Integration of Music Streaming Platform Data

Web Data Integration Project

presented by Team 1
Aakriti Istwal
Matriculation Number 1871754
Abhay Augustine Joseph
Matriculation Number 1822329
Anirudh Yadav
Matriculation Number 1823309
Pujit Golchha
Matriculation Number 1823413
Sharan Shyamsundar
Matriculation Number 1870221

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

December 2022

Contents

1 Introduction

With the introduction of numerous streaming platforms like Spotify, Amazon Music, YouTube Music, Deezer, etc., the market for song streaming services has experienced tremendous growth over the past ten years. By using comprehensive song data to identify streaming habits, these streaming service platforms can acquire a competitive advantage over their rivals. Our integrated collection will be an exhaustive group of tracks and associated audio elements. This can be used as a foundational dataset for future music recommendation systems that take into account a user's distinct likes and aural characteristics. But creating such a unified data source might provide a variety of difficulties, from straightforward ones like typographical or omission errors to semantically more difficult ones like the same song being re-released with a new audio translation logged as a separate entity.

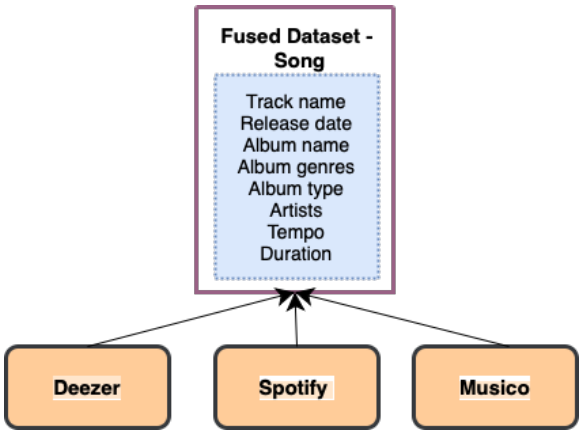

Figure 1: Fused dataset overview

Figure 1 depicts the process of integrating the three datasets into a final integrated dataset. The following three sections elaborate: *a.* Integrated schema mapping transformations. *b.* preparing data matchers. *c.* fusing data with conflict-resolving strategies. Our code and all other related inventory files are available on GitHub repo: WebDataIntegration.

2 Schema Translation

For the process of schema translation, we use the *Altova Mapforce* software. With reference to the Spotify and Deezer datasets, the data is scraped from their respective websites using their *python APIs* [1, 3]. The raw data is converted into *.csv*

Dataset	Source	Class	# Entities	# Attributes	Attribute List
Spotify	Spotify API	Song	43230	17	track_name, release_date, artists, album_type, album_title,duration popularity, acousticness, danceability, energy, instrumentalness,key, liveness, loudness, mode, speechiness, tempo
Deezer	Deezer API	Song	26659	10	track_name,duration, release_date,explicit, bpm, gain, artists, album_title, album_genres, album_type
MusciOSet	MusciOSet Website	Song	11297	8	track_name, artists, popularity, explicit, album_title, album_type, album_genres, release_date

Table 1: Dataset profiles for *Spotify, Deezer & MusicOSet* data sources.

format using *pandas*. The MusicO dataset is available as a relational database in *.csv* format at [2]. The detailed dataset profile information for each data source is provided in Table 1.

For transformations, since our data is available at song granularity we directly map almost all of the fields with the integrated schema as reference, in Table 2. This operation is denoted by *"1-to-1 direct map"*. The "id" field for each of the data sources is created by appending an incrementing sequence value to the data source name i.e. *data-source-name_inc-count*; this operation is denoted by *"iterative sequence"*. For the "release_date" field we transform the date values containing only month and year, to the start of that particular month; containing only year, to the start of that particular year, and finally, the missing date values to start of the year "1900"; this operation is denoted by *"date rounding-off"*. For the "tempo" and "duration" fields the missing values are replaced with -1.0 and -1 respectively; this operation is denoted by *"replace null_to_-1"*. For both the fields "album_genres" and "album_type", we replace the missing values with ""(empty string); this operation is denoted by *"replace_null_to_empty"*. Furthermore, list attributes were directly mapped to lists in our final *XML* file, this operation is denoted by *"1-to-1 list map"*.

Class	Attribute Details	Parent Sources	Type & Transformation Logic
song	id: *string*	*D1, D2 & D3*	iterative sequence
song	track_name: *string*	*D1, D2 & D3*	1-to-1 mapping
song	release_date: *string*	*D1, D2 & D3*	1-to-1 direct mapping, date rounding-off
song	artist_names: *list[str]*	*D1, D2 & D3*	1-to-1 list map
song	album_title: *string*	*D1, D2 & D3*	1-to-1 direct mapping
song	album_type: *string*	*D1, D2 & D3*	1-to-1 direct mapping replace null_to_empty
song	album_genres: *list[str]*	*D2 & D3*	1-to-1 list map replace_null_to_empty
song	duration: *integer*	*D1 & D2*	1-to-1 direct mapping, replace null_to_-1
song	explicit: *boolean*	*D2 & D3*	1-to-1 direct mapping
song	popularity: *integer*	*D1 & D3*	1-to-1 direct mapping
song	tempo: *integer*	*D1 & D3*	1-to-1 direct mapping replace null_to_-1

Table 2: Integrated schema description alongside schema translation transform operations analysis for data sources *Spotify(D1*), Deezer(D2*) & MusicOSet(D3*).*

3 Identity Resolution

We use the *WInte.r* framework in the identity resolution phase to implement matching algorithms to identify unique identity correspondence pairs across the data sources [5, 7]. In this section, we first prepare our gold standards, implement and compare matcher performances, plot group distribution sizes and finally, conduct error analysis from debug reports.

3.1 Gold Standard Preparation

For the evaluation of our identity resolution algorithms, three separate gold standard comma-separated (*.csv*) files are constructed with over 500 entity pairs, each. These files are subsequently used for the performance calculation of correspondences derived from *Spotify-Musico*, *Spotify-Deezer*, and *Musico-Deezer* matchers. We create our gold standard with the assistance of simple matchers and blockers.

- Specifically, for the *Spotify* and *MusicOSet* datasets we have Spotify-IDs as the unique identifier for tracks. Using these IDs and checking for matching track names, we create the perfect matches. For corner case matches we compare entities with the same Spotify IDs and then look at the tracks which do not have matching track names or album names or both mismatched. To further evaluate corner cases, one such case includes joining the datasets on the track name and manually labeling the records based on the other fields. Another such case includes joining these datasets on the track name, checking if either album name is contained in the other, and then manually labeling the records as matches or non-matches based on the other fields. In order to create the perfect non-matches we perform a cross-join on these datasets and use the pairs which do not have the same Spotify IDs and mismatching track names.

- For creating the other two gold standards, the perfect matches include entities that have the same track name and album name. In the case of corner cases and perfect non-matches, the same approach is used as in the case of Spotify-Muscio.

(Note: Different editions of the same album name are considered positive matches. For example: ”Baby One More Time (Digital Deluxe Version)” and ”Baby One More Time” are considered the same album. However, track editions that are specified as ”Remastered” are considered as non-matches while the rest are considered as the same song provided they have the same album name.)

All the gold standards are created according to the following conditions: 20% of all cases are perfect matches, 30% are corner cases and 50% are non-matches.

3.2 Attribute Section for Matching

There are five overlapping song attributes between all three data sources namely, track_name, release_date, artist_names, album_name, and album_type. For Identity Resolution the attribute album_type is not as reliable as the other overlapping fields and also does not have a significant amount of unique values, therefore we create comparators only for the other attributes.

3.3 Matching & Blocking Experimentation

Our baseline comparators rely solely on the Track Name and Album Name, comparing only if the names match exactly, the results of which are displayed in Table 3. Then, for each of the data sources, we methodically construct linear matching criteria. We initially utilize a straightforward comparison based on the track name using *Levenshtein similarity*. Further, we enhance this comparator by including text preprocessing. In addition, we develop multiple comparators based on various variables such as album name, release date, artists, and duration. Since the artist attribute is a list, we used *Generalized Maximum of Containment* along with *Levenshtein Similarity* in order to compare each element of the list. For the release date, we compute the year similarity with a max difference of 1 year. As for the duration based comparator, songs having a difference of fewer than 10 seconds are considered similar.

Specifically, we find that on combining the list attribute artists with overlapping coefficient similarity, release date with a permissible absolute difference of 1 year, preprocessed track name with Levenshtein similarity and album name with maximum of containment, we obtain a great metric boost as shown in Tables 5, 6 and 7. Our final best matching rules for *Deezer-Musico* & *Spotify-Musico* is stated by $sim(x, y) = 0.4 \times sim_{track\_anomaly}(x, y) + 0.2 \times sim_{album}(x, y) + 0.2 \times sim_{release\_year}(x, y) + 0.2 \times sim_{artists}(x, y)$ with thresholds of 0.8 for each. However, in the case of datasets *Spotify* and *Deezer* we have an additional field *Duration*. Utilizing the created matcher for duration, we now have the best matching rules for *Spotify-Deezer* given by $sim(x, y) = 0.2 \times sim_{track\_anomaly}(x, y) + 0.2 \times sim_{album}(x, y) + 0.2 \times sim_{release\_year}(x, y) + 0.2 \times sim_{artists}(x, y) + 0.2 \times sim_{duration}(x, y)$ with a threshold of 0.8

We tried and tested different blockers for our matching algorithms, namely:

- TrackNameBlocker: Uses the entire Track Name as the blocking key.
- TrackNameStartBlocker: Uses the first four characters of the Track Name.
- AlbumNameBlocker: Uses the entire Album Name.
- AlbumNameStartBlocker: Uses the first four characters of the Track Name.
- 2YearBlocker: Uses intervals of 2 years from the earliest release date to the latest release date.
- DecadeBlocker: Uses the decade of the release date.

Matching Rule	Datasets	Threshold	Blocker	P	R	F1	Corr	Time
TrackName Equals	Deezer-MusicOSet	0.6	Standard Track Name	0.70	1	0.83	9065	0.54 sec
TrackName Equals	Spotify-Deezer	0.6	Standard Track Name	0.47	1	0.64	15626	0.523 sec
TrackName Equals	Spotify-MusicOSet	0.6	Standard Track Name	1	0.97	0.987	14719	0.286 sec
Track Name & AlbumName Equals	Deezer-MusicOSet	0.6	Standard Album Name	1	0.61	0.76	2855	0.344 sec
Track Name & AlbumName Equals	Spotify-Deezer	0.6	Standard Album Name	0.93	0.59	0.72	3620	1.149 sec
Track Name & AlbumName Equals	Spotify-MusicOSet	0.6	Standard Album Name	1	0.63	0.77	7073	0.292 sec

Table 3: Identity Resolution Performance Table for Baseline Matchers

With our matching analysis we find that, while both standard blocker approaches for *album_name* and *track_name* are similar, blocking by the start of album and track names takes longer. Furthermore, as we see in table4, in comparison with the sorted neighborhood blockers, the standard blockers are more accurate and significantly faster. Hence, we use the standard blocker for our final matcher's result, as demonstrated in Tables 5, 6 and 7.

We are unable to execute linear combination with *No Blockers* as well as *Decade Blocker* as we run into an *'Out of Memory'* error. This is due to the formation of large buckets, leading to vast number of comparisons.

#	Matching Rule	Blocker	P	R	F1	Corr	Time	Red. Ratio
1	Track&Album&Year&Artist	StandardAlbumName	0.84	0.76	0.80	3,227	5.8 sec	0.999
2	Track&Album&Year&Artist	StandardAlbumNameStart	0.83	0.98	0.90	3,374	100.00 sec	0.984
3	Track&Album&Year&Artist	StandardTrackName	0.78	0.98	0.875	3,433	0.225 sec	0.999
4	Track&Album&Year&Artist	StandardTrackNameStart	0.79	0.98	0.875	3,485	22.901 sec	0.996
5	Track&Album&Year&Artist	Standard2Year	1	0.64	0.78	2,362	114.00 sec	0.9799
6	Track&Album&Year&Artist	SNTrackName	0.79	0.98	0.87	3,485	231.00 sec	0.9567
7	Track&Album&Year&Artist	SNAlbumName	0.85	0.88	0.86	3,306	234.00 sec	0.9566

Table 4: Performance Table of Blockers Used

#	Matching Rule	Threshold	Blocker	P	R	F1	Corr	Time
1	Rule1:TrackName	0.8	Standard Track Name	0.70	1	0.83	11,129	0.896 sec
2	Rule2:TrackName & AlbumName	0.8	Standard Track Name	0.73	1	0.84	3,569	1.317 sec
3	Rule3:TrackName & AlbumName & Release Year	0.8	Standard Track Name	1	0.64	0.78	2,170	1.361 sec
4	Rule4:TrackName & AlbumName & Release Year & Artist	0.8	Standard Track Name	0.79	0.98	0.87	3,433	1.667 sec

Table 5: Identity Resolution Performance Table for *Deezer* and *MusicOSet* data sources.

#	Matching Rule	Threshold	Blocker	P	R	F1	Corr	Time
1	Rule1:TrackName	0.8	Standard Album Name	0.93	0.84	0.88	4,338	5.500 sec
2	Rule2:TrackName & AlbumName	0.8	Standard Track Name	0.85	0.98	0.91	4,738	2.426 sec
3	Rule3:TrackName & AlbumName & Release Year	0.8	Standard Track Name	0.98	0.75	0.85	3,018	2.467 sec
4	Rule4:TrackName & AlbumName & Release Year & Artist	0.8	Standard Track Name	0.90	1	0.95	4,344	4.056 sec
5	Rule5:TrackName & AlbumName & Release Year & Artist & Duration	0.8	Standard Track Name	0.90	1	0.95	4,308	3.890 sec

Table 6: Identity Resolution Performance Table for *Spotify* and *Deezer* data sources.

3.4 Group Size Distribution

The group size distribution for our final correspondences are demonstrated in Figure 2. Here, we observe some correspondences falling onto groups of sizes $\{5, 6\}$, these pairs are manually analyzed to comprehend the findings.

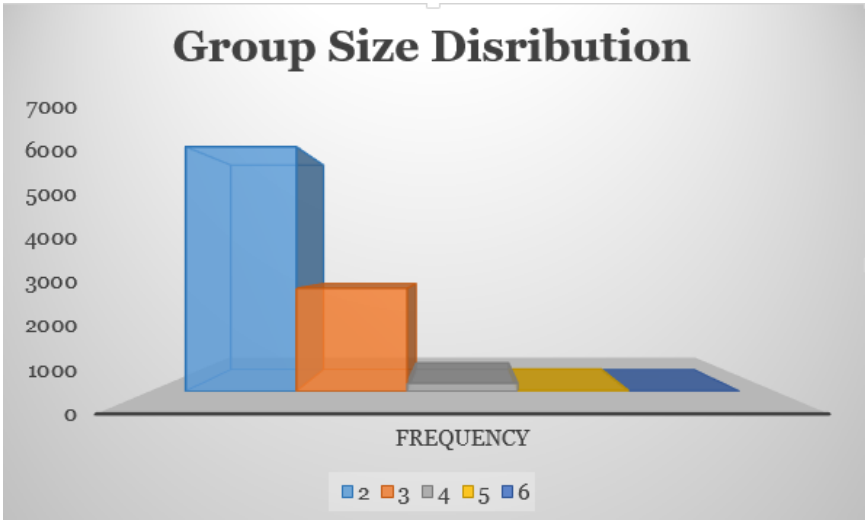

Figure 2: Final group size distribution plot

#	Matching Rule	Threshold	Blocker	P	R	F1	Corr	Time
1	Rule1:TrackName	0.8	Standard Track Name	1	0.97	0.99	17,583	0.700 sec
2	Rule2:TrackName & AlbumName	0.8	Standard Track Name	1	0.96	0.98	7,908	1.140 sec
3	Rule3:TrackName & AlbumName & Release Year	0.8	Standard Track Name	1	0.91	0.95	7,551	1.235 sec
4	Rule4:TrackName & AlbumName & Release Year & Artist	0.8	Standard Track Name	1	0.97	0.99	7,811	1.958 sec

Table 7: Identity Resolution performance table for *Spotify* and *MusicOSet* data sources.

3.5 Error Analysis

The high F1-scores can be attributed to the decent quality of the data sources utilized for matching. However, relative performance improvements in more recent reported matchers could be a result of the right blocking techniques and the addition of informative attributes.

3.5.1 Tracks Belonging to the Same Album

One reason we observe the maximum group size distribution of 6 (as shown in figure 2), even though we have significantly high F1 scores, is because our track_name comparator uses Levenshtein similarity. Therefore, tracks FEAR and FEEL, get categorized in the same group as they both have a very high similarity, appear in the same album, and also have the same values for *artists* and *release_date*. Moreover, both tracks appear in all 3 datasets and hence create a group of size 6.

In order to account for such anomalies, we change the weights attributed to each field, assigning a higher weight to the track name, and increasing the *threshold* of the linear combination. This would ensure that different tracks belonging the same album are not rendered as the same entity.

3.5.2 Multiple Versions of the Same Entity

From 2, we observe the group distribution with group sizes of 4, 5 and 6. This could be a case explained by the occurrence of multiple versions of the same track in each individual data source, for example, ”Don’t Cry (Original)”, ”Don’t Cry (Deluxe Version)”, ”Don’t Cry (Live)”, ”Don’t Cry”. According to our understanding all of these renditions point to the same entity. A reason for such conflicting correspondences is possibly due to the fact that when creating the *track_name* comparator, we exclude strings present within the parenthesis. However, the comparator condition still specifies that remastered entities are rendered different.

Finding the proper balance is crucial because with fewer variables, predictions become skewed and heavily rely on the accuracy of those traits, which may not be completely accurate in real-world data. However, while adding more features can improve generalization, it also makes our variance fragile, as evidenced by the final matchers' slightly off score.

4 Data fusion

In this section we describe the different conflict resolution strategies we employed for removing inconsistencies in our final fused dataset based on the provenance information and metadata provided for our data sources [6, 4].

4.1 Dataset Provenance

For the purpose of data fusion all 3 song datasets are used. In setting the meta variables pertaining to data provenance, we set the highest priority (trust) to the Spotify dataset. This is due to the platform's continuous and ongoing updates, as one of the largest music streaming service provider, globally. Priority is then assigned to Deezer, followed by MusicOSet. Last updates are set as: { (*Spotify*: "2022-10-10"), (*MusicOSet*: "2020-05-20"), (*Deezer*: "2022-10-10") }, as the Spotify and Deezer scraped data is consolidated, using their respective APIs, while the MusicOSet data is readily available online.

4.2 Gold standard Preparation

We compile the Fusion Gold Standard by manually selecting 16 songs which are present in our datasets. There are 8 attributes which appear in at least two sources. 5 attributes, namely Track Name, Release Date, Artist Name, Album Type, Album Title are present in all 3 datasets. Song Duration and Tempo are present in Spotify and Deezer, and Album Genres is only available in Deezer and MusicOSet datasets.

We observe significant variation in Track Release Date and Album Genre. Track release dates vary due to Compilation albums, Remastered versions. In this case, gold standard results are validated through Google searches. Additionally, multiple editions are available for the same Album. For instance, "Deluxe", "Expanded" and "International" editions. We resolve this by selecting the original name of the Album release.

Attributes	Fusing Strategy
Album_Name_Fuser	Shortest String
Album_Type_Fuser	Favour Sources
Artists_Fuser	Union
Duration_Fuser	Max
Genres_Fuser	Union
Release_Date_Fuser	Earliest
Tempo_Fuser	Max
Track_Name_Fuser	ShortestString

Table 8: Fused attributes and their respective fusing strategy

4.3 Conflict Resolution Functions

We choose the fusing strategy for each attribute on basis of the underlying attribute type. In the case of String types, specifically *track_name* and *album_name*, we find the *ShortestString* strategy to be most effective. This helps overcome variations in both, Track and Album names. For instance album names such as ”Purpose” and ”Purpose (Deluxe)”. To fuse list attributes, we utilize the *Union* operator. This is relevant to the *artists* and *album_genres* attributes. The union operator provides coverage for the case of ”featuring artists” in *artists*. In case of genres, the Union operator covers genre sub-types, such as ”Metal”, ”Glam Metal”. Additionally, in order to resolve numeric types *duration* and *Tempo*, we choose the *Max* values and moreover, for the evaluation rule, add 5% tolerance to robustly resolve conflicts. In the case of song duration, Max strategy helps account for Radio Edit version of songs, where the song is shortened.

The field *release_date* is the most variant attribute in our datasets. We believe the most effective fusing strategy is picking the Earliest date. This stands to intuition, as there are instances of same song being present in different albums in the form of Compilations, Remastered and EP Albums. However, extracting the earliest date is not a complete solution. This is because we map `NULL` release date values to the date 1900-01-01. Therefore, we implement a custom *Earliest* function with added functionality to ignore the `NULL` date value (1900-01-01), and correctly pick the next earliest date. Lastly, we fuse Album Type by the *Favour Sources* strategy. We provide largest weight to Spotify due its global presence and dominance in the music streaming industry.

4.4 Evaluation Metrics

Table 9 and 10 shows the result of consistencies and densities of our fused dataset for the common attributes. It should be noted we present values merely for the common attributes. This is because conflict resolution performance can only be evaluated for these attributes only. With the aforementioned fusing strategies we achieve an accuracy of **0.81**.

Attribute	**Consistency**	**Attribute**	**Consistency**
Album_Genres	0.07	Duration	0.99
Album_Name	0.91	Release_Date	0.76
Album_Type	0.93	Tempo	0.86
Artists	0.98	Track_Name	0.97

Table 9: Consistency for fused attributes, which are available in atleast 2 datasets

Dataset	**Spotify**	**Deezer**	**MusicOSet**	**Fused Data**
Album_Name	1	1	1	1
Album_Type	1	0.856	1	1
Album_Genres	0	0.856	0.970	0.969
Artists	1	1	1	1
Duration	1	1	0	1
Release_Date	0.990	0.998	0.999	1
Tempo	0.997	1	0	1
Track_Name	1	1	1	1
Average Density of Datasets	0.873	0.964	0.746	0.996

Table 10: Initial (per Dataset) and Fused dataset density for common attributes in atleast 2 datasets

4.5 Error Analysis & Limitations

We manually inspect the conflict resolution performance through comparison between *fuxed.xml* and the Gold standard *Gold_Standard_Data_Fusion*. Firstly, errors are made as the true value in the datasets is simply not present. For instance, the song "If we ever meet again" is only present in form of "If we ever meet again (Expanded Edition)" in all Datasets. We witness the same case with Genres. However with Album Genres the high error rate is also due (1) its low consistency and (2)

the inexhaustive genre type values.

Additionally,we observe error in track *release date*. The error is attributed to our `NULL` value handling method. Tracks with only Year present, the date was mapped to the first day of the year. Therefore, picking the Earliest date, pick the first day of that year.

5 Conclusion

We aim to merge music data from leading music streaming sources across the web. These are **Spotify, MusicOSet,** and **Deezer**. We perform schema integration, identity resolution through a set of matchers. Our best combination includes 4 attributes (Track_name, Album_name, Release_date and Artists) with weights (0.4, 0.2, 0.2, 0.2) respectively for the Spotify-MusicOSet and Deezer-MusicOSet. For Spotify-Deezer our best combination consists of (Track_anomaly, Album_name, Release_year, Duration and Artists_name) with weights (0.2, 0.2, 0.2, 0.2, 0.2). We observe our model has limited capabilities in distinguishing short Track names belonging in same albums. Lastly, we define fusers on common attributes. In reference to our Fusion Gold Standard we achieve accuracy of 0.81. Post fusing we see errors, as there were instances of the true values not belong present across the our datasets. In conclusion, an accuracy of 0.81 covers large variety of standard and corner cases. This process results in 9451 entities, with 700 additional entities to the Spotify datasets. Finally we achieve 15% increase in average density.

References

[1] Deezer API. https://api.deezer.com/track/.

[2] Musicoset. https://marianaossilva.github.io/DSW2019/.

[3] Spotify API. https://developer.spotify.com/documentation/web-api/.

[4] Oliver Lehmberg and Christian Bizer. Stitching web tables for improving matching quality. *Proceedings of the VLDB Endowment*, 10(11):1502–1513, 2017.

[5] Oliver Lehmberg, Christian Bizer, and Alexander Brinkmann. Winte. r-a web data integration framework. In *International Semantic Web Conference (Posters, Demos & Industry Tracks)*, 2017.

[6] Oliver Lehmberg, Dominique Ritze, Robert Meusel, and Christian Bizer. A large public corpus of web tables containing time and context metadata. In *Proceedings of the 25th International Conference Companion on World Wide Web*, pages 75–76, 2016.

[7] Dominique Ritze, Oliver Lehmberg, and Christian Bizer. Matching html tables to dbpedia. In *Proceedings of the 5th International Conference on Web Intelligence, Mining and Semantics*, pages 1–6, 2015.