City, University of London in MSc Data Science

PROJECT REPORT

2021

Embedded Driven Semantic Table Understanding

Zacharias Detorakis (zacharias.detorakis@city.ac.uk)

Supervised by: Ernesto Jiménez-Ruiz (ernesto.jimenez-ruiz@city.ac.uk)

By submitting this work, I declare that this work is entirely my own except those parts duly identified and referenced in my submission. It complies with any specified word limits and the requirements and regulations detailed in the assessment instructions and any other relevant programme and module documentation. In submitting this work, I acknowledge that I have read and understood the regulations and code regarding academic misconduct, including that relating to plagiarism, as specified in the Programme Handbook. I also acknowledge that this work will be subject to a variety of checks for academic misconduct.

Signed:

To my mum who's no longer able to see my progress in life but whose values will always be a shinning beacon for me to follow…

Abstract

This project aims to enhance a set of data given as input (e.g. tabular data) with semantic meaning using existing Knowledge Graphs (KG) as reference. The approach that has been implemented is inspired by ColNet, a system implemented as part of the SemTab challenge, and comprises a series of modules for i) parsing the tabular data ii) using the (cell) values to identify candidate entities with lexical similarity and their candidate KG classes iii) training a set of binary cnn classifiers (one per class). Finally the system is employing different ways of predicting the class of each column using the above and subsequently suggests an entity for specific cell values. A pretrained word2vec model was used to train the cnns and transform the column values to inputs for predication.

Keywords: Knowledge Graphs, Convolutional Neural Networks, Class, Entity, Word2Vec

Table of Contents

1 Introduction and Objectives

1.1 Background of the problem

One of the main problems of computer science is the ability to represent human knowledge and model the world in a form that can be understood and processed by computers. Once this problem is resolved, models could then be subsequently trained to perform complex reasoning and draw conclusions and new knowledge in an intelligent way that resembles human intelligence.

The World Wide Web Consortium (W3C), has developed two languages of knowledge representation for the Internet: RDF(S) and OWL. These languages form the syntax of knowledge bases KB or Knowledge Graphs (KG). Simply put, a knowledge graph provides a structured representation of information using a directed vector of the form (subject, predicate, object) according to the Resource Description Framework (RDF). Each node in that triple (i.e. subject and object) represent an entity belonging to a class and the edge between them (i.e. the predicate) represents the relationship between the classes of the edges. Several knowledge graphs have been developed by:

- domain experts that have transferred their specialized knowledge into these ontologies and
- knowledge engineers, that have translated this knowledge into a standard language such as OWL.

In most cases, however, for reasons of convenience, simpler forms of knowledge representation are chosen, such as plain text in natural language (unstructured information). Such cases of unstructured or semi-structured information are text files (i.e. txt files, excel files, html files) posted on the internet, information entered as text in database systems, etc. In this (unstructured) form, knowledge is not comprehensible to computers and cannot be used in its entirety to draw useful insights.

In order to solve this problem, the scientific community has turned its attention to areas such as natural language processing and the automatic extraction of terms. With the tools built into these areas, we try to automatically extract knowledge from unstructured language descriptions and use it to map to existing ontologies and even expand them, creating new specialized classes or entities. The exported knowledge can then be used to classify objects of our world into categories (classes) of these well-established knowledge bases.

1.2 Reasons of the choice of the project and beneficiaries

The main contribution of this project it to create an end-to-end pipeline with that can assist with type and entity identification of tabular data. By creating individual modules for parsing tabular data, identifying candidate entities from knowledge graphs and having multiple mechanisms of predicting column types this system can be used as an extensible framework to swap in/out different types of input data, knowledge bases and classification models.

The main components created as part of this project assume input tabular data in a csv format and use dbpedia KB as the reference for identifying entities and types. The pipeline implements a simple voting algorithm of the identified objects, a more elaborate approach inspired by TF-IDF used in information retrieval as well as a more sophisticated set of CNN classifiers (ColNet) for type identification. Finally, the predicted column types are utilised to enhance the entity annotation of the cell values in the tabular data.

Pipelines like the one implemented by this project, can be used in information retrieval from large unstructured data that could subsequently inform the analysis of information maintained by online applications or other systems that generate data in the internet of things. Although the KG often suffer from knowledge gaps (i.e. not all entities of a given class exist as instances of that class) suggesting even a type for a given column could also be a useful first step in analysis to limit the universe of the exploratory data analysis EDA an analyst may need to do. Finally, cell values that have failed to be annotated with an entity by the pipeline can then be fed back to the domain experts and knowledge engineers to enhance the KB with the potentially missing entities.

1.3 Objectives of the project

A set of objectives were set out, in order to answer the research question on how to enhance a set of data given as input (e.g. tabular data) with semantic meaning as follows:

- Column-Type Annotation (CTA): Assign a class from the KG to an entire column of the table
- Cell-Entity Annotations (CEA): Assign an individual entity of the KG to each specific cell

This will be done using existing knowledge graphs (e.g. DBpedia, WikiData) as reference by considering the following tasks:

- Retrieve candidate entities using the provided lookup service APIs as well as custom SPARQL queries
- Examine the effect of using the KG hierarchy “rdfs:subClassOf” to filter down candidate classes
- Use the retrieved entities to predict the correct type for the target columns in the tabular data
- Replicate the code given by ColNet and use that as a baseline to:
    - train individual ccn classifiers for each candidate class and
    - use them to predict the type of the target columns
- Similar to the type prediction, implement a pipeline to identify the entity that each cell value corresponds to
- Use the predicted type of the columns to enhance the performance of entity identification

1.4 Methods used

As mentioned in previous sections, the end-to-end pipeline comprises multiple modules with catch points in between in order to be able to plug different implementations in and out. Figure 1 illustrates high level the steps of the proposed method.

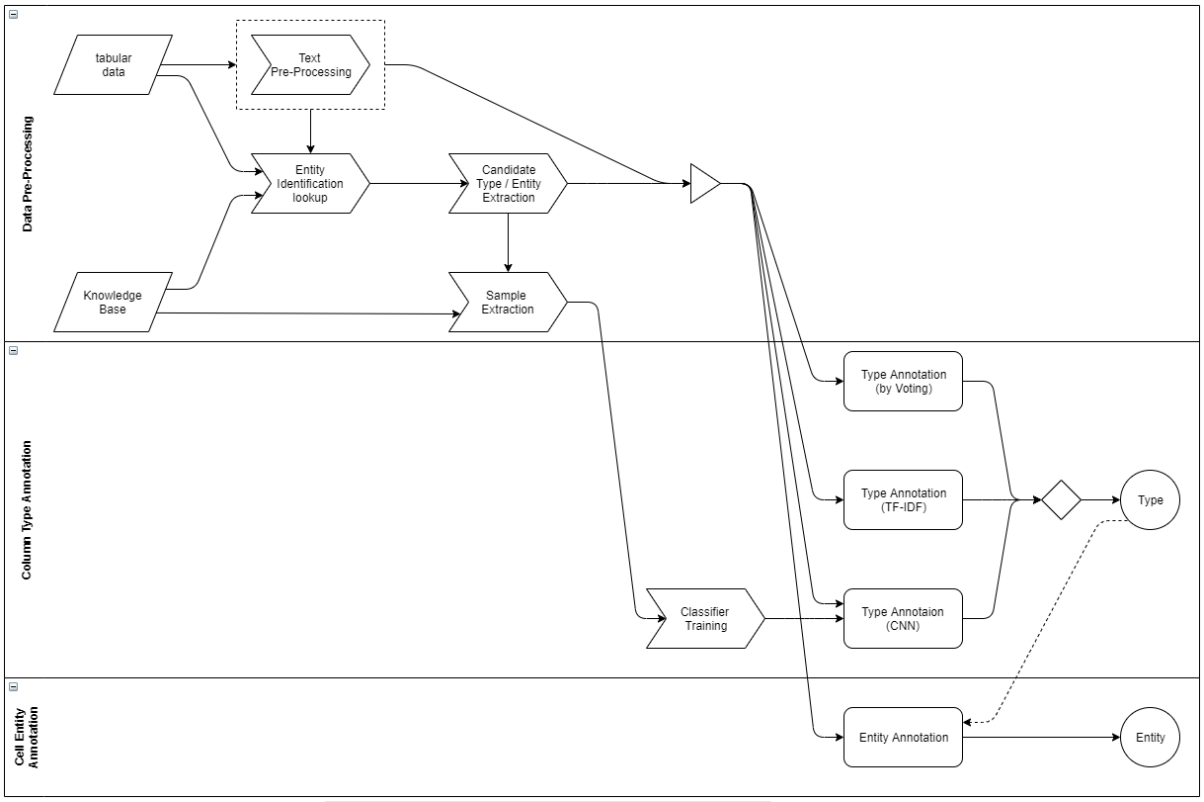

Figure 1. Process of predicting types and entities for tabular data from a reference KG

Parsing: Reading tabular data and storing them into a json object that can then be used by the next step

Lookup (entity + class): The lookup step is the module that queries the reference knowledge base with the cell values to retrieve candidate entities. The knowledge based used for this component in the current project is DBpedia and there are two types of lookups:

- The entity lookup using the DBpedia lookup endpoint and the
- SPARQL endpoint that retrieved the classes of the identified entities in case the lookup URL failed to retrieve them

The lookup URL gives the ability to limit the number of retrieved results, since it performs fuzzy matching based on the input string (i.e. cell value). The limit was kept quite flexible to allow for more results to be retrieved however further downstream we imposed stricter limits to narrow down the candidate classes considered in voting and TF-IDF as well as the number of classifiers that are assessed as candidates for each column.

Given that the dbpedia lookup URL is doing a fuzzy matching on the query string, the 'text pre-processing' step in Figure 1 was skipped as part of this iteration.

Sample extraction: this step is used in order to get training data for the candidate rdf:type classifiers. This is a similar step to the above when we extracted the class from the entity but here the reverse path is followed. Thereby, entities are extracted, at random, from a given class.

Classifier Training: in this step a binary cnn model is created for every candidate type (more details on this in Chapter 0) and trained with positive and negative samples. The positive samples are taken from the sample extraction of the previous step whereas the negative samples are taken from candidate entities that DO NOT belong to the current class the model is trained for.

Predict Type: In this step all previous processing is brought together to predict a type for every target column in the input tabular data. Several approaches to predict the type of a column have been implemented:

- Voting: This approach simply selects the type with the most votes based on the simple lookup
- TF-IDF: In this approach the candidate types are scored using a TF-IDF formula and sorted based on the results
- CNN: This approach ranks the candidate types based on the score of the relevant binary classifiers for the given column in the input data

Predict Entity: This step annotates the cell values with an entity based on the lookup results and optionally using the predicted classes from the previous step to enhance the precision

1.5 Work plan

To achieve the solution of the components listed above, the work was divided in individual milestones dedicated to the predefined modules.

After completing the initial literature review at the early stages of the project to better understand the background and related work the next task was to design the text processing feature and integration with the reference knowledge base. As part of that design, several json structures were structured to save the data for subsequent runs and avoid repeating the lookup step (which is rather time consuming and requires to be online) unless new input data or reference knowledge base were considered. The catch-up jsons files expedited testing as the lookup step takes a lot of time to complete and is deterministic i.e. always yields the same results for the same (input, knowledge base) combination. Implementation of this step followed shortly after using Python (jupyter notebooks for debugging at this stage).

Getting candidate classes and entities at this point enabled us to get the first type identification experiments in place using just a majority vote of candidate classes and the tf-idf logic

The next major step following on from this was the design and implementation of the binary cnn classifiers. This was yet again another computational and time expensive step so the solution was designed in a way that can save and load models. These models can be trained offline for any number of classes from the reference knowledge base (which is the time consuming part) and readily used for the type annotation task as and when needed.

With the classifiers implemented the last design step for the CTA end-to-end pipeline was to come up with a way of using the prediction results to finally decide on a column class and perform experiment to identify which hyperparameter combinations would work best in the given project setup.

In parallel to the above, we started completing the relevant sections of the report as soon as e.g. context or intermediate experimental results because available for reporting and discussion. Thereby, the report was prepared alongside the design and implementation to capture the details as they were worked on.

Finally, after a series of experiments highlighted the best approach for CTA the pipeline for the CEA was designed and implemented. This pipeline reused a lot of element from the previous one (i.e. the candidate entities, the predicted classes) and proposed a few different ways of entity identification optionally using as additional input the best results of the CTA pipeline.

1.6 Major changes of the goals or methods that happened during the project

TBC

1.7 Report outline

This first chapter of the report provides a brief background of the area that the project is placed in and an introduction of the problem that the project is trying to resolve. The inspiration for this project is also stated along with a proposed methodology as a series of high-level steps of the proposed implementation. Moreover, the objectives are set in terms of what the project is aiming to achieve along with a plan on how these objectives will be achieved within the timeframe that has been allocated to this project.

The next chapter (Chapter 2) follows on from the introduction to provide a more deep dive view of the theorical background the project was built up on, including detailed literature review. The two main sections in that chapter revolve around the theory of knowledge graphs and the convolutional neural networks.

After covering the theory supporting this analysis, the next chapter (Chapter 3) focuses on the methods that were applied during the implementation of this research project. Details about each of the end-to-end pipelines' steps mentioned in paragraph 1.4 as well as key decisions taken for the execution of several experiments will be discussed in detail. Therefore, this chapter will set the framework on which the remaining chapters of this report will expand with presenting the results.

Following on from Chapter 3, the next chapter (Chapter 4) presents the experimental results of the implemented methods for the cases that were designed. These experimental results will highlight strengths and weaknesses on the approach and will inform the next steps for future analysis.

In Chapter 5 we will pick up the critical discussion around the results of the previous chapter to analyse further the efficiency of the chosen models and get potential ideas for future improvements.

Finally, Chapter 6 presents a platform for Reflection, and at the same time concludes this project while offering a general evaluation of its results. Any potential for future work will also be presented in this section.
(2,200_2,650)

2 Context

This chapter will present the theory of the Semantic Web and Knowledge Graphs that is essential for the understanding of the task at hand as well as related work that has been proposed and/or implemented to semantically enhance information in tabular data. By examining existing work, published papers and best practices regarding column and entity annotation, this section aims to evaluate how existing systems approach the problem undertaken by this project, and hence, analyse and present the background knowledge which formed the basis of the work described in this thesis. There is also theorical background on convolutional neural networks and how they've been used in respect to the semantic enhancement of tabular data as that will also form part of the design and implementation of this project.

2.1 Semantic Web

Semantic web, also known as Web3.0, is an extension of the current Web which offers a smart web service that structures data in a clear and disciplined way making it easily accessible by machine and humans alike. The data is interconnected through ontologies which can be shared and processed by automated tools as well as manually inspected so that computers and people can work together.

The Semantic Web network comprises a multilayer architecture (Berners-Lee, 2000) as shown in Figure 1

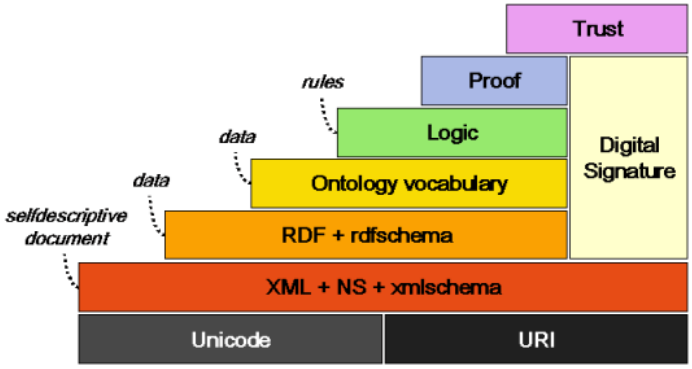

Figure 2. Semantic web layers

- The Uniform Resource Identifier (URI) layer is responsible for encoding the resources by defining an identifier that uniquely represents each resource in a uniform way. The URI makes it possible to gather (and share) all the information for a given resource (i.e. types, attributes, instances) even if they have been generated by separate resources. The URI is an umbrella term that encloses two other (perhaps more familiar) terms:
    - The Uniform Resource Locator (URL) and
    - The Uniform Resource Name (URN)

- The XML layer deals with the separation of the data content, its format and its linguistic meaning, as well as the data representation using a common formatting language. XML uses metadata to describe the type and format of the actual data and is considered the basis for organizing data on the internet, in terms of form (NOT semantics). This layer helps describe the data so that it can be understood by the higher layers.
- Moving up the stack, the Resource Description Framework (RDF) and RDF Schema (Yong-gui and Zhen, 2010) defines the semantic information of the data and it's type. As the term suggests, it provides a standard framework for describing statements about resources and their properties. Combined with the XML layer, this layer creates the basic data formatting language layer and enhances the metadata that will be used by the upper layers.
- The Ontology Vocabulary aims to reveal the semantics, defining common knowledge and semantic relationships between different types of information, thus creating a network of meaning. Ontologies are useful to represent objects and the relationships between them, to understand their meaning by machines but also to facilitate the exchange of information.
- The Logic layer is responsible for providing axioms and inference principles that will be used by other intelligent services that will process the data of the semantic web. It is used as a framework for drawing assertions, new conclusions and facilitates how these inferences should be expressed for the implementation of the Semantic Web (Janjua and Hussain, 2010).
- Finally, the last two layers provide a mechanism to ensure the validity of the statements made at the source and that this source can be trustworthy (Medić and Golubović, 2010). They enrich network security, using encryption and digital signature mechanisms to record and recognize changes in data. These are two of the most important layers of the semantic web since failure to verify the reliability of the results and provide trust between information and user would render the web unusable.

Of all the layer described above, the RDF and Ontology are the ones that revolve around the actual data and are therefore the most relevant for this project. The following two sections expand a bit further on these two layers introducing terminology that will be used throughout the remainder of this report and presenting simple examples as a guide.

2.2 RDF

As mentioned in the previous section, the RDF is a framework for describing metadata relevant to an individual piece of information. The RDF has four main constructs:

- Resource: This is the core of the RDF and corresponds to anything that can be assigned a URI. Typical examples of resources include web pages, elements in an XML document, or anything in the universe of things that surrounds us whether it is on the web or not. For illustration purposes

let's assume that the thing the resource pertains to is a book and the URI is the book's isbn (e.g. urn:isbn:0141182709)

- Property: This represents a named attribute that pertains to the resource. For the example mentioned above, such properties could be the author of the book, its title, or the year of its publication. These properties can in turn also be resources under the RDF framework.
- Value: This represents the value of a given property. This value can have any of the know data types (i.e. string, integer, etc.) or as mentioned above be a URI of another resource with its own set of properties. For instance, a value for property author can be another resource URI for author George Orwell
- Statement: This can be viewed as a sentence that combines the above three constructs. So for instance one could state that 'The *author* of *urn:isbn:0141182709* is *George Orwell* '. Similar to the properties and values, statements can also be resources of their own with properties in this recursive RDF framework. These statements are also known as semantic or RDF triples that have the form of subject-predicate-object.

If we assume a table of data like in Figure 3, each row (or record) represents a resource with the URI as the identifier of that resource, the column header is the property, and the cell value is the value. However, the RDF framework is far more flexible than traditional tabular data as it allows for meta descriptions to be applied to properties, values, and even the entire statement itself thus converting the table into a graph.

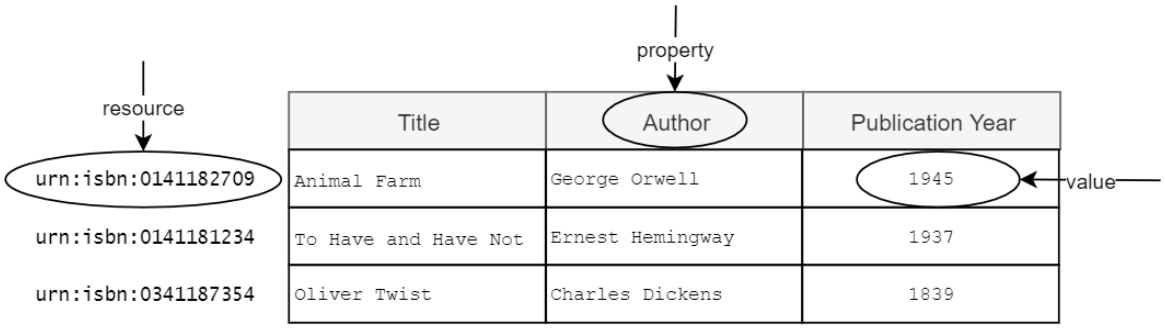

Figure 3. RDF to Tabular structure parallels

The questions on the information are formed in a language that is suitable for accessing RDF data (e.g. SPARQL)

2.3 Ontology

Following on from the RDF section that provided an overview of the framework that can be used for the representation of the information, this section focuses on the ontology or knowledge base itself. An ontology is a generalised data model that can be used to describe the general types of things (i.e. it

doesn't include any information about domain specific individuals). To do so, an ontology comprises three main components:

- Class: which is a type of a thing included in the ontology
- Attributes: which are properties that describe an individual class and
- Relationships: which are properties that connect to classes

Returning the example of the book table presented in Figure 3, a simple ontology as illustrated in Figure 4 can be built to model the data. Two of the classes (or types) in that ontology would be Book and Author. The class of Book could have the attributes Title and PublicationYear while the class Author could have the attribute of FullName. Finally, the two classes could be linked together by a relationship like hasAuthor.

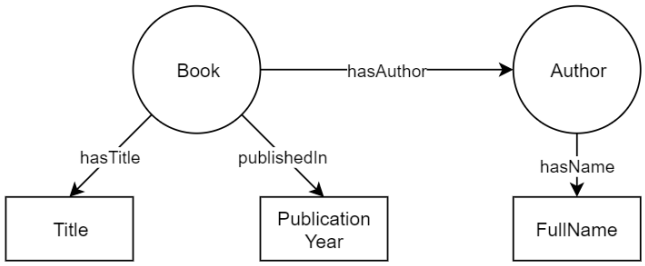

Figure 4. Ontology Example

Using this ontology as a blueprint a knowledge graph can be created by adding real data for books and authors. The term knowledge graph was first introduced by Google (Singhal, 2012) and has since been widely used to describe a knowledge base whose content is organised as a graph. The graph's nodes are entities of a specific type (i.e. class), and the edges are relationships between them. This graph structure enables data exploration by navigating from one part of the graph to another other using the provided links.

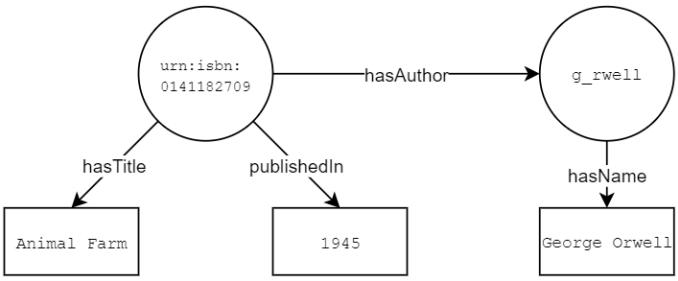

Figure 5. Example of a knowledge graph (only two nodes presented)

Some of the most widely used knowledge graphs are Wikidata (Denny and Markus, 2014) and DBpedia (Lehmannet et al., 2014). In fact, DBpedia will be used for the experiments conducted in this paper

Now that we've covered the theory of the Semantic Web and knowledge graphs is worth revisiting the problem that the project aims to solve. The pipeline will accept as input a set of tabular data and will aim to identify the class (type) for each of the columns as well as the instance itself (i.e. entity). In other words, using the example above the pipeline will try to identify if the contents of a columns belong to the class 'Book' or the class 'Author' (column type annotation) and following that try to identify the individual instance of that class that corresponds to the cell value (e.g. 'Animal_Farm' or 'Oliver_Twist') (cell entity annotation).

2.4 Related Work

The scope of this project has been inspired by the Semantic Web Challenge on Tabular Data to Knowledge Graph matching (SemTab) challenge that was first introduced in 2019 (Jimenez-Ruiz et al., 2020). This challenge invited parties to implement systems that deal with the column type annotation (CTA), cell entity annotation (CEA) and column property annotations tasks (CPA).

To generate the benchmark dataset SemTab 2019 used DBpedia as the reference knowledge graph (and in fact its English annotations) however the approach is generic enough that the KG can be replaced by any other (e.g. Wikidata or other domain specific KG).

To evaluate the proposed solutions the challenge used the traditional precision, recall and F1 scores for the CEA and CPA task whereas for the CTA task two different measures were used Average Hierarchical Score and Average Perfect Score in order to take into account the taxonomy of the classes.

In the following sections present some of the proposed solutions for the SemTab challenge that formed the basis for the implementations suggested by this project.

2.4.1 MTab

MTab (Nguyen et al. 2019) is the top performing proposal out of all participants in the 2019 challenge across all three tasks.

Prior to dealing with the tasks as hand the proposed system is performing some preprocessing step to clean the data and extract some metadata for the given tables. In summary the preprocessing is attempting to rectify incorrect encoding predict the language of the table cell values as well as metadata on their data types and entity type from another KG (OntoNotes 5) which is also manually matched to DBpedia. Finally, the system performs a lookup directly to DBpedia or indirectly through redirected links from Wikidata to retrieve a list of candidate entities considering the language parameters identified in one of the previous steps.

Following the prepossessing there is a two-phase approach whereby:

- in the first phase the system estimates the candidate according to relevance for each of the 3 tasks (entity, type and relation) and

- in the second phase those candidates are refined to come up with the final output of entity, property and class respectively. For entity re-estimation the algorithm is calculating the probability by combining the probability of the candidates from the first phase (entity, type and relation) with certain weights. Finally, it is using the entity with the highest probability to define the property and class.
- The key strengths and differentiating elements of this approach are:
- the use of additional services, specifically DBpedia Lookup, DBpedia endpoint, Wikipedia and Wikidata.
- the language consideration
- the introduction of a fuzzy matching (e.g. Levenshtein distance) instead of exact term matching when looking up candidates

2.4.2 IDLab

Similar to MTab, IDLab (Steenwinckel et al. 2019) is also proposing a solution for all three subtasks of the challenge. IDLab adopts a multistep approach trying to resolve the more specific subtasks of CEA and then using the results to infer the properties and classes.

To identify the candidate entities for CEA IDLab uses the cell value, following some basic text processing, to generate a URL in the dbpedia domain. If that URL is a valid one that entity is added to the pool. In parallel, the DBpedia Lookup is utilized with the cell value to produce more candidate and in case both of the above yield no valid results, DBpedia spotlight is used. Finally to derive the best match the system calculate the smallest Levenshtein distance between the rfs:label of each candidate entity in the pool and the cell value.

For the CTA tasks the class of the identified CEA entities are assessed taking into account a majority vote as well as the taxonomy that in inherent to DBpedia. This solution is trying to select the deepest node in the tree while at the same time also maintaining all the ancestors and equivalents as candidates for classes. This step is repeated near at the final step to refine the selection.

For CPA the system is also assessing all predicates of the (subject, object) pairs between columns and selects the most frequent one. Domain and range of column types is considered to break ties.

2.4.3 ColNet

Another approach to assign types (i.e. classes) to columns of tabular data has been proposed by ColNet. ColNet (Chen et al.2019) doesn't assume that there are any metadata like column names or even entities of the cell values in the reference knowledge graph. Instead, it uses the KG to automatically train a convolutional neural network CNN that would then predict types for columns not only based on the individual cell values but also the embedded semantics of the entire column. That way it also manages

to address the presence of knowledge gaps in the KG (i.e. the instances where not all cell values from the tabular data have a corresponding entity in the KG).

In summary ColNet comprises three key steps: lookup, prediction, and ensemble. In the lookup step entities in the KG that are matched to the column cell values are retrieved and their classes are added to the list of candidate classes for the column type annotation. The matched entities form a set that is called particular entities and all entities of the candidate classes for a set of general entities. These two sets are used to form positive and negative training sample to train the CNN.

After training the CNNs for each candidate class with the positive and negative samples that have been converted to vectors using a word representation model like word2vec ColNet tries to predict each column type. In the final step, ensemble, the CNN predictions are combined with the entities retrieved by the lookup in a way that rejects classes supported by few cells while accepting classes supported by a large part of the cells.

2.5 Convolutional Neural Networks

Convolutional Neural Networks (CNN) are a specialised type of neural networks (NN) that have potential of taking into account the special correlation of the input data (LeCun, Bengio and Hinton, 2015). A traditional neural network consists of a set of units (i.e. perceptrons) (Rosenblatt, 1958), which perform a linear transformation on the input vector followed by a non-linear activation function. The depth of a NN is defined as the number of layers including the output layer but not the input layer. There isn't any restriction as to how deep a neural network can be.

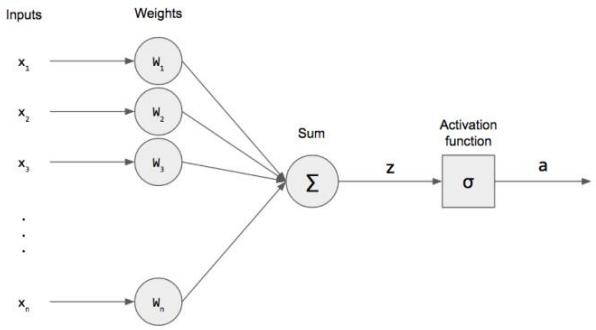

Figure 6. Example of a perceptron

The difference between CNNs and traditional NNs is that CNN employ convolution instead of traditional matrix multiplication in at least one of their layers (Goodfellow, 2016).

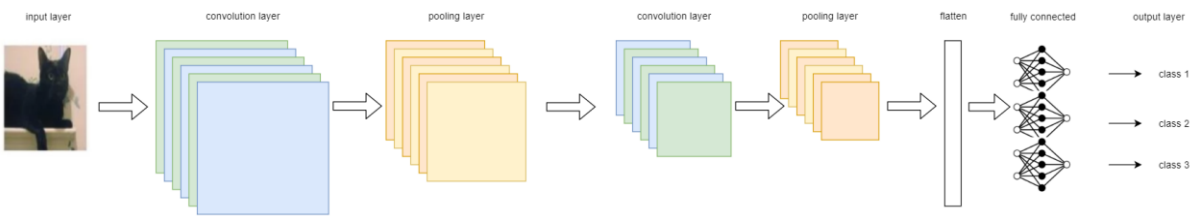

Figure 7. Example of a convolutional neural network

A typical CNN comprises the following types of layers:

- Convolutional layer: This is the layer that convolves the input and passes the result to the next layer. The input is typically a grid of data (i.e. tensor) with a shape that is defined as [(number of input samples), (height), (width), (channels)].
- Activation layer: This layer applies the activation function to the output of the convolutional layer (Yingying et al. 2020). There are may activation functions suggested in literature like the sigmoid or the tanh but the most commonly used one (and the one selected in this project) was the Relu function. Relu is a piecewise function that will force the output to be zero if the input value is less than or equal to zero. Otherwise, it will make the output value equal to the input value.
- Pooling layer: This layer reduces the dimensions of the tensor of the previous layer by representing each cluster of neighbouring cells as a single feature for the next layer. This is typically done with the use of tiles of specific dimensions (e.g. 2X2) that may or may not overlap as they scroll over the input tensor. Two types of pooling are suggested. Max Pooling that represents a cluster by the max value of its participants and Average Pooling that uses the average of the participant values in its stead.

The above layers convolutional > activation and pooling can be repeated multiple times in a CNN and eventually are followed by a fully connected layer that eventually categorise the input samples in the given classes. Before sent to the fully connected layer, the tensor from the final convolutional layer is flattened into a shape that can be provided as input to the NN.

3 Methods

SORT INTRO

3.1 Input Data

The scope of the project is to automatically predict the types of columns in tabular data and the entities that are relevant to cell values. Therefore, by definition we needed to have two sets of inputs:

- The *tabular data* to analyse and predict the types / entities of and
- A *reference knowledge base* containing candidate types / entities the columns / cell would be matched with accordingly

3.1.1 Tabular Data

In theory, the tabular data could be in any format however for the purposes of this project the input used was from the SemTab Challenges (2019 and 2020) who provided the data in csv files. Each csv file has a header row with the column titles, and several rows of data. On top of the tabular data that form the input dataset SemTab also provide two more datasets that are optional for the end-to-end pipelines:

- Targets: Files containing the column indexes from each file that need to be considered for type annotation or column/row indexes for the cells that need to be annotated with an entity
- A file with the ground truth (i.e. the actual classes from the reference knowledge base corresponding to each of the columns)

Even though these additional files are optional for the end-to-end annotations, the pipelines use them in order to filter down the columns / cells that need to be considered for the prediction. The pipelines also use the ground truth files to evaluate the annotation precision.

3.1.2 Reference Knowledge Base

The reference knowledge base really depends on the input tabular data. However, for the experiments that were run as part of this project DBpedia was considered as input.

3.1.3 Word2Vec

In order to provide vectors of words as input to a neural network we need to convert each work to a numerical vector in a way the relevant words are closer in space than less relevant words. For the purposes of this project a pre-trained word2vec model that is readily available was used to convert strings to numerical vectors.

3.2 Data Processing

The next step is to load the data from the inputs mentioned above in a structure that can then be used further down in the lookup, training and annotation steps. Throughout this project json was used as a

flexible structure for the data so that they can be easily accessible. Moreover, a dictionary is a mutable structure in Python so updates on it could be done at different stages.

The format of the dictionary called ‘data’ is illustrated below. As shown in the example each csv file is a separate object in the dictionary with the following attributes that are also dictionaries:

- column_titles: This attribute holds the titles of the columns as specified in the incoming csv. This is an optional feature that could be deactivated in case the incoming data do not have any titles. For the analysis and testing done on this project, column titles are indeed ignored
- data: This is the key attribute and has the cell values for each of the columns in the tabular data. The processing of loading data takes into account the target columns mentioned in 3.1.1 and ignores any other column. This feature can also be disabled in case the target columns are not available. Additionally, when loading the data, the function provides the option of storing every cell value in a column or only the unique values that appear in that specific column. Experiments have shown that keeping multiple instances of the same value does not improve the results. Therefore, unless specifically mentioned in the experiment, the default for the function is to retrieve the unique cell values.

  Due to the above statement the cell values across the columns are not aligned. i.e. the n^{th} value in the array for column 1 in the below example doesn’t correspond to the n^{th} value in column 3.
- gt: Finally, this attribute has the expected type for each of the columns. This attribute is only used at the final stage of the process, after a type has been predicted, to assess the accuracy of each approach that is being tested. Once more if the ground truth, referenced in 3.1.1, is not available, this key can be eliminated from the dictionary without any impact in the downstream pipeline of predicting the column types

```
{
    "58891288_0_1117541047012405958": {
        "column_titles": {
            "1": "Title",
            "3": "Director(s)"
        },

        "data": {
            "1": [
                "Gone with the Wind",
                "The Shawshank Redemption",
                "The Battleship Potemkin",
                ...,
                "Gladiator"
            ],
            "3": [
                "Mel Gibson",
                "Orson Welles",
                "Francois Truffaut",
                ...,
                "Woody Allen"
            ]
        },
        "gt": {
```

```
            "1": "Film",
            "3": "Person"
        }
    },
    "8468806_0_4382447409703007384": {
    ...
    }
}
```

Restructuring the csv inputs into the above dictionary structure enables the rest of the pipeline to remain agnostic of the input format. Therefore, if the tabular data was presented in a different structure (i.e. not the one provided by SemTab), so long as a component could be built to transform the data in the above dictionary, the pipeline could still proceed with the type annotation without any further changes.

3.3 Entity Lookup

With the data loaded in the data dictionary the next step is to lookup the cell values in the DBpedia endpoint and get the candidate entities and their types. Lookups against other reference knowledge graphs haven't been considered as part of this project as the target types were all from DBpedia, however the module can be replaced as long as the output results are still logged in the structure that will be described later in this section

The DBpedia lookup comprises two steps. The second being an optional one:

3.3.1 Step 1: Cell Value Lookup DBpedia API

For this first step the function is making a call to the DBpedia lookup API.

`http://lookup.dbpedia.org/api/search/KeywordSearch?`**`MaxHits`**`=5&`**`QueryString`**`=Cell Value`

The API provides two keys for the request as follows:

QueryString: This is a mandatory field that contains the keywords that needs to be queried. The API does a fuzzy matching between the keyword and the dbpedia entity labels, so the retrieved results are not always exact matches on the label. This is the reason that rendered the text pre-processing step originally provisioned in the pipeline (e.g. stemming, lemmatizing) unnecessary. The only pre-processing of the cell value that is performed is the removal of characters ('[' and ']') that appeared in some cell values and made the API request invalid.

MaxHits: This is an optional attribute that enables the user to limit the number of the returned results. As mentioned above the results are retrieved based on a fuzzy matching with the most relevant appearing at the top followed by less relevant results. For the purposes of this analysis the process is flexible enough to store the x top lookup results for each cell value since. The value of x has been defaulted to 5 to enable more candidate entities and classes to be accessed for each column while at the same time not exploding the size and computational requirements of the experiments (if a very high value was selected for x).

The response from the API is an xml that is being parsed by the lookup function to retain the URI of the retrieved entity, and the URI(s) of the associated class(es). Only eWe also maintained the rank (i.e. place in the top 5) the result came in

3.3.2 Step 2: Retrieve Entity Type(s)

The next step was added later on in the process when the analysis of the lookup results illustrated gaps in the retrieved entities from the API. There are many instances where, for whatever reason, the lookup API response fails to retrieve the class(es) of the identified entity. In the example below the third result of the request for the cell value 'A Streetcar Named Desire'

(i.e.https://lookup.dbpedia.org/api/search/KeywordSearch?MaxHits=5&QueryString=A%20Streetcar%20Named%20Desire) is URI 'https://dbpedia.org/page/A_Streetcar_Named_Desire_(1951_film)' and it is retrieved without any associated classes (i.e. <Classes/>).

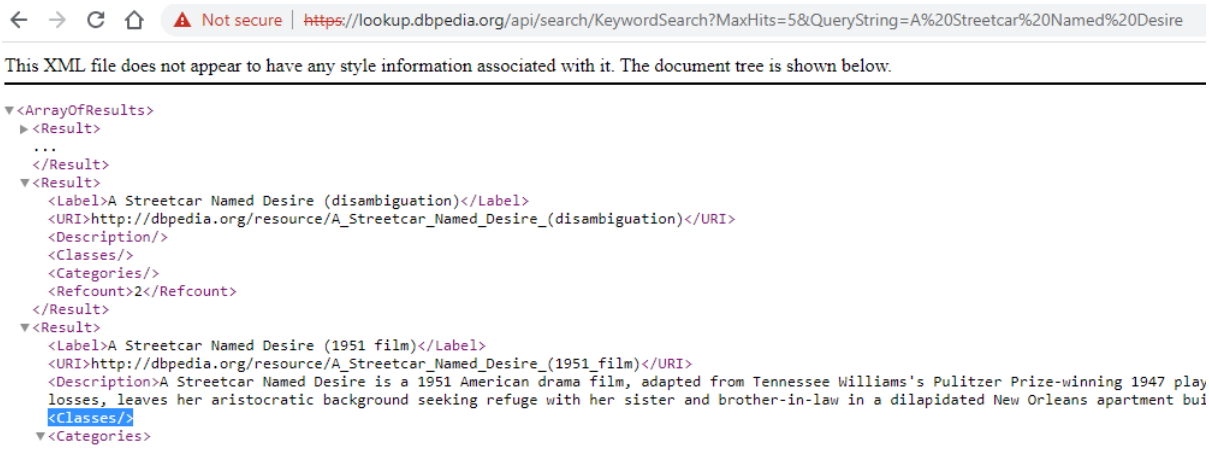

Figure 8. Results from the dbpedia lookup api when querying the cell value 'A Streetcar Named Desire'

However, when visiting the actual URI of the retrieved entity it appears that the entity does indeed have associated rdf:types [Figure 9]. To overcome this issue, the process performs a second lookup for those entities that came back without a type. This time it using the dbpedia sparql endpoint to send a request of the specific entity URI and retrieve any rdf:types in the dbo namespace. In Figure 9 the two relevant types have been highlighted with a black frame.

Finally, if the retrieved entity doesn't have any associated type (e.g. the second result in the lookup response http://dbpedia.org/resource/A_Streetcar_Named_Desire_(disambiguation)) then the entity isn't considered at all in the structure the process is creating.

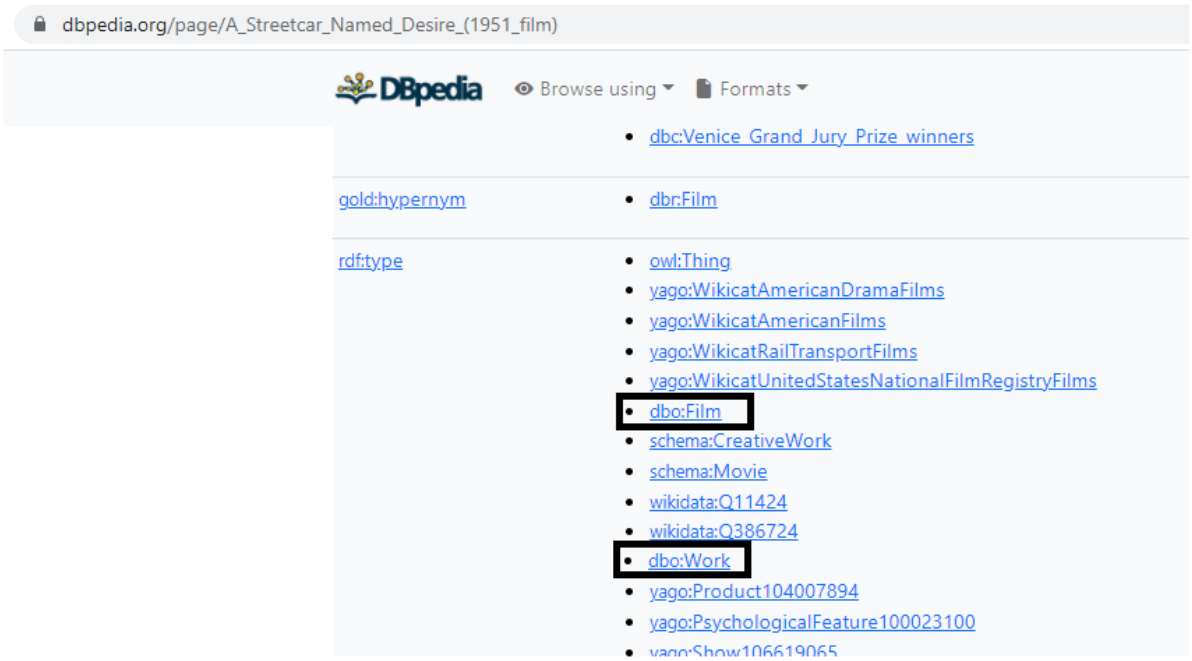

Figure 9. Types associated to the entity 'https://dbpedia.org/page/A_Streetcar_Named_Desire_(1951_film)'

Each cell value is only looked up once when it appears for the first time in the 'data' json, however the process still keeps track of any additional columns the same cell value might have appeared in, as well as all candidate entities and classes it may have matched to.

Once again, the results of this lookup are maintained in a json object so that the lookup process need only be run once (even offline) for every new in tabular data dataset.

The outcome of the lookup is stored in the cell_values dictionary as follows. The example below shows the results for the cell value for 'A Streetcar Named Desire' as a continuation of the previous example


```
"A Streetcar Named Desire": {
        "location": [
            [
                "58891288_0_1117541047012405958",
                1
            ],
            [
                "20135078_0_7570343137119682530",
                1
            ],
            [
                "35188621_0_6058553107571275232",
                1
            ]
        ],
        "candidate_entities": {
            "A_Streetcar_Named_Desire": {
                "rank": 1,
                "candidate_classes": [
                    "Play",
                    "WrittenWork",
                    "Work"
                ]
```

```
        },
        "A_Streetcar_Named_Desire_(1951_film)": {
            "rank": 3,
            "candidate_classes": [
                "Film",
                "Work"
            ]
        },
        "The_Originals_(season_3)": {
            "rank": 4,
            "candidate_classes": [
                "TelevisionSeason",
                "Work"
            ]
        },
        "A_Streetcar_Named_Desire_(opera)": {
            "rank": 5,
            "candidate_classes": [
                "TelevisionShow"
            ]
        }
    }
```


In this json structure each cell value is a dictionary in its own right with the following keys:

- location: This is an array of tuples where each tuple contains a (file, column index) describing where the value appeared in. As mentioned earlier, each value may appear only once in a given column therefore for the CTA task the actual index of the row the cell value appeared is irrelevant
- candidate_entities: this is a dictionary where each retrieved entity (up to 5) is a separate key. Within the entity keys the structure provisions for:
    - rank: the order in which the entity appeared in the results from the lookup API starting with 1 for the most relevant and ending at 5 for the least relevant. It's not necessary that all 5 ranks will appear as keys for a given cell value since the pipeline ignores candidate entities that don't have types in the dbo namespace. This is the reason why rank 2 is missing from the above example
    - candidate_classes: an array of all the dbo types associated to the specific entity (the can be one or more type per entity)

Finally, on top of the *data* and *cell_values* structure mentioned in 3.2 and 3.3 (above) respectively there are two more structures that reshape that data in order to be used for the next steps of the pipeline. The first is a structure that is used to predict a class by voting whereby each input csv is represented as a key with the following attributes:

MORE….

3.4 Create and train convolutional neural network models

The idea behind the use of the CNNs to annotate a column is to combine multiple cell values from one column to identify collectively the appropriate classes. The CNN created for each candidate types comprises an input layer, followed by several convolutional layers (2-3) with max pulling for feature selection and dropout. The output of each convolutional layer is passed on to the next apart from the very last one that goes into a dense fully connected layer after having been flatten and finally there is the last dense layer to provide the binary classification. Relu is used as the activation function for the convolutional and dense layers

Limitations on training size some classes are overlapping so what is a film may also be a playwright

The next sections describe the pre-processing used to create the samples for the input layer. As shown in the above Figure, this input essentially an x by y matrix of numerical values. To reshape the cell values into a format that is compatible with the input layer, the following steps are followed

3.4.1 Create synthetic columns of cell values

For every column, several synthetic column samples are created with a synthetic column of size x (parameterised as 'synthetic_column_size'). Two approaches have been used for the sample generation:

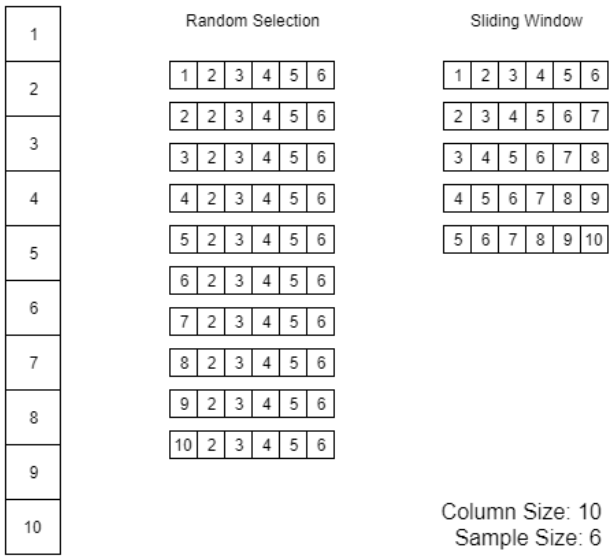

Figure 10. Sample generation from a column of size 10 using the random selection and sliding window approaches

Random selection: In this case, the synthetic column contains the cell value itself plus an additional x-1 randomly selected cells. In case the column length is smaller than the number of samples needed (i.e.

< x-1) then all cell values are selected and the remaining positions in the synthetic column are populated with nulls.

Sliding Window: In this case, a window of size x starting from the current cell value is used to generate the sample. For the next sample, the window slides by one position and so on and so forth. This technique will generate less samples than the random selection (i.e. [column_size – x + 1] samples as opposed to [column_size] samples. In case column_size – x + 1 only one sample will be created which will have all values in the column and nulls for the remaining positions needed to complete the correct sample size x.

Example

cell_value = ‘Joseph L. Mankiewicz’

synthetic column = ['Joseph L. Mankiewicz', 'James Whale', 'Frank Darabont', 'Sam Mendes', 'John Huston', 'Mike Nichols', 'John Frankenheimer', 'Charles Chaplin', 'Harold Ramis', 'Federico Fellini']

3.4.2 Break sample of values to sample of words

The next step is to convert the list of cell values in the synthetic column above to a list of words. That is done by replacing special characters, which are typically used to separate words (e.g. '_', '-', '.', '/', '"', '"""), with spaces. Finally the derived string value is tokenised using the space (i.e. ‘ ’) as the delimiter. The size of this sequence is typically longer than the size of the synthetic column in order to allow for cell values comprising more than one words. Any words produced by the tokenizer that cannot fit the length of the sequence are dropped. For instance, in the above example where synthetic_column_size = 10 if sequence_size = 20 then the word ‘felini’ will be dropped and the sequence for the specific synthetic columns will be

synthetic_column_sequence_20 = ['joseph', 'l', 'mankiewicz', 'james', 'whale', 'frank', 'darabont', 'sam', 'mendes', 'john', 'huston', 'mike', 'nichols', 'john', 'frankenheimer', 'charles', 'chaplin', 'harold', 'ramis', 'federico']

On the flipside if the produced words from the tokeniser are less than the length of the sequence then the remaining positions are once again filled up with nulls. For instance, in the same example, if sequence_size = 30 the produced sequence will be as follows

synthetic_column_sequence_30 = ['joseph', 'l', 'mankiewicz', 'james', 'whale', 'frank', 'darabont', 'sam', 'mendes', 'john', 'huston', 'mike', 'nichols', 'john', 'frankenheimer', 'charles', 'chaplin', 'harold', 'ramis', 'federico', 'fellini', 'NaN', 'NaN', 'NaN', 'NaN', 'NaN', 'NaN', 'NaN', 'NaN', 'NaN']

There needs to be a balance as to how long the sequence should be in relation to the synthetic column size. If the ratio is too low, then we may end up losing many words from the cell values but if too big we will have longer sequences to process with many nulls that are not adding any value to the

classification. For this reason the pipeline calculate the average word length the cell values (typically a float between 2 and 3) and sets the value of the sequence size in multiples of the sample column size. For the experiments presented in Chapter 0 these [sample column size] = 10 and the sequence size is three times that i.e. [sequence size] = 30.

3.4.3 Word2Vec

Finally, each word is converted into a numerical vector based on the pretrained model that was mentioned in 3.1.3. so that vectors of semantically related words appear closer to one another. The size of the word vector in the model used is 200 therefore each sample is finally a 30 x 200 size matrix.

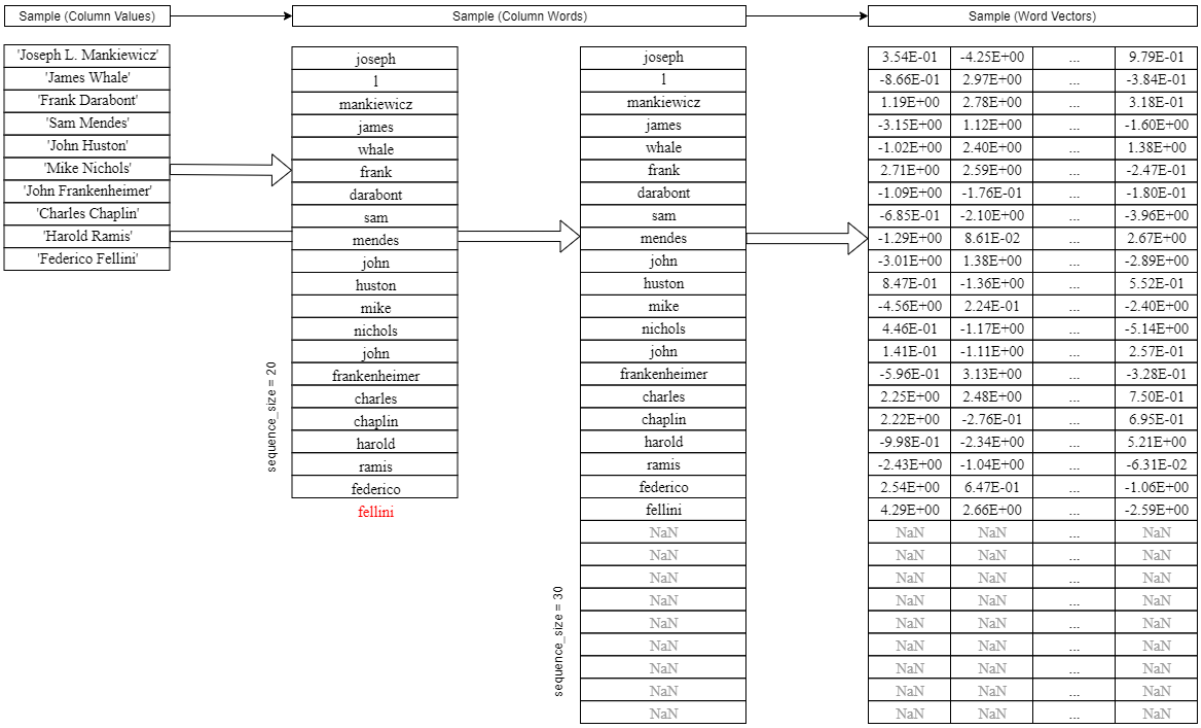

Figure 11. Creation of samples for the input layer of the cnn (used for training and predictions)

3.5 Column Type Annotation (CTA)

Having pre-processed the data, extracted candidate entities / types and trained the convolutional neural networks for every potential candidate type that appears in the retrieved entities, the next and final step of the type annotation pipeline is to predict the types of the target columns. The following sections present the different approaches implemented for the task at hand.

3.5.1 Voting

Using the list of candidate entities and types of each cell value of a given column, the pipeline proceeds with a very basic column annotation by considering the types that are proposed by the retrieved candidate entities. There are several experiments that will be presented in Chapter 4 in terms of how the voting works. Basic permutations take into account:

- the rank of an entity in the api response (only top 5 results are considered), as well as
- the entity hierarchies within a knowledge base (i.e. parent child relationship by the rdf:subClassOf predicate) and
- the frequency of the candidate entities in the target columns

3.5.1.1 Equal votes

In this voting approach a vector of candidate classes is created for each column C_{ij}:

$$C_{ij} = \begin{bmatrix} p_{type\_1} \\ p_{type\_2} \\ \cdots \\ p_{type\_n} \end{bmatrix} \quad (3.1)$$

where C_{ij} is the vector of column *j* in file *i*, *n* is the number of candidate types suggested for this specific columns and $p_{type\_1} \ldots p_{type\_n}$ are the number of votes that a candidate class has received. The votes for each candidate type are calculated as follows:

$$p_{type} = \frac{1}{N} \sum_{e_1}^{e_k} \begin{cases} 1 & ,if\ rdf{:}type(e) = type \\ 0 & ,otherwise \end{cases} \quad (3.2)$$

where *N* is the total number of candidate entities retrieved for the cell values in column *j* in file *i* and rdf:type(e) is the class that is assigned to the entity e. Please note that in case a retrieved entity *e* is allocated more than one classes, then it contributes more than one votes (i.e. one in each class) and N is also increased accordingly.

Finally, as depicted in (3.2) each entity's vote has the same weight, regardless of the entity's rank in the retrieved results. For instance, if type *film* is suggested by two entity results, one with rank=1 and one with rank=3 the total votes the type *film* will receive will be 2 (i.e. $p_{film} = 2/N$)

3.5.1.2 Weighted votes

This is a variation of the previous voting approach where the retrieved entities get a vote that is inversely proportional to their rank. The calculation of the vote in (3.1) is as follows

$$p'_{type} = \frac{1}{N} \sum_{e_1}^{e_k} \begin{cases} 1/r & ,if\ rdf{:}type(e) = type \\ 0 & ,otherwise \end{cases} \quad (3.3)$$

where r is the rank the entity came in in the results. For instance, as illustrated in Figure 8 there are two entities retrieved :

- http://dbpedia.org/resource/A_Streetcar_Named_Desire with r = 1 and
- http://dbpedia.org/resource/A_Streetcar_Named_Desire_(1951_film) with r = 3

The first entity contributes a vote of 1/1 = 1 in all three associated types ['Play', 'WrittenWork', 'Work'] and the second entity contributes a vote of 1/3 = 0.33 in all two associated types ['Film', 'Work'].

3.5.2 TF-IDF

This approach resembles the logic of TF-IDF in an effort to promote more specific candidate classes and penalize others that appear as candidates for a large number of columns. In the use case of this project, the corpus is a set of columns instead of a set of documents and the term is a candidate class (type) instead of a generic word or string.

First, we calculate the equivalent of the term frequency as the follows:

$$tf_{type,col} = \frac{f_{type,col}}{\sum_{i=1}^{k} f_{type_i,col}} \qquad (3.4)$$

where $f_{type,col}$ is the frequency of the candidate class in the given column of the file, and k is the total number of candidate classes appearing in this column. The experiments that will be presented in Chapter 4 examine different thresholds for the rank of the lookup results so when $rank \leq 5$ the number of candidate classes k will be greater than the number k when $rank = 1$.

Next, we calculate the equivalent of the inverse document frequency as follows:

$$idf_{type} = \log \frac{N}{1+|\{col \in Cols: type \in col\}|} \qquad (3.5)$$

where N is the total number of target columns (i.e. columns we want to predict the type of) and $|\{col \in Cols: type \in col\}|$ is the total number of columns for which the specific type appears at least once as a candidate class. The denominator is adjusted by adding 1 to avoid division by zero for types that are not present in any target columns even though that use case is not possible in this implementation since we would not be calculating the tf-idf for classes that don't appear at least once.

Finally, for each candidate class for each column we calculate the tf-idf score as follows:

$$tf\_idf_{type,col} = tf_{type,col} * idf_{type} \qquad (3.6)$$

It should be reminded that C_{ij} (i.e the vector of column j in file i) is still represented as in (3.1) but now

$$C_{ij} = \begin{bmatrix} tf\_idf_{type\_1} \\ tf\_idf_{type\_2} \\ \cdots \\ tf\_idf_{type\_n} \end{bmatrix} \qquad (3.7)$$

Where $tf\_idf_{type\_1}$ is the tf-idf score of type_1 in column j

3.5.3 CNN

The CNN approach for column type annotation the pipeline is presented in Figure 12. As a first step for every column that needs to be annotated the pipeline retrieves the candidate types are calculated in previous steps (3.3.2). The candidate type scores from the previous steps (3.5.1, 3.5.2) can also be used if there's a need to limit the universe of type that are assessed for each column.

Next the pre-trained cnn models for all or a top x subset of the candidate classes based on scoring are retrieved. Please note that these models have been trained offline as part of the steps presented in 3.4.

Additionally, samples are extracted from the column to provided as the inputs to the models. The sample generation is the same as that described in section 3.4 using both the random selection and the sliding window.

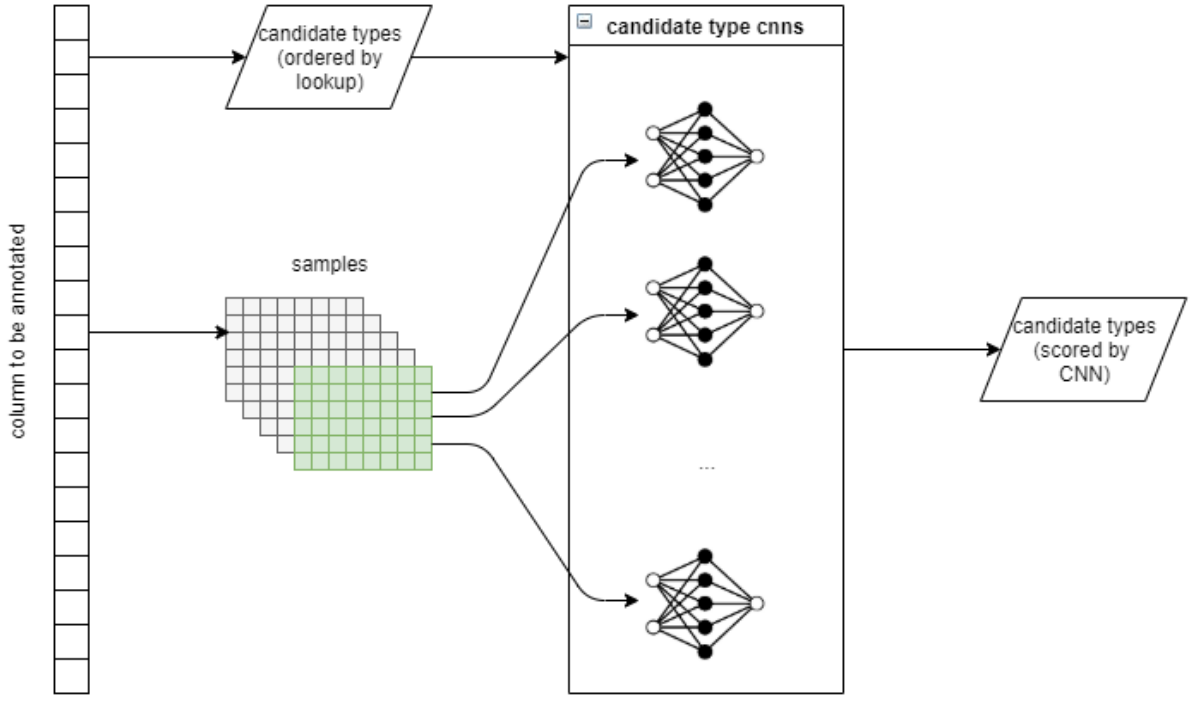

Figure 12. Using pre-trained cnns for column type annotation

Finally, the pipeline assigns a likelihood score to each of the candidate types ranging from 0 to 1. This score is calculated as the number of samples predicted to be in the class (i.e. cnn output = 1) divided by the total number of samples provided as input to the classifier. By definition, the closer the score is to 1 the more likely it is for the type the cnn is trained for to be the correct type (as it has been suggested by a large number of samples). Same as with voting, the cnn outputs can help sort the candidate types (i.e. have a set of possible types rather than just one)

3.6 Cell Entity Annotation (CEA)

The cell entity annotation is an independent task for which a new pipeline is created. There are, however, elements of the CTA pipeline that are being reused (everything up till the entity / type retrieval). Moreover, optionally the CEA uses the outcomes of the CTA (best selected approach for type annotation) pipeline to further enhance the cell annotation.

Picking up after the retrieval of entities the simplest way for the cell annotation is to pick up the first entity retrieved from the lookup API without any prior knowledge of the type is should have.

Alternatively, the top 5 retrieved entities were assessed and the first entity in that sorted list with a type that matched the predicted classes was selected. If no entity in the top 5 had the correct type then no entity was suggested for the column annotation.

26.5(3,200_4,000_6250)

4 Results

This section presents the results for the experiments conducted with different permutations of the pipeline.

DATASET

4.1 Column Type Annotation (CTA) Results

To assesses the prediction results, the gt truth (i.e. the expected type per column) reference data is used. There are two measures employed:

- Strict precision: This only considers when the predicted class(es) contain the actual class. If the actual class is not in the predicted class(es) the column is counted as a false positive
- Relaxed precision: This measure will also give .5 a point when the predicted class(es) contain a parent class that the actual class is a subClassOf.

Given that the pipeline always produces a candidate type for each column the precision and recall mentioned in 4.2 calculate the same value. Moreover, when precision and recall are equal the f1_score is also the same value. That is why for the CEA results there only precision is being mentioned.

4.1.1 Lookup Voting

This series of experiments aims to find the optimum way of deriving the type of a column (using DBpedia as the reference knowledge graph) based exclusively on the results from the entity lookup API and querying the SPARQL endpoint.

4.1.1.1 Assess Ranking – Equal votes

In this series of tests, the candidate types for each column are ranked using an equal voting system from the respective candidate entities are described in section 3.5.1.1.

As mentioned in section 3.3 the pipeline retrieves the top 5 results from the lookup API. In this experiment we set a threshold increasing from 1 to five 5 with a step of 1 to examine the effect of being more relaxed with the retrieved lookup results. The precision illustrated in Figure 13 and Table 1 is the number of correctly annotated columns divided by the total number of target columns.

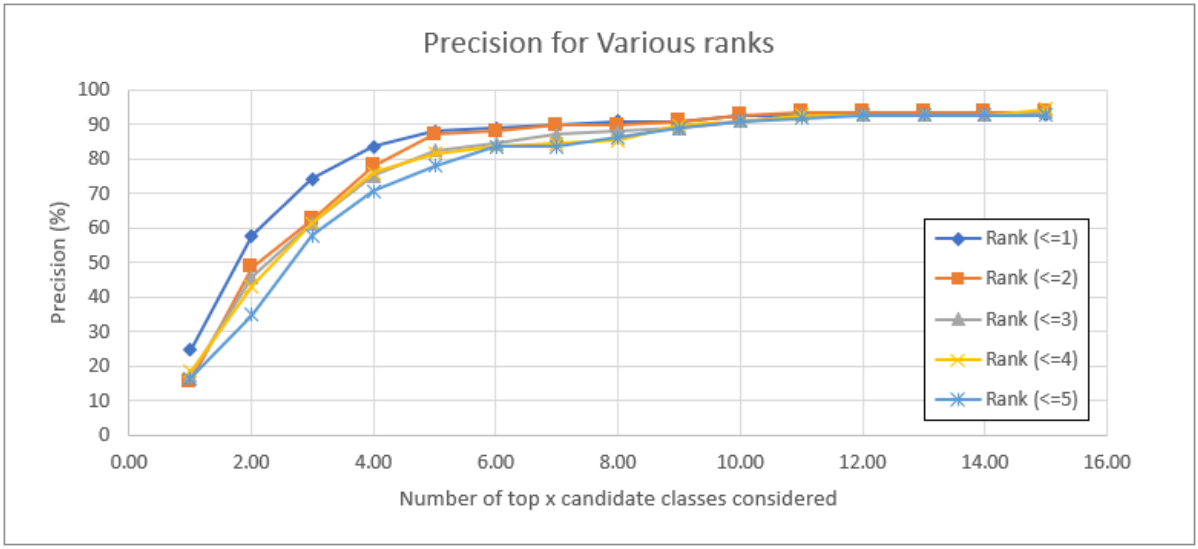

Figure 13. Results of lookup voting for increasing thresholds of candidate entity ranks

	Rank (<=1)	Rank (<=2)	Rank (<=3)	Rank (<=4)	Rank (<=5)
1	24.77 (34.86)	15.6 (27.06)	17.43 (28.9)	18.35 (28.9)	16.51 (27.06)
2	57.8 (64.22)	48.62 (58.26)	45.87 (56.42)	43.12 (54.59)	34.86 (48.62)
3	74.31 (80.28)	62.39 (72.02)	61.47 (70.64)	61.47 (70.64)	57.8 (67.89)
4	83.49 (87.16)	77.98 (83.03)	75.23 (80.28)	76.15 (81.19)	70.64 (78.44)
5	88.07 (89.91)	87.16 (88.99)	82.57 (84.86)	81.65 (83.94)	77.98 (82.11)
6	88.99 (90.83)	88.07 (89.45)	84.4 (86.7)	83.49 (85.78)	83.49 (85.78)
7	89.91 (91.74)	89.91 (91.28)	87.16 (88.99)	84.4 (87.61)	83.49 (87.16)
8	90.83 (92.2)	89.91 (91.28)	88.07 (90.37)	85.32 (88.99)	86.24 (89.45)
9	90.83 (92.2)	90.83 (92.2)	88.99 (90.83)	89.91 (91.74)	88.99 (91.28)
10	92.66 (93.12)	92.66 (94.04)	90.83 (91.74)	90.83 (92.2)	90.83 (92.2)
11	92.66 (93.12)	93.58 (94.5)	92.66 (93.58)	92.66 (93.58)	91.74 (93.12)
12	92.66 (93.12)	93.58 (94.5)	92.66 (93.58)	92.66 (93.58)	92.66 (93.58)
13	92.66 (93.12)	93.58 (94.5)	92.66 (93.58)	92.66 (93.58)	92.66 (93.58)
14	92.66 (93.12)	93.58 (94.5)	92.66 (93.58)	92.66 (93.58)	92.66 (93.58)
15	92.66 (93.12)	93.58 (94.5)	93.58 (94.5)	94.5 (94.95)	92.66 (93.58)

Table 1. Results of lookup voting for increasing thresholds of candidate entity ranks

Each result in Table 1 has two percentage values (e.g. 24.77 (34.86)). The first one, is only considering columns that have been correctly annotated i.e. columns where the target type is in the list of predicted type

$$precision_{col} = \frac{1}{N}\sum_{1}^{N}\begin{cases}1 & ,if\ type_{target} \in Type_{pred} \\ 0 & ,otherwise\end{cases} \quad (4.1)$$

Where N is the total number of target columns

The second percentage (i.e. the one in brackets) also rewards columns for which the predicted types don't include the target type, however include at least of the classes for which the target type is a subclass of.

$$precision'_{col} = \frac{1}{N}\sum_{1}^{N}\begin{cases} 1 & ,if\ type_{target} \in Type_{pred} \\ 0.5 & ,if\ parent\_class(type_{target}) \in Type_{pred} \\ 0 & ,otherwise \end{cases} \quad (4.2)$$

It was decided to take into account the parent classes in the evaluation since in those cases the pipeline manage to predict the "domain" in which the entities in the column belong but not the exact "subdivision".

As illustrated by the results allowing more candidate entities in the majority vote is hurting the precision of the prediction in comparison to only retrieving the top 1 class. For instance, the expected class is at the top 1 spot for 24.77% of the columns when using only rank 1 entities as opposed to 15.6% when using the ranks 1 and two. In raw number this mean 27 columns have a correct type predicted as opposed to 17 columns.

4.1.1.2 Assess Ranking – Weighted votes

In this series of experiments the second alternative to voting (as described in 3.5.1.2) is examined. The results are presented below in the same setup as in 4.1.1.1

	Rank (<=1)	Rank (<=2)	Rank (<=3)	Rank (<=4)	Rank (<=5)
1	24.77 (34.86)	17.43 (28.9)	20.18 (31.65)	20.18 (31.19)	19.27 (30.28)
2	57.8 (64.22)	49.54 (58.26)	46.79 (56.88)	44.95 (55.96)	44.95 (55.96)
3	74.31 (80.28)	64.22 (72.94)	61.47 (71.1)	61.47 (71.1)	61.47 (70.64)
4	83.49 (87.16)	79.82 (84.86)	77.98 (83.03)	77.06 (82.11)	77.06 (82.11)
5	88.07 (89.91)	88.07 (89.91)	85.32 (87.16)	83.49 (86.24)	83.49 (86.24)
6	88.99 (90.83)	88.99 (90.37)	86.24 (88.07)	83.49 (87.16)	83.49 (87.16)
7	89.91 (91.74)	90.83 (92.2)	89.91 (91.28)	88.99 (90.83)	88.07 (90.37)
8	90.83 (92.2)	90.83 (92.2)	90.83 (92.2)	90.83 (92.2)	88.99 (90.83)
9	90.83 (92.2)	91.74 (93.12)	90.83 (92.2)	90.83 (92.2)	89.91 (91.28)
10	92.66 (93.12)	93.58 (94.04)	91.74 (92.66)	91.74 (92.66)	90.83 (91.74)
11	92.66 (93.12)	93.58 (94.04)	91.74 (92.66)	92.66 (93.12)	92.66 (93.58)
12	92.66 (93.12)	93.58 (94.04)	92.66 (93.12)	92.66 (93.58)	92.66 (93.58)
13	92.66 (93.12)	93.58 (94.04)	93.58 (94.5)	93.58 (94.5)	93.58 (94.5)
14	92.66 (93.12)	93.58 (94.04)	93.58 (94.5)	93.58 (94.5)	93.58 (94.5)
15	92.66 (93.12)	93.58 (94.5)	93.58 (94.5)	93.58 (94.5)	93.58 (94.5)

Table 2. Results of lookup voting for increasing thresholds of candidate entity ranks(rank-weighted vote)

As expected, when only considering the top 1 result the precision of the weighted vote is the same as that of equal vote since all the weights are 1 and so the two voting approaches collapse into 1. However, we see that when increasing the rank gradually from 1 to 5 the resulting precision is marginally better with the weighted votes compared to the equal vote. This effect is particularly visible when only a few top classes from Cij are considered. As we consider more of the voted classes, the effect of the weighted vote wears off. Figure 14 shows the results when retrieved entities from all 5 ranks are considered. It is apparent that when selecting the less than the top 6 voted classes the precision with the weights is always

higher. However, from the top 6 onwards the precision of the two methods flips back and forth until it wears off altogether.

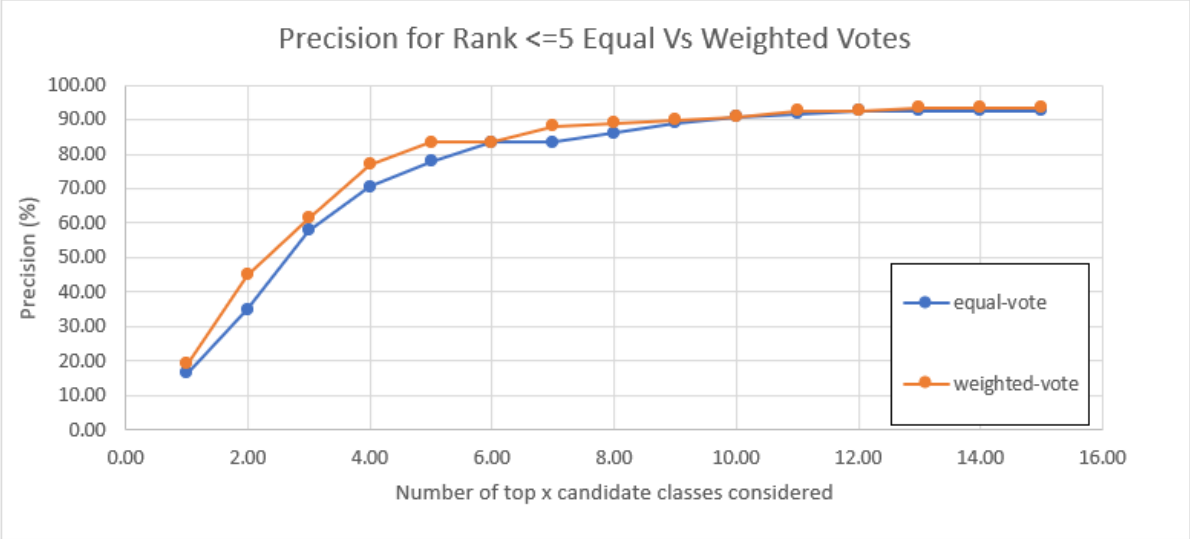

Figure 14. Results of lookup voting when all 5 ranks are considered with equal vote and majority vote respectively

4.1.2 Lookup with TF-IDF

For the next set of experiments, examines the performance of the TF-IDF logic presented in 3.5.2.

The same series of experiments were executed, whereby we gradually increased the rank of the considered candidates entities by one. The results are illustrated in Table 3:

	Rank (<=1)	Rank (<=2)	Rank (<=3)	Rank (<=4)	Rank (<=5)
1	75.23 (78.9)	72.48 (76.61)	67.89 (70.18)	54.13 (58.72)	52.29 (53.67)
2	81.65 (84.4)	79.82 (83.49)	77.06 (80.28)	68.81 (72.02)	69.72 (72.48)
3	87.16 (88.53)	88.99 (90.83)	84.4 (85.78)	78.9 (80.28)	74.31 (76.15)
4	87.16 (88.99)	88.99 (90.83)	88.07 (89.45)	83.49 (84.4)	78.9 (79.36)
5	88.07 (90.37)	91.74 (93.12)	89.91 (90.83)	86.24 (86.7)	80.73 (81.19)
6	88.99 (90.83)	92.66 (94.04)	90.83 (92.2)	88.07 (88.99)	81.65 (82.57)
7	90.83 (92.2)	93.58 (94.5)	92.66 (93.58)	90.83 (91.28)	84.4 (84.86)
8	91.74 (92.66)	94.5 (94.95)	92.66 (93.58)	90.83 (91.74)	85.32 (86.24)
9	91.74 (92.66)	94.5 (94.95)	94.5 (94.95)	90.83 (92.2)	86.24 (87.16)
10	93.58 (93.58)	94.5 (94.95)	94.5 (94.95)	91.74 (92.66)	86.24 (87.16)
11	93.58 (93.58)	95.41 (95.41)	95.41 (95.41)	94.5 (94.95)	88.99 (89.91)
12	93.58 (93.58)	95.41 (95.41)	95.41 (95.41)	94.5 (94.95)	89.91 (90.83)
13	93.58 (93.58)	95.41 (95.41)	95.41 (95.41)	94.5 (94.95)	90.83 (91.28)
14	93.58 (93.58)	95.41 (95.41)	95.41 (95.41)	94.5 (94.95)	92.66 (93.12)
15	93.58 (93.58)	95.41 (95.41)	95.41 (95.41)	94.5 (94.95)	92.66 (93.12)

Table 3. Results of lookup using the tfidf logic for increasing thresholds of candidate entity ranks

Compared to the previous two voting approaches, the tf-idf approach has a much superior performance with the precision of the rank 1 results jumping from ~25% with voting (equal or weighted) to ~75% with the tf-idf logic. The intuition behind this superior performance is that tf-idf penalises types that

may be too generic and thus appearing very frequently as candidate types for many columns while at the same time, it promotes more specific types which are usually subclasses of the above. However, it still allows for those generic classes to be selected as types for a column if their frequency of appearance for that specific column is quite high.

To further support this theory, we executed a small series of experiments where we filtered the list of candidate classes of a column by removing parent classes when any of their offsprings also appeared in the list of candidates.

The results are shown in Figure 15. It is clear that if we only considered the top 1 candidate class in each C_{ij} vector as the prediction even the crude approach of removing the parents provides better results than the equal and weighted vote, however, it is also quite inferior when compared to the tf-idf logic. Moreover, the results indicate that by removing the parent classes we filter out some useful candidates. For instance, for both tf-idf and equal/weighted vote the expected class for 90% of the columns is in the top 5 candidate classes but this percentage drops to 60% when removing the parent classes from the list of candidates.

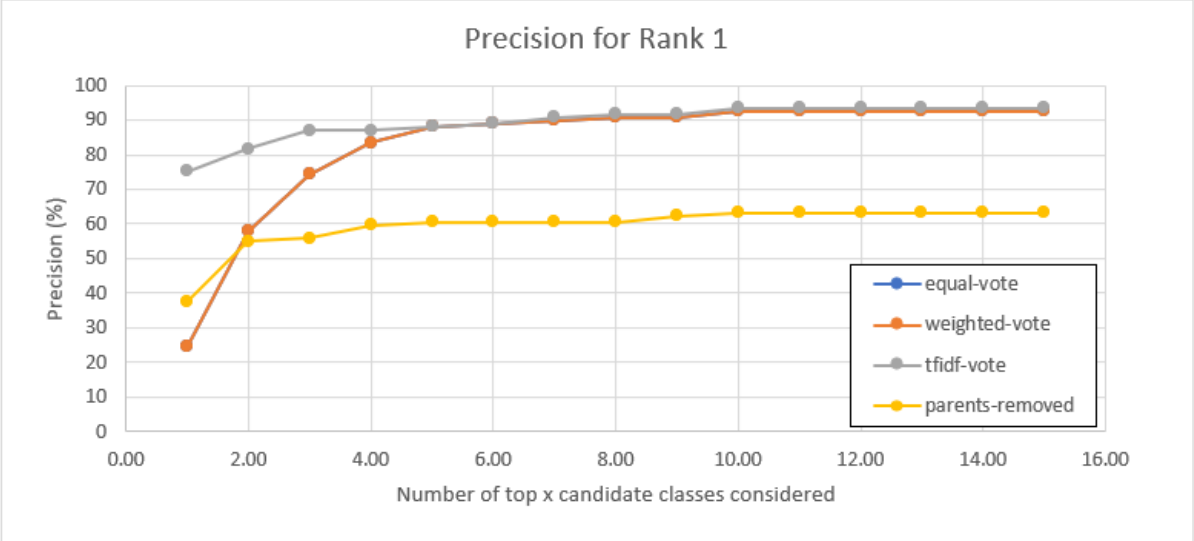

Figure 15. Comparative results of prediction based on voting with all candidate classes, with a subset of candidate classes removing parents and with tf-if for rank 1 entities

To further explain the above results, assume the following 2 lists of 5 candidate types for two different columns (taken from actual data that were processed in the pipeline). Only the top 6 candidates instead of the full list of candidate classes are presented for illustration purposes:

File 1	File 2
Filename: 58891288_0_1117541047012405958 Ground Truth: Film Column: 1 Equal Vote: ('Work', 20.77922078), ('Film', 15.58441558), ('Agent', 6.49350649), ('Organisation', 5.19480519), ('Location', 3.8961039), ('Place', 3.8961039) TF-IDF: ('Film', 0.16422260610432587), ('Work', 0.10112317149884212), ('Book', 0.048329385843283626), ('WrittenWork', 0.044680536816172754), ('City', 0.027370434350720983), ('Settlement', 0.02242739889841565)]	Filename: TOUGH_WEB_MISSP_celebrities Ground Truth: Person Column: 1 Equal Vote: ('Agent', 33.33333333), ('Person', 33.33333333), ('Artist', 8.33333333), ('Athlete', 8.33333333), ('BaseballPlayer', 8.33333333), ('MusicalArtist', 8.33333333) TF-IDF: ('Person', 0.23980503579351845), ('BaseballPlayer', 0.22801842317240886), ('Athlete', 0.134119826036175), ('Artist', 0.1199615729698919), ('MusicalArtist', 0.11490952115185567), ('Agent', 0.09754419253721236)

In the first file the ground truth is Film and is a subClassOf type Work which also appears in the list.

In the second file the ground truth is Person which is a parent class of Artist and Athlete that appear in the list. Furthermore, Artist is the parent of class MusicalArtist and Athlete is the parent class of BaseballPlayer.

Let's examine the effect of the different approaches:

Equal Vote: The equal vote will always favour the parent class since a **Film** in the majority of the cases is also a **Work.** In fact, there are certain retrieved entities that are not the classified as Film but rather as a Play, for instance, because the lookup returns the theatrical rather that the cinema version. As a result, this voting strategy may prove problematic when the expected type is a specific rather than a generic one. For the column in File 2 the equal vote strategy scores the Person class higher than its offspring (even though there is still an issue with very generic types such as Agent)

Removing of Parents: In this approach the list of candidates for File 1 will reduce to [Film, Agent, Organisation, Location, Place] and between them the highest scored type i.e. Film, is the correct one. However, in the case of the second file, by recursively removing the parents, the remaining candidates are reduced to [BaseballPlayer, MusicalArtist] none of which is generic enough to describe the type of the values in the target column. Therefore, this strategy of removing the parents will also fail to predict the correct class.

TF-IDF: The tfidf approach seems to be working in both cases and producing the correct prediction.

For file 1, given that a more generic class is most probable to appear more frequently in other columns as well (for instance columns describing Artwork, MusicalWork, WrittenWork, etc) the idf, which is the inverse log frequency of the type's appearance in other columns will be lower. As a result, it will

decrease the tfidf score of Work, allowing the class Film, which has a comparable tf in this column, to rise above. In this case, Film is the dominant offspring for this column and as a result tf-if will select that.

For file 2, however, no subclass of the parent Person appears to be dominant. As a result, even though the idf of person is lower compared to 'BaseballPlayer' and 'MusicalArtist' the term frequency of those 2 offspring is so low (there is no dominant between them) that the tfidf of either of them is still lower than that of the Person, which being a more generic class in this case, can better describe all the values in this column.

Returning back to the results in Table 3, similar to the two test series presented in 4.1.1 the results when only considering entities of rank 1 are superior to those when taking into account lower ranks. For instance, the precision drops by 1/3 from ~75% to ~54% when considering entities up to rank 5. This is the final confirmation that allowing anything other than the most relevant retrieved entity introduces more noise, on top of additional candidate types to process.

Moreover, the TF-IDF approach seems to converge with the voting approach both of which manage to come up with a relatively good top 5 that will contain the expected class for approximately 90% of the target columns.

4.1.3 CNN

The last round of experiments run for the CTA pipeline examine the efficiency of having pre-trained convolutional neural networks predict the type of a column. As a recap of what was discussed in section 3.5.3 a collection of cnn binary classifiers have been trained offline (one for each class that appears as a candidate type in any of the columns). For the purposes of the SemTab 2019 round1 dataset a total of 316 cnns were trained and saved offline since it takes several hour for all of them to complete. The training step follows the entity / type extraction step as the pipeline needs to have a list of types to train for as well as a list of positive and negative samples for the training.

For the prediction of the type of a column, the cell values are presented as samples for the input layer using either random selection or a sliding window approach. Instead of providing these samples as inputs to all 316 classifiers, we benefit from the previous lookup results (using TF-IDF which performed better than the voting) to focus on a smaller subset of cnns. This decision was made for practical reasons as it takes several minutes to predict whether a column belongs to the class the cnn is trained for and there are several columns that need to be annotated. At the same time the TF-IDF results suggest that the correct type for the ~90% of the columns can be found in the top 5 classes suggested by the TF-IDF. Therefore, focusing on a smaller subset should not deteriorate the performance of the system.

The following table illustrates the results from the type annotation using CNNs in 4 different setups. Using random sampling for input and assessing the top 5 and top 10 TF-IDF classes and using a sliding window for sampling for the same top5-top10 classes per column.

	Sliding Window		Random Sampling	
	Top 5 TF-IDF Types	Top 10 TF-IDF Types	Top 5 TF-IDF Types	Top 10 TF-IDF Types
1	67.89 (74.31)	66.06 (73.39)	68.81 (76.15)	65.14 (73.39)
2	78.9 (84.4)	77.06 (82.11)	79.82 (84.4)	77.06 (82.11)
3	84.4 (88.07)	81.65 (86.24)	84.4 (88.07)	81.65 (86.24)
4	86.24 (89.91)	86.24 (89.45)	86.24 (89.91)	84.4 (88.99)
5	88.07 (91.28)	88.07 (91.28)	88.07 (91.28)	88.99 (92.2)
6		90.83 (93.12)		90.83 (93.12)
7		92.66 (94.04)		93.58 (94.5)
8		93.58 (94.5)		93.58 (94.5)
9		93.58 (94.5)		93.58 (94.5)
10		93.58 (94.5)		93.58 (94.5)

Table 4. Results of the type annotation using cnns for the top 5 and top 10 types provided by TF-IDF

The results indicate that indeed limiting the number of cnns to 5 or 10 isn't hurting the performance of the pipeline since the results for top 5 or 10 are comparable. If anything, filtering out more irrelevant types actually improves the performance as the top 1 predicted class for sliding window sampling is the correct one for 67.89% when only 5 cnns are considered vs 66.06% will the number of accessed cnns double in size that.

Moreover, the two techniques (random sampling vs sliding window) seem to provide similar precisions. This was expected as there is typically no additional information in the order in which the rows appear in the tabular data, therefore selecting random cells is equivalent to selecting them sequentially.

Figure 16 places the CNN results in the context of the previous approaches discussed so far and appears to be better than all of them apart from the TF-IDF.

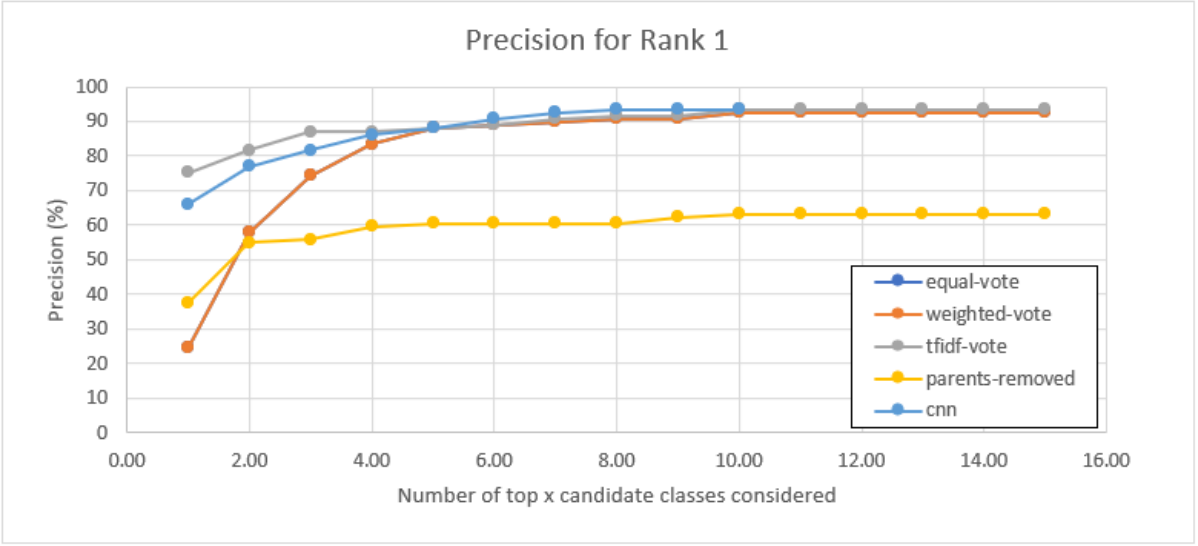

Figure 16. Comparative results of type annotation based on all approaches considered

Of course, the performance is boosted by the fact that the CNN is only considering the top5-10 candidate classes per column suggested by the TF-IDF which means that it is protected from choosing a type that is suggested by a small number of entities retrieved for a column. For a more unbiased approach, that took longer to execute, we allowed the pipeline to consider all the candidate type cnns suggested for each column and the results are illustrated below.

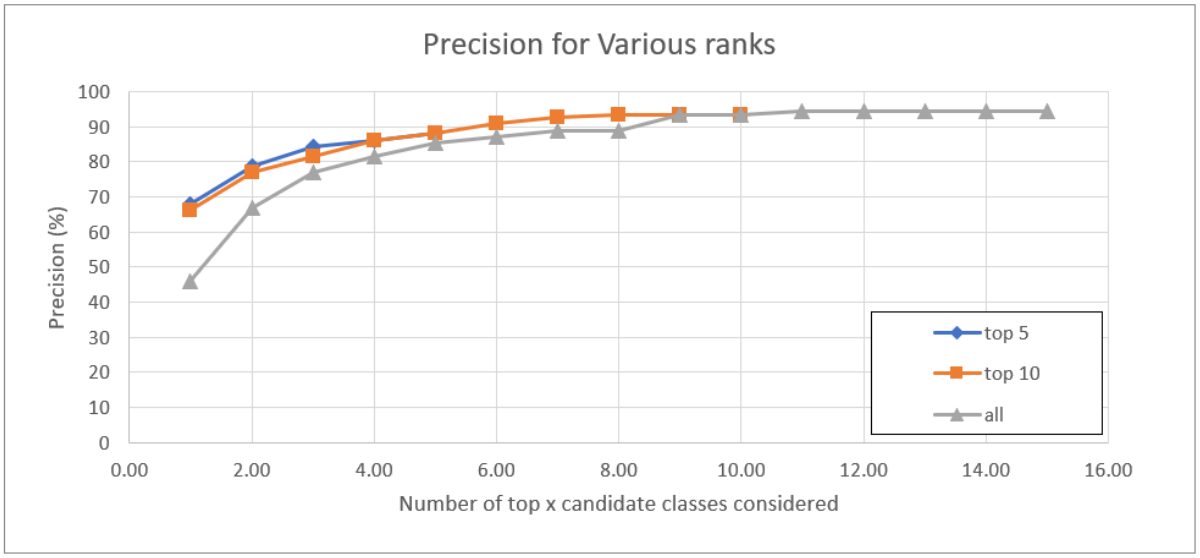

Figure 17. Comparative results of the CNN type annotation taking into account only the top 5 or top 10 classes or all candidate classes

The results in Figure 17 indicate that without the assistance of the TF-IDF to filter down on candidate classes, the precision of the CNN type annotation drop significantly from ~67% to ~46%. This precision is still superior to the voting strategy ~25% but not as good as the ~75% correct types proposed by TF-IDF.

4.2 Cell Entity Annotation (CEA) Results

There are 3 sets of experiments executed for the CEA task:

- Set1: In this set the predicted entity is the top 1 retrieved
- Set2: In this set the CTA type annotations are considered (i.e. the entity must have a type in the list suggested by the CTA). As implied in 4.1 the CTA types are presented as a sorted list of classes. Therefore, in the experiments conducted we either select the top1 class in the vector or the top two classes. The first retrieved entity with the right type is selected. If none of the retrieved entities has the right class, then there is not annotation for the cell. It is reminded that only up to 5 candidate entities are retrieved from the lookup API

- Set3: This is a hybrid approach where the logic of set1 is used as fallback when the logic of set2 fails to produce an entity. More specifically, we first try to identify an entity in the correct class and if that is not possible then we just get the top retrieved entity

The results of these three sets of experiments are presented in the table below

		Precision	Recall	F1_score
TOP1	Set1	69.69	67.54	68.60
	Set2	86.07	66.94	75.31
	Set3	75.21	72.89	74.03
TOP2	Set1	69.69	67.54	68.60
	Set2	79.77	67.93	73.38
	Set3	73.25	70.99	72.10

Table 5. Results of cell annotation for the 3 sets of experiments for the SemTab 2019 round1 data

Where the precision is determined (base on the SemTab evaluator as):

$$precision = \frac{len(correct\ entity\ annotations)}{len(annotated\ cells)} \quad (4.4)$$

And recall is determined as:

$$recall = \frac{len(correct\ entity\ annotations)}{len(target\ cells)} \quad (4.5)$$

And the f1 score it the product of the two.

$$f1\_score = \frac{2*precision*recall}{precision*recall} \quad (4.6)$$

Let's examine the results in the first 3 data rows that correspond to the top1 predicted class. As expected, the precision of set 2 is much higher than that of set 1 by almost 17% which indicates that bringing in information from the column annotation is beneficial for the CEA task. This comes at a cost of a drop in recall which, however, is quite minimal <1% between sets 1 and 2. Finally the hybrid approach of set 3 manages to increase the recall compared to the other two probably for cases where the predicted type was not the correct one however the precision dropped by almost 10%.

Similar are the results when we consider entities belonging to the top 2 (instead of top 1) classes. Set 2 still has the best precision and the hybrid set 3 has the best recall. However, the precision of set 2 when considering the top 2 classes has dropped by ~6% when compared to the precision when only considering the top CTA class. This means that for this dataset, the CTA pipeline, more often than not, managed to predict the expected type in the top 1 spot, therefore using that as a filter for the entities produces better results.

Another variation was attempted whereby instead of only considering entities belonging to the top class (or top 2 classes) of the CTA those classes and all of their parents were considered for the entity filtering. This approach was inferior to set 2 since more cells were being annotated however the quality of the

annotation was slightly worse. Therefore, as per the definition of the precision and recall used for this experiments, both of these measures dropped

	Precision	Recall	F1_score
TOP1 (CTA)	69.69	67.54	68.60
TOP1+parent classes (CTA)			

Finally, as mentioned in the appendix we also conducted experiments for another dataset with tough tabular data. For that dataset the CTA pipeline performed poorly and didn't manage to predict column types to the same precision as it did for the SemTab 2019 presented in this section. Even so, considering the CTA output (set 2) still gives a better precision than just relying on the results from the lookup api (Table 6). However, in this case, relaxing the threshold on the classes that we consider valid proves beneficial with the precision only dropping by 0.5% between top 1 and top2 and the recall also increasing by almost 0.5%. The F1 scores of the top1 and top2 are comparable but choosing the second option makes the solution more generic

		Precision	Recall	F1_score
TOP1	Set1	77.34	70.43	73.72
	Set2	84.17	73.60	78.53
	Set3	81.87	74.55	78.03
TOP2	Set1	77.34	70.43	73.72
	Set2	83.84	74.07	78.65
	Set3	81.98	74.65	78.14

Table 6. Results of cell annotation for the 3 sets of experiments for the SemTab 2020 tough tables

5 Discussion

This chapter aims to elaborate further on the results obtained by the experiments conducted as part of this project and compare them with the original objectives that were set for the project. More specifically, the pipelines that have been implemented employee a number of approaches that tackle two out of the three tasks of the SemTab challenges, i.e. the column type annotation and the cell entity annotation. Before going to the details of the results on the above tasks the first section of this chapter revisits the pre-processing or offline steps.

5.1 Offline pre-processing

The decision to query the reference knowledge base offline and store the results in a dictionary structure (entity / type retrieval) improved the performance of the system and removed the need to be connected to the internet while conducting the experiments. Retrieving entities for a cell value was therefore reduced from a call to an api to a query in a dictionary which is much faster not only due to the reduced cost of the activity itself but also the fact that it protects the pipeline from making the same request more than once. Since querying the Knowledge Graph using the same input parameters always retrieves the same results the decision to store these results offline has no impact on the CTA/CEA precision itself.

On top of the entity retrieval from the reference KG, the other computational expensive component of the pipeline (only used in CTA) is the training of the CNN. Training of the CNNs for a sample size of 30 words per vector and 200 elements per word took between 15-24 hours for the SemTab 2019 round 1 and SemTab 2020 Tough Tables. Training the models once and reusing them in the various combinations needed by the experiments also saved a lot of time for the pipeline. It should be noted that at time this offline training had to be performed multiple times (e.g. different sample sizes, different network architectures 2-3 convolutional layers) however once the best setup was selected the rest of the experiments were conducted using the offline trained models

Finally, although the impact of creating predefined structures of data was not prominent as part of this project, since data from SemTab appeared in the same format, it should be noted that this abstraction from the input format is a useful engineering step if the pipeline is to be used for other tabular data

5.2 CTA

For the CTA task the pipeline dealt with the original objective in a more flexible way. Instead of suggesting just one time for each column (as the challenge suggested) the pipeline provide a sorted list of possible types. This is quite important for analysts doing exploratory data analysis since it provides them with more options they can consider when looking at the metadata for a column. That way a final step could be to either blindly select the top one suggested class (which collapses the list to the outcome expected by SemTab) or have the outcome manually inspected by a subject matter expert.

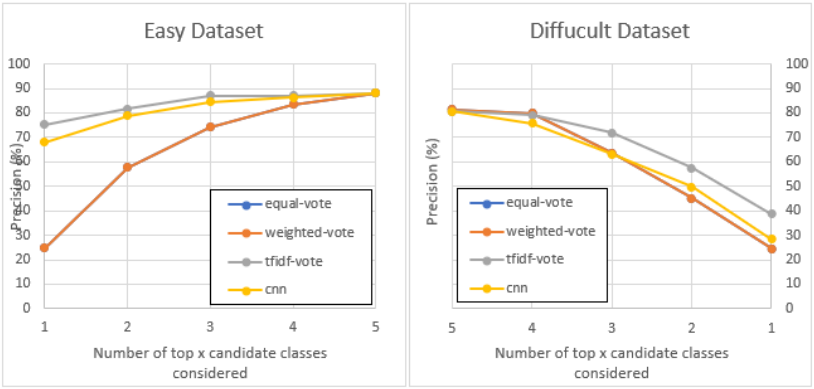

Figure 18. Results of type annotation for the SemTab 2016 round 1 (easy dataset) 2020 tough tables (difficult dataset)

The importance of the above statement is illustrated in Figure 18 that presents the CTA results for the pipeline when applied to an easy dataset vs a difficult dataset containing a lot of noise. In this case considering the top type leads to a significant drop in the precision from ~75% for the TF-IDF approacj to ~40%. However, if we consider the top 5 suggested types then the precision for the two datasets is comparable (~88% vs ~81% respectively).

Finally, it is clear from the results that in all cases applying the TF-IDF logic on the candidate types for the columns is outperforming all other CTA approaches implemented as part of this project. This strategy is also not as computationally expensive as the CNNs. The CNN approach could still yield good results however the problem is with the training of the CNN since the models rely significantly on the quality of the training set.

5.3 CEA

The CEA task seems to benefit greatly when combined with the results from the CTA pipeline even when the CTA hasn't managed to identify the correct type with a high precision. This means that when filtering out retrieved entities that do not belong to the expected class, the pipeline has a better chance of identifying the correct candidate compared to blindly selecting the top result retrieved from the API. Using the top result provides an adequate safety net for when none of the top 5 retrieved entities are within the expected type(s) however such results could be marked for manual inspection as they are less likely to be correct.

The precision on the CEA task has proven more stable across different dataset in comparison to the CTA task. In detail, the best precision achieved by the CTA pipeline for the easy dataset (~75%) dropped almost by half (~39%) while at the same time the respective CEA f1 scores remained comparable (~86% vs 84%)

5.4 Results and literature comparison

The tasks of CTA and CEA are part of the SemTab challenge that was first introduced in 2019 (Semantic Web Challenge on Tabular Data to Knowledge Graph Matching). Several teams took part in the

challenge which comprised 4 rounds with most participants (17) entering the 1st round. The results of the participants were evaluated using similar functions to the ones used as part of this project (Codes for the AIcrowd evaluator) and presented in Table 7. The evaluation functions had to be rebuilt for this project since one of the objectives was to evaluate a set of classes for the CTA rather than just one however the logic followed is the same so the results can be comparable. The results shown below for the CTA task are taken from the TF-IDF approach, top1 suggested class considered only rank1 candidate entities which was the best performant approach presented in Chapter 4. For the CEA the results of this project correspond to the CEA set 2 which only considers entities that are of the correct type as suggested by the CTA pipeline which was also the one that was performing the best.

SemTab 2019 - Round 1					
CEA Task			**CTA Task**		
Team	F1-Score	Precision	**Team**	**F1-Score**	**Precision**
Team_sti	1	1	*MTab*	1	1
MTab	1	1	*VectorUP*	1	1
Team_DAGOBAH	0.897	0.941	*Team_sti*	0.929	0.933
Tabularisi	0.884	0.908	*LOD4ALL*	0.85	0.85
bbk	0.854	0.845	*IDLab*	0.833	0.833
LOD4ALL	0.852	0.874	*ADOG*	0.829	0.851
Tabularisi 2	0.816	0.851	*Tabularisi*	0.825	0.825
PAT-SEU/zhangyi	0.794	0.804	*Siemens Munich*	0.754	0.7
Project_ZD	0.753	0.861	Project_ZD	0.7523	0.7523
VectorUP	0.732	0.733	*Team_DAGOBAH*	0.644	0.58
ADOG	0.657	0.673	*f-ym*	0.527	0.358
IDLab	0.448	0.627	*ahmad88me*	0.485	0.581
			It's all semantics	0.192	0.192
			kzafeiroudi	0.033	1

Table 7. Comparative results for the CEA and CTA tasks for SemTab 2019 round 1

The approach developed as part of this project ranks somewhere in the middle compared to the top ranking systems proposed for the SemTab challenge. Some of the systems that outperform the project's approach like MTab (Nguyen et al. 2019) or IDLab (Steenwinckel et al. 2019) rely heavily on text processing of the cell values. This is something that we tried to avoid as part of this implementation since it is computationally expensive and relies less on machine learning techniques. It does indicate however that cleaning the data before feeding them into a CTA or CEA pipeline greatly improves the accuracy of the system. Even so, the results for both the CEA and CTA annotations made by the pipelines of this project are comparable in terms of precision and f1_score (it is reminded that for the CTA task these two numbers are identical since we annotated every single column that is in the target)

Moreover, the implementation of TF-IDF for the CTA task that was inspired by DAGOBAH (Chabot et al. 2019) seem to have performed better that the originally suggested system (Team_DAGOBAH)

Finally, to prove the robustness of the implementation the CTA and CEA pipeline were applied on a set of tough tables from the SemTab 2020 challenge still manage to produce good results that are comparable to other participants and illustrated in Table 8.

SemTab 2020 - Tough Tables					
CEA Task			CTA Task		
Team	F1-Score	Precision	Team	F1-Score	Precision
DAGOBAH team	0.945	0.946	JenTab	0.46	0.468
Project_ZD	0.79	0.84	DAGOBAH team	0.422	0.424
GBMTab/seudocode	0.692	0.692	Project_ZD	0.3866	0.3866
JenTab	0.607	0.669	Magic	0.159	0.628
Magic	0.184	0.506			
KEPLER-aSI	0.11	0.644			

Table 8. Comparative results for the CEA and CTA tasks for SemTab 2020 tough tables

The above results indicate that the objectives set for this project to implement a system that accurately enhances tabular data information have been by providing two pipelines for column and cell annotation that systematically produce good results considering the input

9.5%(1,150-1,400)

6 Evaluation, Reflections, and Conclusions

10%(1,200-1,500)

Glossary

Term	Description
Knowledge Base	
Class/ Type	
Entity	
Convolutional Neural Network (CNN)	

References

Appendix