Capstone Project: TADA: TAbular Data Annotation using semantic web

Ahmad Alobaid

August 24, 2017

Content

1 Definition **2**
1.1 Project Overview 2
1.2 Problem Statement 2
1.3 Metrics 3

2 Analysis **3**
2.1 Data Exploration 3
2.2 Exploratory Visualization 4
2.3 Algorithms and Techniques 5
2.4 Benchmark 6

3 Methodology **6**
3.1 Data Preprocessing 6
3.2 Implementation 7
3.3 Refinement 7

4 Results **8**

5 Conclusion **9**
5.1 Free-Form Visualization 9
5.2 Reflection 10
5.3 Improvement 11

1 Definition

1.1 Project Overview

In this century, people are storing and gather data more than ever in human history. We have a massive amount of data scattered in different places. Cafarella et al. surveyed web pages in Google Crawl (where pages are inspected) and found around 154M tabular data [2].

To utilize the huge amount of data that are increasing every day, we need to understand the context to each collection of data. With that, we can link data from different sources from different domains together. This would enable us to answer complex questions we couldn't answer before. We want to learn what a collection of data represents. Given a set of input files, we can deduce what a particular set of data represents, in other words, learn the *semantic types*.

The learning process needs data sources (training set) to use it for the learning process. For that, we rely on the *Semantic Web*, which is an extension of the current Web where information is given a well-defined meaning[11]. This kind of information is often stored as graphs in SPARQL endpoints.

In this project, we learn the semantic types of a collection of tabular data in the domain of Olympic Games using DBpedia as the training set. We focus on the classification of numerical columns in the input tables. We use a clustering technique called *Fuzzy c-means* which is invented by James Bezdek [1]. We apply his fuzzy clustering technique with some changes in the way we apply it. This algorithm is not implemented in scikit-learn (formally known as sklearn), so we have to implement this algorithm. We studied the implementation of *k-means* in scikit-learn and followed similar design decisions (e.g. the use of *fit* method).

1.2 Problem Statement

It is common for tables to have missing headers or headers that can't be used as mentioned in [2] (e.g. if a column has the header "from", we don't know if it is a time (a bus timetable) or a location (manufacturing countries)). Hence, we will be relying on the data in the column regardless of the header. In the semantic web community, people use RDF (Resource Description Framework) and RDF Schema (RDFS) which are W3C recommendations. RDF is a framework to represent information on the Web while RDFS is a data-modeling vocabulary for RDF data [12].

In our work, we automatically identify/learn the types of the numerical columns of the data sources (e.g. CSV files) with the types being the one used in the semantic web (e.g. from DBpedia[8]). Here we are not referring to the primitive types (e.g. integer, float, ..., etc), instead, are referring to the *semantic types* (e.g. a height of a person). We will use a technique called *fuzzy c-means*, which is a clustering technique that provides the most probable *semantic types* for each column in the input files, ordered from the most probable to the least probable ones. The training set is the extracted data from the English

DBpedia[1], and the testing set is a set of CSV files (not extracted from DBpedia).

1.3 Metrics

We use *precision* as the metric for measuring the performance of our application of *fuzzy c-means*. First, we manually annotate each numerical column in the CSV files with a *semantic type* from DBpedia. Next, we apply our approach on these files with the model being trained on the data extracted from DBpedia. For each column, each value is classified separately (which produce a membership vector). After that, the membership vectors are averaged for all the values in that column resulting in a single vector. The score is calculated for $k = 1$, $k = 3$ and $k = 5$; k represents the number of top candidates. A classification of a column is considered true if the manual *semantic type* is in the top k candidates (which have the highest membership scores in the vector). Finally, the precision is computed for each k.

We choose *precision* to measure how many of the classified columns are matched correctly. We are interested in how many of the classified columns are classified correctly (match our manual annotation). We compute the *precision* by dividing the *true positives* by the *true positives* + *false positives. True positives* are columns that are classified correctly, and *false positives* are the columns that are misclassified. It doesn't make sense to use *recall* because we are comparing against manually selected columns and it will always be 100%. Columns that are not classified and should not be classified are irrelevant to us (*true negatives*). Also, the *false negatives* are irrelevant to us, it is always zero, which makes *recall* 1.

2 Analysis

2.1 Data Exploration

There are two sources of data we are using in our application, training set, and testing set. For our training set, we use a subset of the data in the English DBpedia[2]. The English DBpedia includes data about several domains; we only focus on the ones related to the domain of Olympic Games. Our testing set is a collection of CSV files related to the domain of Olympic Games. The input files are composed of numerical columns and non-numerical columns. Around 27% of the input files are numeric on average. The average number of rows per input file is around 97 row per file. We found missing values in some of the columns that we eliminate in the preprocessing step. We also noticed that some numerical data are represented in a format that can not be consumed right away and need to be transformed (e.g. feet and inches include single and double quotes). For outliers detection, we use Tukey's test and found around eight columns with outliers, ranges from one to four outliers per column. Comparing that to the

[1]DBpedia is a public domain SPARQL endpoint `http://dbpedia.org/sparql`
[2]`http://dbpedia.org/sparql`

average number of values per column which is 97 means that around two values out of the 97 (in the eight columns that have the outliers) which is less than 2%. A sample of the data is the heights of badminton players [173, 166, 164, 177, 181].

2.2 Exploratory Visualization

In the below graphs, the X-axis represents the values in the columns of the input files, and the Y-axis represents the class and column header. We show two diagrams, one with all the columns and values, and the other one is a magnified view to show the differences of the values more clearly. The outliers are denoted in the diagrams with the x sign. We can see that they look far from their corresponding cluster. The other data points are shown in the diagram as circles. Here we can see how data points of the same column are closely located. For example, there is a clear separation between the *height* cluster and the *weight* cluster for *handball players*. On the other hand, we can see how the values of the events column of the *olympics* file smear all over the horizontal line; this aspect is discussed in section 4.

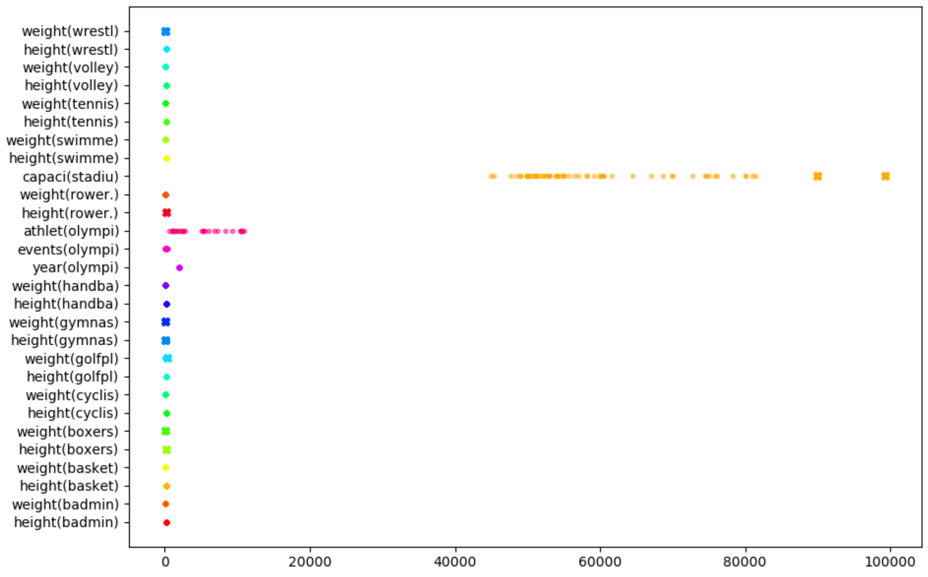

Figure 1: The data points of the input files with outliers

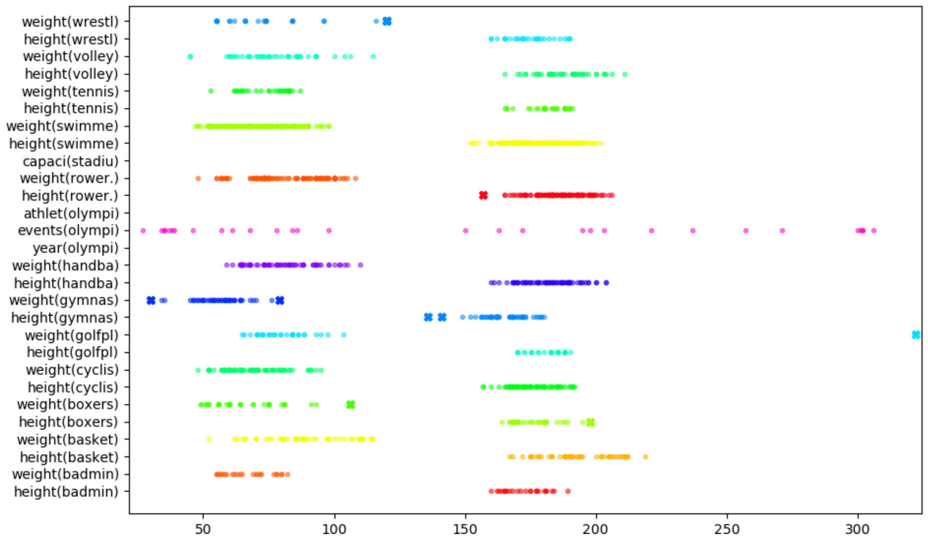

Figure 2: The data points of the input files with outliers (magnified)

2.3 Algorithms and Techniques

We adopt *fuzzy c-means* clustering technique, which is a generalization of *k-means* [1]. The algorithm expects the url of the endpoint, the relevant classes, and the input files. The url is the address of the SPARQL endpoint(e.g. `http://dbpedia.org/sparql`), where the queries will hit to extract the data. The classes are concepts that are related to the input files (e.g. `http://dbpedia.org/ontology/BadmintonPlayer`). The input files are the test set to be classified. The algorithm outputs for each numerical column (in the input files) the list of most probable labels.

The algorithm starts with the **data extraction** method, which queries the given endpoint. It queries the properties (attributes) for each given class (e.g. for the *BadmintonPlayer* it will return *height*, *weight*, *activestartyear*, *highestrank*, ... ,*etc.*). Then, for each $< class, property >$ pair, it will return the corresponding list of values (e.g. for $< BadmintonPlayer, height >$ it will return the list [165, 170, 173, 174, 176][3]). Since not all the properties are numerical (e.g. *name*), the algorithm performs a check for every property, if most of its values are numeric, then this property is kept and non-numeric values are discarded (if any). Otherwise, the property will be discarded. The output of this method is a list of numerical columns.

Next is the **training** method. The values of each column will form a cluster. This is done by setting what is called *membership matrix*. In fuzzy clustering, a data point can belong to multiple clusters, each with a belonging score. These

[3]typically, for each property it returns hundreds of values

belonging scores are organized in a vector referred to as *membership vector*. The sum of the belonging scores for any data points should be 1. So the higher the value is, the more belonging that point has to the corresponding cluster. The *membership vectors* for each data point are gathered into a matrix referred to as the *membership matrix*. Using the data points and the *membership matrix*, the centroids for each cluster is computed[4].

The last part of the algorithm is the **classification** method. For each column in the input files, each data point is classified separately, which produces a *membership vector* for each data point. For each column, the mean of the *membership vector* for all its data points is aggregated into one vector. The cluster corresponding to the highest score is the most probable *semantic type* for that column.

We use this approach because it provides multiple candidates ordered by most probable *semantic type* (label). This is beneficial for this project because if a column is misclassified by the application, the user can choose the second most probable label rather than looking into all possible labels and choose the correct one. The technique also performs well without exact matching (e.g. if the wrestlers in the training set are completely different that the wrestlers in the testing set, it will still classify it correctly given the values of the attributes reside within in the same range).

2.4 Benchmark

Before running our application, we manually annotated the input files with *semantic types*. We searched for the concepts in DBpedia using loupe[10]. After that, we went to the DBpedia endpoint[5] and looked for the properties for the classes we found with loupe. We compare the score of our application (the *precision*) against randomly labeling the columns of the input correctly[6]. We calculate the probability for each attribute to match the correct *semantic type* separately which turns out to be 2.39% on average.

3 Methodology

3.1 Data Preprocessing

Data are not always in the way we want it and often have missing values or in a format that is not easy to consume. In our application, we focus on numerical columns. Having missing values, the application will consider columns with missing values as non-numerical column and hence will not be classified. To solve this problem, we removed any row that has missing values in the numerical columns and since they are only few, they won't negatively affect the classification outcome. The reason is that missing values per column does not

[4]using *fuzzy c-means* centroid computation formula[1]

[5]`http://dbpedia.org/sparql`

[6]we discussed in section 1.3 the reason for choosing the precision

exceed 3 and the average number of rows per column is around 97. There are also some columns that are in feet and inches (e.g. 6'11") which are considered a non-numerical and is discarded accordingly. We solved that by converting the values to centimeters so it can be consumed by the application.

3.2 Implementation

This algorithm despite being around for more than twenty years, it is still not implemented in scikit-learn. It was a good experience implementing the algorithm from scratch relying on the research paper by *James Bezdek* where *fuzzy c-means* were introduced to the research community as an application of *fuzzy set theory* which was introduced by *Lotfi Zadeh* and *Dieter Klaua*. As this algorithm is a generalization of *k-means*, I went to its source code in scikit-learn and developed *fuzzy c-means* in a similar way with similar *methods* (e.g. fit) and *class variables* (e.g. $cluster\_centers\_$)[7]. There were some complications that occurred during the development. One of the complication is the division by zero which we needed to handle. Another complication which also messed up the calculations is having *nans*. Missing values in the input are then handled be removing all the *nans*. Expecting a number and getting a string from the training set is something we solved by discarding non-numerical values when numerical values are expected.

We implemented the *fuzzy c-means* as an independent standalone module. It can be used with other application as well without any change. The application is developed as a web app using *Django* framework[8]. So the model can be created with the specified concepts and the url of the SPARQL endpoint via the web interface. For extracting the data from the SPARQL endpoint, we used *SPARQLWrapper*[9]. Then, for the prediction step, the input files can be uploaded to the application to be labeled with the *semantic types* from the provided endpoint. The classification of each column is shown as a set of top candidate *semantic types* ordered in descending order from the most probable ones to the least probable.

3.3 Refinement

In the initial implementation, the data is extracted from the endpoint is used as is with the assumption that no textual data would exist in numerical properties. However, in the endpoint, we found several textual values in numerical properties (e.g. the value "high" instead of an actual number). A check is added that if more than the half of the values in a given bag of numbers (for a property) are numbers, then, non-numeric values will be discarded. Another refinement that we included is to accept CSV files without headers. This is possible as the application relies solely on the data itself.

[7]We only implemented the necessary methods for the application
[8]https://www.djangoproject.com/
[9]https://github.com/rdflib/sparqlwrapper

4 Results

The model we chose has the advantage of suggesting not only the most likely label but also the second, third, ..., etc. This enables picking the second, third, and any kth top candidate in case the top suggested label is incorrect. Looking at the scores below, for $k = 1$ we got a precision of 73% which is way higher than randomly labeling a column correctly (2.39% on average). Nevertheless, it is important to investigate why the algorithm mislabeled some of the columns in the input files. Looking deeply into the misclassified columns, we see that one of the misclassified properties are related to the year of an event. It is understandable to misclassify the year given that years for an event, in general, is not distinguishable from years for another event. An exception of that is if the years for one event is older than another event (e.g. the birth years of the emperors of the Roman Empire and the years were Nobel prices have been awarded in the Literature domain). The emperors birth years range from before the Birth of Jesus (B.C) until the birth year of *Palaiologos* on 1449 A.D. while the first Nobel price in Literature was awarded to Sully Prudhomme in 1901, so classifying data points of those two won't be mistaken for sure. However, if we are working on years for two different events with overlapping years, then there is a higher probability of getting the classification wrong depending on the training and testing data sets. Another type of misclassified columns is count-based type were columns of such a nature report number of participants, population, ... etc. This kind of data do not have a range where most of them reside within, and some actually increase over time (e.g. population). The scores in the below table align with that as well as the scores get higher when such properties are eliminated the from the scores (see the 3rd column in the below table).

The application solves the problem introduced earlier for annotating the columns in the input files with the correct *semantic types* from the endpoint. It does not only succeed in that, but it is also resistant to extreme values in the input files. This is because the membership values are averaged and that the values of the membership are normalized that it is between 0 and 1, so even if an extreme value is introduced, it will not skew the classification of the column even without outliers removal. Maybe I should include the removal of outliers in the training set in the code itself

Top k-candidates	Score	Score specific range	Score unspecific range
1	73 %	85%	0 %
3	86 %	100%	0 %
5	86 %	100%	0 %

Table 1: Classification score of annotating the columns of the input files with semantic types

5 Conclusion

5.1 Free-Form Visualization

In the below figures we depicted the points with the means, once with the outliers and another without the outliers (each with a different marker). Here we are referring the outliers detected using Tukey's test. The X-axis represents the data points, and the Y-axis represents the column headers and the file names. We can see in the figures that the difference between the two means is relatively small. The outliers in the input files are very few and not very far to affect the means so much to cause the columns to be misclassified.

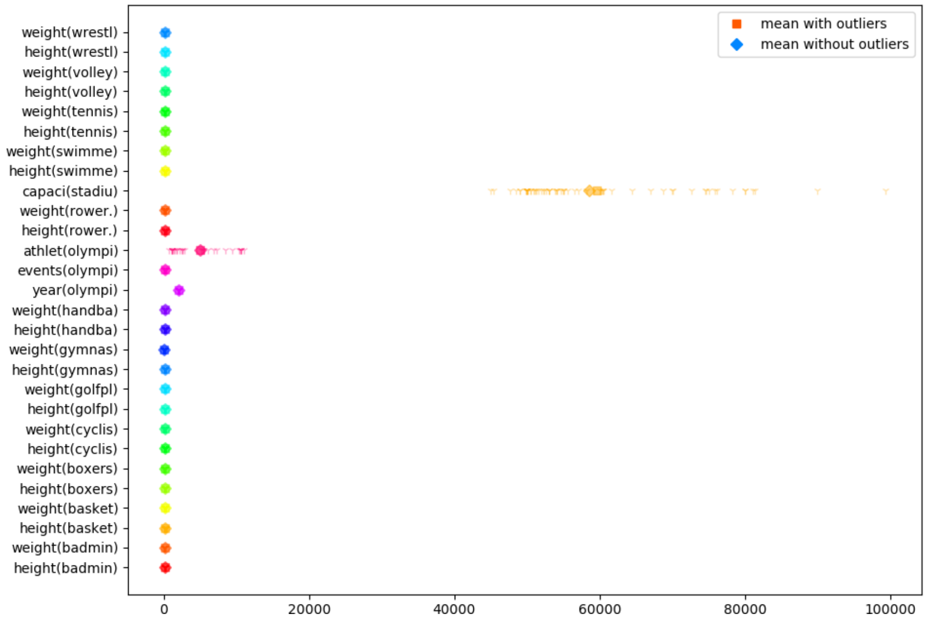

Figure 3: The means with and without outliers

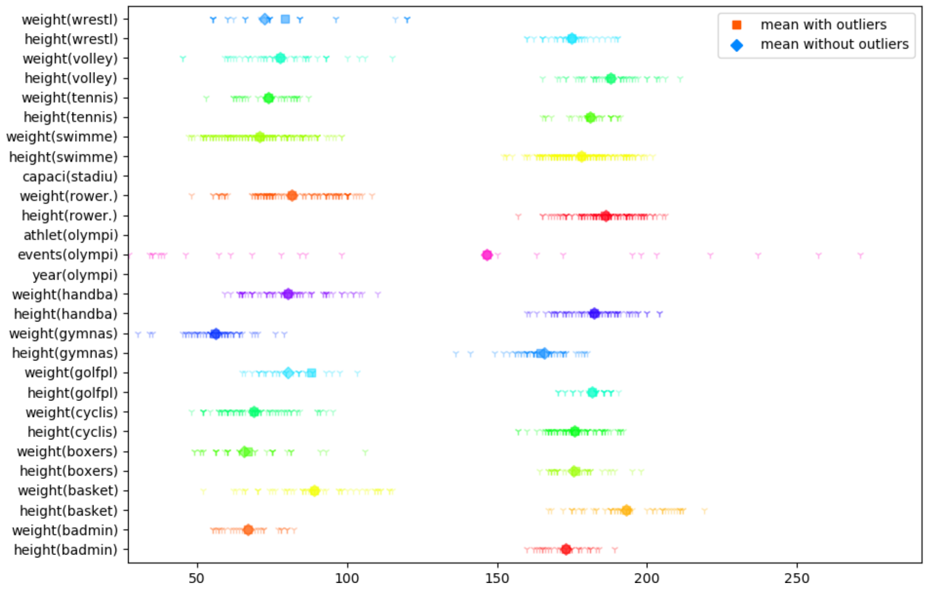

Figure 4: The means with and without outliers(magnified)

5.2 Reflection

We summarize the process of automatically annotating the columns of input tables with *semantic types* extracted from a SPARQL endpoint. First, the data are extracted from the English DBpedia. Given a set of classes as input, numerical properties are picked. This is done by performing a majority type check, if most of the values for a given property are numeric, then this property is considered numeric and non-numeric values are discarded; if not, the property is discarded. Next, the training process starts by calculating the centroids. This is done by initializing the *membership* matrix with zeros and ones that reflect the belonging in the training set and then computing the center of each cluster reducing the total error. After that, the input files are read, and non-numeric columns are filtered out. For each column, the data points are classified, and then the whole column is classified by calculating the mean of the membership for each data point.

An interesting aspect of the project beside implementing *fuzzy c-means* is how accurate the results are with minimal information used from the training set. In our application, we do not consider the relationship between columns and do not perform entity detection. We rely solely on the data itself, not even taking into account the headers of the tables (if exists). One of the most difficult tasks was to actually understand the research paper where *fuzzy c-means* were introduced. Also implementing the algorithm was not easy especially the concept of calculating the centroids from the membership with fuzzy belongings scores to reduce the total error. This application is also applicable to other domains as well. Getting a high score of labeling the columns of input files depends on the intersection of ranges in the training set. The lower the intersection of ranges, the higher the classification score would be.

5.3 Improvement

There are further improvements that can be used to increase the accuracy, to check if the exact numbers actually exist (exact match) and consider that as an extra feature[3]. Performing entity recognition can also improve the classification were columns can only be matched to properties related to the recognized entity class[4][5][6]. Regarding the benchmark, we believe that this benchmark is not comparable to other benchmarks if used with different data. The reason is that this algorithm is highly sensitive to the nature of the data sets used. The more the intersection of the ranges of the columns exists in the data the lower the accuracy would be as explained in the results section. A practical benefit would be to generate R2RML mappings from the classification[7]. The R2RML mappings can be used to query the input and can ask complex questions using SPARQL[10].

References

[1] Bezdek, James C., Robert Ehrlich, and William Full. "FCM: The fuzzy c-means clustering algorithm." Computers & Geosciences 10.2–3 (1984): 191–203.

[2] Cafarella, Michael J., et al. "Webtables: exploring the power of tables on the web." Proceedings of the VLDB Endowment 1.1 (2008): 538-549.

[3] Zhang, Meihui, and Kaushik Chakrabarti. "Infogather+: Semantic matching and annotation of numeric and time-varying attributes in web tables." Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data. ACM, 2013.

[4] Limaye, Girija, Sunita Sarawagi, and Soumen Chakrabarti. "Annotating and searching web tables using entities, types and relationships." Proceedings of the VLDB Endowment 3.1-2 (2010): 1338-1347.

[5] Syed, Zareen, et al. "Exploiting a web of semantic data for interpreting tables." Proceedings of the Second Web Science Conference. Vol. 5. 2010.

[6] Ritze, Dominique, Oliver Lehmberg, and Christian Bizer. "Matching html tables to dbpedia." Proceedings of the 5th International Conference on Web Intelligence, Mining and Semantics. ACM, 2015.

[10]https://github.com/oeg-upm/morph-rdb

[7] de Medeiros, Luciano Frontino, Freddy Priyatna, and Oscar Corcho. "MIRROR: Automatic R2RML mapping generation from relational databases." International Conference on Web Engineering. Springer, Cham, 2015.

[8] http://wiki.dbpedia.org/

[9] https://www.w3.org/wiki/SPARQL

[10] http://loupe.linkeddata.es/

[11] https://www.w3.org/RDF/Metalog/docs/sw-easy

[12] https://www.w3.org/TR/2004/REC-rdf-concepts-20040210/