Integrating Web Data on Video Games and Companies

Project Report

presented by
Group 4
Kuan-Min Chen, Rahul Taneja, Roman Salzwedel, Se Won Yeu, Vasili Bocicariov

submitted to the
Data and Web Science Group
Prof. Dr. Christian Bizer
University of Mannheim

30.11.2018

Contents

1 Introduction

In our project, we focus on building an integrated database of video games and video game developers which will be informative to video game players and professionals working in the industry alike. The video game industry is a rising market that accounts for a global revenue of no less than $137.9 billion in 2018. It is one of the key market that makes up a large share of sales for software developers and publishers[1]. At the same time, the worldwide number of video game players has grown over 2.2 billion and it is expected to surpass 2.6 billion by the end of the decade in the future, according to the latest figures[2]. With an increasing number of gamers, the number of video entertainment related websites is also growing. In this context, the idea of our project is to provide a single integrated data source which combines information from different video game websites. The goal of our work is twofold. First, we want to offer video game players a common source of information where they can find details about their favourite games and insights on the latest trends in the market including ratings from experts and users, potential sellers, prices, and many other. In this regard, we hope our data will help consumers to gain useful insights and assists them to make better buying decisions. Second, our integrated data includes information on sales and game developers working in the game entertainment industry. We believe that our combined data can offer interesting new visions that can assist on business decision making and drive video game businesses to a success. Simply by exploring and mining this data, one will be able to gain a better understanding of current video game trends. For example, our integrated data can provide answers on manifold questions such as *'Which game platform currently generates the most revenue?'*, *'Which genre types are most popular among the users?'*, *'How does game experts' judgment affect the market sales'*, or *'How does the revenue of games differ from the worldwide regions?'*. On the following pages we present the results for the three main phases of our data integration project and provide a discussion on the challenges we faced.

2 Data Translation

2.1 Data

To build a comprehensive data source of video games and developers, we integrate four different datasets. The datasets cover our two main classes: Game and Developer. All datasets are obtained by scraping the gaming websites outlined in Table 1 using the Python and Java libraries BeautifulSoup, Selenium WebDriver, and Jsoup3. The data is collected between 28th September and 3rd October 2018.

The datasets cover information on video games and the respective game developers from 2010 to 2018. More detailed descriptions for each of the input datasets are given in Table 2 and Table 3. Additionally, where attributes with more than 30 percent of missing values are marked as '(MV)'[1].

Table 1: Data Set Information

Data Set	Source	Format	Class	# of Entities	# of Entities after duplication removal
vgchartz.csv	VGChartz.com [3]	CSV	Game	17,957	17,929
vgchartz.csv	VGChartz.com	CSV	Developer	3,103	3,103
igdb.json	IGDB.com [4]	JSON	Game	45,611	35,727
igdb.json	IGDB.com	JSON	Developer	7,827	7,827
metacritic.csv	Metacritic.com [5]	CSV	Game	7,370	7,370
developers.csv	Wikipedia [6]	CSV	Developer	569	569

Table 2: Data Set Information - Part II

Data Set	# of Attributes	List of Attributes
vgchartz	15	Publisher, Name, Console, Genre, Release Date, Developer User Score (MV), Critic Score (MV), VGChartz Score (MV) Total Sales (MV),NA Sales (MV), PAL Sales (MV), Japan Sales (MV), Other Sales (MV), Last Update (MV)
igdb	8	Publishers(MV), Name, Platforms, Genre(s), Release Date Developer (MV), Matruity Rating(MV), Expert Score(MV)
metacritic	8	Publisher, Name, Platform, Genre(s), ReleaseDate, UserScore, MetaScore, Rating
developers	7	Developer, City, Autonomous area (MV), Country, Est., Notable games (MV), Notes (MV)

Table 3: Attribute Intersection with Integrated Schema

Class Name	Attributes Name	Datasets in which attribute is found
Game	nameOfGame	vgchartz, metacritic, igdb
Game	publisher	vgchartz, metacritic, igdb
Game	developer	vgchartz, igdb
Game	platform	vgchartz, metacritic, igdb
Game	genre	vgchartz, metacritic, igdb
Game	releaseDate	vgchartz, metacritic, igdb
Game	userScore	vgchartz, metacritic
Game	maturityRating	metacritic, igdb
Game	expertsScore	vgchartz, metacritic, igdb
Developer	nameOfDeveloper	developers, igdb

[1]Table 3 only shows attributes that appear in at least two of the input datasets

2.2 Consolidated Schema

In this phase of the project, the integrated XML schema is built. For the game class we define 13 attributes and for the developers class we define 7 attributes. Thus, there are 20 attributes in total. To come up with an integrated schema four main steps are taken. First, we define the two classes for our integrated schema. Second, we manually identify correspondences between the attributes in our input datasets. Third we assign the respective attributes to our game and developer class. Finally, we build our integrated schema resolving conflicts by successively changing the structure of the individual schemata. The next step is to do the schema matching for each of our datasets and to create unique IDs for the games and developers class. The data from each dataset is transformed, normalized and mapped to the integrated XML schema using MapForce. The output of each mapping is an XML file which is used as input for the Identity Resolution.

2.3 Transformations

We use several transformations to transform the input data. To mention some of them, the reported user and expert scores are transformed to a common scale ranging from 0 to 100. This is done in order to have the comparable scale for all datasets. Also, the releaseDate attribute is normalized in all the datasets in order to have the same date format. For VGChartz, the missing values for userScore and expertScore are set to -1, since the data is of type integer. Similarly, the missing values for totalSales, NASales, PALSales, JapanSales and otherSales are also set to -1. For more details on the other transformations, please refer to the attached .mfd files.

3 Identity Resolution

3.1 Gold Standard

To apply a set of supervised machine learning (ML) classifiers for the entity matching, we first create a gold standard. For each of the binary comparisons between the input data we manually label a set of record pairs as matches or non-matches. More specifically, we apply the following iterative procedure: First we randomly add 100 games where title, platform and release date are identical and verify manually that these games actually constitute true matches. Similarly, we choose 250 games for which title, platform and release date are all different and verify manually that these games actually constitute non-matches. Next we add around 50 to 100 interesting corner cases to the gold standard. Examples of non-matching corner cases in our data include pairs such as *Dead Rising 4* <> *Dead Rising*,

Mega Man X Legacy Collection 2 <> *Mega Man X Legacy Collection 1* or *Forza Horizon 2* <> *Forza Horizon 2: Storm Island*. Examples of matching corner cases include pairs such as *Marvel Pinball 3D* == *Marvel Pinball*, *Oddworld: Stranger's Wrath HD* == *Oddworld: Stranger's Wrath*, or *Need for Speed* == *Need for Speed (2015)*, as well as cases where the game titles do match, but the release dates differ [2]. Given the comparison between Metacritic and VGChartz the initial version of the gold standard contains ca. 450 record pairs.

Next we start with the machine learning part and iteratively update the gold standard with further corner cases. After each run of a number of different classification algorithms, thresholds and comparators, we evaluate the most promising models in more detail. Therefore, we randomly select subsets of ca. 20 cases out of the predicted correspondences. These cases are then checked and subsequently added to the gold standard and the training is continued. This process is repeated several times until we reach an acceptable model performance. Given the case of the comparison between Metacritic and VGChartz the initial gold standard is updated six times. In the end, it contains 629 labeled record pairs.

3.2 Data Preprocessing

In order to normalize the data and improve the performance of the entity matching, the following data preprocessing steps are required. All string values are transformed to lower-case letters and punctuation signs are removed. Second, stop words such as *'of'*, *'the'*, *'by'* are removed. To remove stop words, we use the NLTK Stopword list. [7] Third, we remove the phrase *'Read the review'* which is attached to some of the game titles in the Metacritic data. In addition, we standardize the maturity rating across the input datasets by transforming it to the Entertainment Software Rating Board (ESRB) rating schema. Finally, the platform names are standardized for all input datasets, changing for instance both PlayStation4 and Ps4 to PS4. Additionally, we normalize the abbreviations of company names in the datasets. Since there exist various acronyms and abbreviations in different languages, this requires some manual effort. For instance, *'s.r.l'* refers to *'Sociedad de responsabilidad limitada'* in Spanish which corresponds to *limited liability company* or *LLC* in English. Unfortunately, our input data is not duplicate free. Thus, as a last step, we remove all duplicates from the input datasets [3]. This leads to the removal of 9,884 records in IGDB and 28 records in the VGChartz data (see Table 1).

[2] Note that we do not consider cases such as *MX Vs ATV: Supercross* <> *MX vs. ATV Supercross* since this cases are dealt with in the data preprocessing. For Data Preprocessing see below.

[3] This is done by removing all but one record that share the same values on all attributes.

3.3 Similarity Measures, Comparators and Blocking Keys

To find corresponding records between our input data, we define several different comparators and blocking keys. During entity resolution for game, we focus on the attributes nameOfGame, publisher, genre, and releaseDate. For releaseDate we define a `1`, `2`, and `3YearComparator` that check whether the release date of two games is within the respective range. In addition, we implement a `WeightedDateComparator`. For game name, publisher, and genre we apply a variety of string similarity measures. In addition to the predefined `Equal`, `Jaccard`, and `Levenshtein` similarity, we implement `Damerau`, `JaroWinkler`, `JaroWinklerTfIdf`, `MongeElkan`, `MongeElkanTfIdf`, and `SoftTfIdf` similarity measures from the Java SecondString API [8]. In terms of developer, we focus on the nameOfDeveloper attribute only. With the predefined `Equal`, `Jaccard`, and `Levenshtein` similarity, and `Damerau`, `JaroWinkler`, `JaroWinklerTfIdf`, `MongeElkan`, `MongeElkanTfIdf`, and `SoftTfIdf` similarity measures as well for developer. A full overview of the implemented comparators is shown in Figure 2 and 3. Given the size of our data running identity matching with no blocking is not feasible [4]. Therefore, we rely on two main blocking schemes. The first blocker blocks by release year and platform, while the second blocker blocks by platform and the first two characters of the game title. Both blockers lead to a reduction ratio of more than 99% and reduce the runtime significantly. Likewise, for the developer models, we use a blocker blocking by the first two letters of company name. For a more detailed description of the efficiency and performance of the blockers see Figure 2 and 3.

3.4 Machine Learning

We use stratified subsampling to split the gold standard into a training and a test set. Given that the distribution of matching and non-matching record pairs in our gold standards is unequal (ca. 32% matching and ca. 68% non-matching examples), we balance the training set by upsampling the minority class until we reach a ratio of approximately 1-to-1.

To evaluate the quality of our Identity Resolution we first set up a 'simple' baseline model for each comparison. For the video game class, the baseline model just uses the `EqualComparator` for the game name and blocks by platform and release date. More specifically, this means that the baseline model predicts two games to be a match if (and only if) their title, platform, and release date match exactly. Similarly, for the developer class, we use an `EqualComparator` for the company name and block by the first two letters of the name. All later models that explore many more comparators, blocking schemas and classification

[4] Based on a subsample of 2,000,000 comparisons, the estimated runtime with noblocking is ca. 80 days.

algorithms are evaluated against this baseline. Once again, for a full overview of the models and their respective performance refer to Figure 2 and 3. For instance, Figure 2 shows that the baseline model for the comparison between Metacritic and VGChartz (first row) yields a precision, recall, and F1-Score of 1.0, 0.8, and 0.889 respectively, while finding 3,240 correspondences in the data.

Next to the baseline, Figure 2 also shows a subset of the more advanced models we tried. Among the tested models are `Logistic Regression, Naive Bayes, Decision Tree, AdaBoost, SVC, and RandomForest`. In general, the best performing model for the game class is a `RandomForest` classifier in combination with `MongeElkan` and `MongeElkanTdIdf` comparators for game name. The comparison of Metacritic and VGChartz results in a performance of F1-Score of 0.909. For the game class the more advanced models use the `PlatformNameGenerator` blocker described above with a runtime between 20 seconds and 3:10 minutes. For developers we find that simple logistic with a final threshold of 0.9 gives the best performance (2nd row in Figure 3). This model set-up results in 667 correspondences for the comparison of IGDB and Wikipedia.

After applying the best ML models, we analyze the remaining errors of our classifiers. For the game class, errors include cases such as *Warhammer 40,000: Dawn of War III <> Warhammer 40,000: Dawn of War III - Limited Edition* and *Hatsune Miku: Project Diva Future Tone <> Hatsune Miku: Project Diva Future Tone - Colorful Tone* which are predicted to be matches, although we consider them to be different games. This highlights that the final classifiers has still difficulties to predict separate versions of the same game (such as Limited Editions) or extensions and add-ons, despite many similar cases that were added to the gold standard. This seems especially to be the case when the title strings are long and share many common tokens as in the examples above. Another unique kind of error is highlighted by the example of *We Are The Dwarves <> The Dwarves*. This is because, by removing the stop words *'we'* and *'are'* in the first string, the classifier believes these titles to be the same. This is obviously wrong, however, since overall stop word removal has improved the performance of our entity matching, we are forced to accept this trade-off. The main error that persists for the developer class are cases where two or more developers are listed in one of the input datasets but not in the other. For instance *Armor Project <> Armor Project/Bird Studio*[5]. Fortunately, the kind of errors mentioned for both classes are infrequent, and the overall model performance is acceptable.

Overall, we are able to substantially improve the performance of our entity resolution compared with the baseline models. In terms of F1-Score, our final classifiers lead to an increase in F1-Score between 2.0 and 27.4 percentage points. In addition, we are able to significantly increase the number of predicted corre-

[5]This issue could be addressed with higher manual efforts in preprocessing. However, this is beyond the scope of this project.

spondences found in the data. It turns out that for most comparisons a combination of `RandomForest` and `MongeElkan` similarity with final threshold of 0.8 delivers the best results [6].

< MetaCritics - VgChartz >

Classifier	Matching Rule	Blocker	Threshold	P	R	F1	# Corr	Run Time	Reduction Ratio
Baseline	GameNameComparatorEqual()	GameBlockingKeyBy PlatformDateGenerator()	0.5	1	0.8	0.889	3,240	1.00 sec	0.99983
RandomForest (REPTree) {"-I", "300", "-S", "42", "-K", "7"}	GameDateComparator1Years(), GameDateComparator2Years(), GameDateComparatorWeightedDate(), GameNameComparatorEqual(), GameNameComparatorJaccard(), GameNameComparatorMongeElkan(), GameNameComparatorMongeElkanTfIdf(), BackwardSelection(false)	GameBlockingKeyBy PlatformName Generator()	0.8	0.93	0.889	0.909	4,897	20.01 sec	0.99866

Figure 1: Identity resolution results - MetaCritic vs VgChartz

< IGDB - MetaCritics >

Classifier	Matching Rule	Blocker	Threshold	P	R	F1	# Corr	Run Time	Reduction Ratio
Baseline	GameNameComparatorEqual()	GameBlockingKeyBy PlatformDateGenerator()	0.5	1	0.75	0.857	3,636	2.1 sec	0.9998
HoeffdingTree {"-S", "0", "-L", "2"}	GameDateComparator1Years(), GameDateComparator2Years(), GameDateComparator3Years(), GameDateComparatorWeightedDate(), GameNameComparatorDamerau(), GameNameComparatorEqual(), GameNameComparatorJaccard(), GameNameComparatorJaroWinkler(), GameNameComparatorJaroWinklerTfIdf(), GameNameComparatorLevenshtein(), GameNameComparatorMongeElkan(), GameNameComparatorMongeElkanTfIdf(), GameNameComparatorSoftTfIdf(), GamePublisherComparatorEqual(), GamePublisherComparatorJaccard(); GamePublisherComparatorJaroWinklerTfIdf(); GamePublisherComparatorLevenshtein(); GamePublisherComparatorMongeElkan(); GamePublisherComparatorMongeElkanTfIdf(); GamePublisherComparatorSoftTfIdf(), GameGenreComparatorJaccard(), BackwardSelection(true)	GameBlockingKeyBy PlatformName Generator()	0.8	0.852	0.821	0.836	5,529	1 min 20.77 sec	0.9979

Figure 2: Identity resolution results - IGDB vs MetaCritic

[6] For an overview of the predicted group size distribution see Figure 9.

< IGDB - VgChartz >									
Classifier	Matching Rule	Blocker	Threshold	P	R	F1	# Corr	Run Time	Reduction Ratio
Baseline	MachineLearning SimpleLogistic: Equal	GameBlockingKeyBy PlatformDateGenerator()	0.5	1	0.385	0.556	3,862	2.1 sec	0.9999
RandomForest (REPTree) {"-M", "3.0", "-S", "42"}	GameDateComparator1Years, GameDateComparator2Years, GameDateComparator3Years, GameDateComparatorWeightedDate, GameNameComparatorDamerau, GameNameComparatorEqual, GameNameComparatorJaccard, GameNameComparatorJaroWinkler, GameNameComparatorLevenshtein, GameNameComparatorSoftTfIdf, GameNameComparatorJaroWinklerTfIdf, GamePublisherComparatorEqual, GamePublisherComparatorJaccard, GamePublisherComparatorLevenshtein	GameBlockingKeyBy PlatformName Generator()	0.8	0.957	0.733	0.83	8,278	3 min 10.04 sec	0.9988

Figure 3: Identity resolution results - IGDB vs VgChartz

< IGDB Dev - Dev >									
Classifier	Matching Rule	Blocker	Threshold	P	R	F1	# Corr	Run Time	Reduction Ratio
Baseline	MachineLearning SimpleLogistic: Equal	DeveloperBlockingKeyBy NameGenerator()	0.4	0.485	1	0.653	34,179	2.0 sec	0.9923
SimpleLogistic	DeveloperNameComparatorEqual(), DeveloperNameComparatorJaccard(), DeveloperComparatorMongeElkan(), DeveloperComparatorMongeElkanTfIdf()	DeveloperBlockingKeyBy NameGenerator()	0.9	1	0.625	0.769	667	4.81 sec	0.99233

Figure 4: Identity resolution results - IGDB Dev vs Wiki Dev

< VG Dev - Dev >									
Classifier	Matching Rule	Blocker	Threshold	P	R	F1	# Corr	Run Time	Reduction Ratio
Baseline	MachineLearning SimpleLogistic: Equal	DeveloperBlockingKeyBy NameGenerator()	0.5	1	0.154	0.267	324	1.4 sec	0.9922
HoeffdingTree	DeveloperNameComparatorDamerau(), DeveloperNameComparatorEqual(), DeveloperNameComparatorJaccard(), DeveloperNameComparatorJaroWinkler(), DeveloperNameComparatorLevenshtein(), DeveloperComparatorJaroWinklerTfIdf(), DeveloperComparatorMongeElkan(), DeveloperComparatorMongeElkanTfIdf(), DeveloperComparatorSoftTfIdf()	DeveloperBlockingKeyBy NameGenerator()	0.8	0.818	0.692	0.75	514	2.43 sec	0.99225

Figure 5: Identity resolution results - VG Dev vs Wiki Dev

< IGDB Dev - VG Dev >									
Classifier	Matching Rule	Blocker	Threshold	P	R	F1	# Corr	Run Time	Reduction Ratio
Baseline	MachineLearning SimpleLogistic: Equal	DeveloperBlockingKeyBy NameGenerator()	0.5	0.743	1	0.853	179898	7.5 sec	0.9925
SimpleLogistic	DeveloperNameComparatorEqual(), DeveloperNameComparatorJaccard(), DeveloperComparatorMongeElkan(), DeveloperComparatorMongeElkanTfIdf()	DeveloperBlockingKeyBy NameGenerator()	0.8	0.977	0.808	0.884	2,731	17.90 sec	0.99225

Figure 6: Identity resolution results - IGDB Dev vs VG Dev

4 Data Fusion

4.1 Gold Standard

To evaluate the quality of our data fusion, we first establish a ground truth. To create this gold standard for data fusion, we collect the 'true' values manually from trustable sources. In the context of video games, trustable sources are the websites of the main video gaming brands like PlayStation, Nintendo, and Microsoft. On the respective websites we find information about all PS3, PS4, PSV, NS, Wii, WiiU, as well as Xbox360 and XboxOne games. Since no comparable single source of information exists for PC games, we rely on information from Steam store. The gold standard for the video game class contains 20 records. The first 10 records are selected randomly, while the second 10 records are selected based on cluster-level consistency. Hereby, we aim to include records with a low level of consistency and a high amount of data conflicts. To create a gold standard for developers we use a similar strategy. The developer gold standard consists of 10 records.

4.2 Inspecting the Data and Correspondences

To fuse the data for games and developers we only consider those attributes which are present in at least two of the input datasets (see Table 3). For video games, these attributes are name, publisher, platform, genre, release date, and maturity rating. For game developer, we solely fuse the company name attribute. Figure 7 reports the dataset and attribute specific densities. For instance, it shows that the attribute density for publisher varies between 61% in IGDB, and 100% for Metacritic and VGChartz.

Attribute	Input			Output	
	Density per Attribute			Overall Accuracy	Overall Density
	vgchartz.xml	igdb.xml	meta.xml		
nameOfGame	1	1	1	0.95	1
platform	1	1	1	1	1
maturityRating	0.00	0.31	0.84	0.95	0.76
releaseDate	1	1	1	0.85	1
genre	1	0.92	0.67	0.7	1
publisher	1	0.61	1	0.8	1
Average	0.83	0.81	0.92	0.88	0.96

Figure 7: Accuracy and density result of games

Attribute	Input			Output	
	Density per Attribute			Overall Accuracy	Overall Density
	vgchartz.xml	igdb.xml	developer.xml		
nameOfDeveloper	1	1	1	0.9	1
Average	1	1	1	0.9	1

Figure 8: Accuracy and density result of developers

In addition, Figure 9 depicts the group size frequency for our correspondences. Since we use three input datasets, we expect that each real world entity (video game or developer) is described by at most 3 records. Figure 9 shows that overall we find 9,839 groups. However, a closer look at the table reveals that there are some groups of size four and larger. For video games, there are 349 out of 9,839 cases or 3.5% with a group size of 4 or larger. Similarly, out of the overall 1,831 groups for game developer, 221 or 12% have size of 4 or larger. This is clearly erroneous. However, a closer inspection of the data reveals that the assumption of no duplicates does not hold for the developer data due to errors in the input data. For instance, the same companies that have changed their name in the past, are recorded multiple times under the old and new name. This is also to a lesser extent true for video games, where many of the erroneous groups contain episodic games[7].

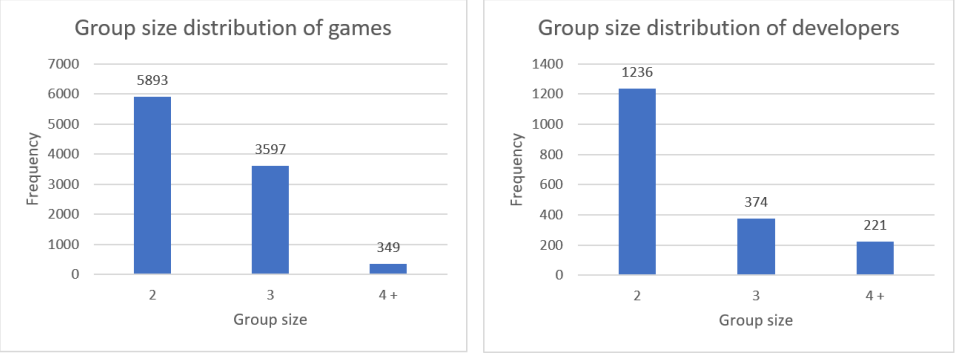

Figure 9: Group size and frequency overview of games and developers

4.3 Creating a Data Fusion Strategy

Next, we set up a data fusion strategy. This means that we define attribute-specific conflict resolution methods for all the attributes that we want to fuse into our final dataset. Since the input data are scraped from various websites on the internet, no obvious ranking or hierarchy in terms of data provenance is given in our case.

[7]Episodic games are games with multiple episodes, for instance Resident Evil Revelations 2 - Episode 1 <> Resident Evil: Revelations 2 - Episode 2: Contemplation

In addition, the websites do not provide reliable information about their last update and do not allow for chronological ordering. Therefore, we implement and test a variety of different attribute fusers and evaluation rules. We start by using a 'simple' `shortestString` fuser for the attributes nameOfGame, publisher, and genre. The `shortestString` fuser prefers short strings over long strings, which makes sense in the context of games in order to select titles that do not contain additions such as *'3DS'*, or *'(Xbox Version)'*. Likewise, it prefers company names that do not contain additions such as *'AG'*, or *'LLC'*. For releaseDate (and platform[8]) we use a `voting` fuser. For maturity rating, which is only present in two out of the three datasets, we implement a `favourSource` fuser. Initially, we assigned the highest provenance to Metacritic. All fusers use an exact equality evaluation rule. This strategy leads to an overall accuracy of 0.53. In addition, it reveals large differences in the attribute specific-accuracy. While the maturity rating value in our final data matches the true value in the gold standard with an accuracy of 95%, the accuracy of genre is just 5%. A closer look at the errors highlights some of the main problems our conflict resolution methods encounter. First, we find cases where the difference between the fused values and the true values is marginal, such as *FlatOut 4: Total Insanity* <> *Flatout 4: Total Insanity* or *Bandai Namco Games* <> *Namco Bandai Games*. To deal with these obvious matches, we change the evaluation rule from equality to tokenized Jaccard similarity which is case insensitive and ignores punctuation. Second, a more detailed error analysis reveals that the genre attribute is most often correctly captured by the IGDB data, but not by the other two data sources. Thus, we change our conflict resolution strategy for genre to use the `FavourSource` fuser. These changes lead to a significant increase in the accuracy. The overall accuracy increases to 0.8, while the attribute-specific accuracy for nameOfGame, publisher, and genre, increases to 0.95, 0.7 and 0.60, respectively. Further inspection of the remaining errors reveals two more striking patterns. The first issue concerns the differences in release dates. The 'true' values of the release date are obtained from U.S. websites and correspond to the release date for North America. However, some of the data sources (sometimes) report the release date for Europe or Australia. This is most given in the case of *White Knight Chronicles*, where one data source reports the release date for North America (2010-02-02), while the other two sources report the release date for Europe (2010-02-05) and Australia (2010-02-25) []. Since all these dates lie within the same month, we decide to change the evaluation rule for release date, in order to allow for a tolerance of 30 days. A similar issue regards the genre and publisher attributes. For instance, our fused record lists the genre as *Role-playing (RPG), Tactical, Strategy* while the gold standard reports *Role-playing (RPG), Strategy*. Since we consider this to be a reasonable close match we

[8]Since we block by platform during the Identity Resolution phase, there are no data conflicts and the choice of a fuser is negligible.

change the evaluation of genre and publisher to tokenized Jaccard distance with a threshold of 0.50. This means that when at least half of the mentioned genre categories match we declare the genre to be similar[9]. The overall as well as the attribute-specific accuracy for our final data fusion strategy is depicted in Figure 7 and 8. This approach leads to an overall accuracy of 88% with attribute-specific accuracy of 95% for nameOfGame and maturityRating, 80% for publisher, 85% for releaseDate, 70% for Genre. Last but not least, we fuse the values for user and expert score to enrich our final data with additional information. We fuse the values using the `averageFuser`. However, since user and expert ratings are inherently subjective and no trustable source exist from which the true scores can be obtained, we left these two attributes out of the evaluation presented above[10]. All in all our final dataset contains 46,495 entities with and overall density of 96% as shown in Figure 7.

5 Summary

To sum it up, we attempted in this project to build a coherent database of video games - an ever growing and important part of the IT industry - by integrating data from four different web sources. We focused on two main classes: video games and video game developers. First, we translated our data into an integrated schema. Second, we tested and implemented various matching rules to find corresponding records across the different input data. Third, we fused the attribute values from the different data sources into a single, consolidated record for each real world entity by using various attribute-specific conflict resolution functions and evaluation strategies. Overall, our final dataset for games contains 46,495 records with density of 0.96. This represents an increase of ca. 15 percentage points compared to our largest input dataset. For developer dataset contains 8,537 records with density of 1.0. The overall accuracy of our final fused dataset compared to the true values in the gold standard is 88% for video game class, and 90% for the game developer class (considering those attributes which were present in at least two of the input datasets). We hope that our efforts will contribute to the web integration research and offer the gaming industry a single yet comprehensive source of information.

[9] For publisher we use the same approach to handle where

[10] To provide a maximum of information, we added those records to our final XML dataset, which are only present in one of the input datasets and which were not considered during data fusion

References

[1] https://www.gamesindustry.biz/articles/2018-04-30-global-games-market-to-hit-usd137-9-billion-this-year-newzoo

[2] https://techfruit.com/focus/number-gamers-worldwide-hits-2-2-billion/

[3] http://www.vgchartz.com/gamedb/

[4] https://www.igdb.com/reviews/games

[5] https://www.metacritic.com/browse/games/score/userscore/all/all/filtered?view=detailedsort=desc

[6] https://en.wikipedia.org/wiki/List_of_video_game_developers

[7] https://gist.github.com/sebleier/554280

[8] http://secondstring.sourceforge.net/