't perfect, the network had variance in the bins it predicted. Based on the test set we fed the network, the final accuracy was 36.13%. The confusion matrix of the different predictions is shown below.

	Bin_0	Bin_1	Bin_2	Bin_3	Bin_4	Bin_5	Bin_6
Bin_0	**757**	284	267	190	101	84	59
Bin_1	0	**1**	0	0	0	0	1
Bin_2	56	99	**169**	122	70	30	20
Bin_3	34	63	98	**126**	100	83	112
Bin_4	0	0	0	0	**0**	0	0
Bin_5	0	0	0	0	0	**0**	0
Bin_6	13	10	14	24	40	32	**82**

VIII. Conclusion

We set out with a dataset which had strong representation for games released in years after 2000, meaning few missing values for more recent generations. Our pre-processing efforts managed to replace missing values and make them usable for the algorithms employed without skewing the data a lot. We acknowledge the missing values as a necessary evil for a market as young as that of video games.

Having clustered the data by publisher we adjusted our understanding of the market. The results were used to generate a new property (Publisher_Type) that was used in the classification algorithms. We gained a more accurately divided picture of the publisher scene. It came to no great surprise that big publishing companies are not strongly impacted by critic reviews of games they put to market. We find it valuable to have segmented the publisher company scene an adjust our understanding of it with empirical evidence.

For our classification and supervised learning we aimed to predict future game sales based on the type of company that is publishing them. The global sales can be predicted with varying success based on their score. We noticed that accuracy for smaller companies which are not market incumbents tends to degrade. We attribute this circumstance to the overall heterogeneous nature of the game industry. The business does not have the tried and tested pedigree of

something like the automotive industry. The more variance between business model, the more noise, the more difficult a prediction task.

IX. Tables and graphs