Question Embeddings for Semantic Answer Type Prediction

Eleanor Bill[1] and Ernesto Jiménez-Ruiz[2]

[1, 2] City, University of London, London, EC1V 0HB

Abstract. This paper considers an answer type and category prediction challenge for a set of natural language questions, and proposes a question answering classification system based on word and DBpedia knowledge graph embeddings. The questions are parsed for keywords, nouns and noun phrases before word and knowledge graph embeddings are applied to the parts of the question. The vectors produced are used to train multiple multi-layer perceptron models, one for each answer type in a multiclass one-vs-all classification system for both answer category prediction and answer type prediction. Different combinations of vectors and the effect of creating additional positive and negative training samples are evaluated in order to find the best classification system. The classification system that predict the answer category with highest accuracy are the classifiers trained on knowledge graph embedded noun phrases vectors from the original training data, with an accuracy of 0.793. The vector combination that produces the highest NDCG values for answer category accuracy is the word embeddings from the parsed question keyword and nouns parsed from the original training data, with NDCG@5 and NDCG@10 values of 0.471 and 0.440 respectively for the top five and ten predicted answer types.

Keywords: Semantic Web, knowledge graph embedding, answer type prediction, question answering.

1 Introduction

1.1 The SMART challenge

The challenge [1] provided a training set of natural language questions alongside a single given answer category (Boolean, literal or resource) and 1-6 given answer types. The provided datasets contain 21,964 (train - 17,571, test - 4,393) questions. The challenge was to achieve the highest accuracy for answer category prediction and highest NDCG values for answer type prediction. The methodology described in this paper attempts to achieve this using knowledge graph and word embeddings alongside question parsing and multi-layer perceptron models.

The target ontologies and pre-computed knowledge graph embeddings used in this project are built on the free and open knowledge graph DBpedia. DBpedia is a knowledge graph consisting of grouped information from various Wikimedia projects.

It can be queried with SQL-based query languages such as SPARQL. The English language version classifies 4.2 million items with a variety of semantic types such as peoples and places [2]. The pre-trained word embeddings from fastText were also used in the question answering system [3].

1.2 Question answering

Question answering is a computer science field concerned with answering questions posed to a system in natural language. The goal of a question answering system is to retrieve answers to natural language questions, rather than full documents or relevant passages as in most information retrieval systems. Open domain QA systems are commercially available in products such as Apple's Siri and Amazon's Alexa [4] while closed domain QA systems are used in specialized fields, such as medical diagnosis.

Question answering systems rely on very large datasets of collated information, such as Wikidata – a knowledge base which is parsed from Wikipedia. These knowledge bases can be structured in different ways: relational databases, or in Wikidata's case, as a knowledge graph, to ensure maximum efficiency and to allow for more meaningful links within the objects in the knowledge base.

Many modern QA algorithms learn to embed both question and answer into a low-dimensional space and select the answer by finding the similarity of their features [5].

A question answering process ultimately needs to fulfil three steps: to parse the natural language question to allow for meaningful understanding of what the user is asking, to retrieve the relevant facts from the knowledge base, and to deliver the facts in an appropriate answer [6].

1.3 Knowledge graphs

Knowledge graphs are a type of knowledge base: information is represented in a data structure. A knowledge graph is an abstract representation of information in the form of a directed labeled multi-graph, where nodes represent entities of interest and edges represent relations between these entities. Facts join these rules in the graph in the form of subject-predicate-object triples (or head, relation, tail) e.g., London, capital, United Kingdom [4] where the fact represented by this link in the knowledge graph is that London is the capital of the United Kingdom.

The meaning, or semantics, of the data in the knowledge graph is encoded alongside the data, in the form of the ontology. An ontology is a set of logical rules that allow inference about the data in the knowledge graph that govern the object types and relationships between entities. This inference can allow implicit information to be derived from the explicit information in the graph. If knowledge graphs are provided with new information or a change in ontology, they are able to apply the rules to the new information, and this can in turn increase accuracy in terms of e.g., question answering classification.

Knowledge graph embeddings differ from each other based on three criteria: the choice of the representations of entities and relationships, the scoring function and the loss function. Representations of entities and relationships are most commonly ex-

pressed using vector representations of real numbers, but common alternatives are matrices and complex vectors. The subject-predicate-object triple can be represented as a Boolean tensor where Q can equal either True or False. The subject and object entities and the predicate are mapped to their latent representations and then to probability $P \in [0, 1]$. The scoring function estimates the likelihood of the subject-predicate-object triple by aggregating the information coming from it. The loss function defines the element of the system that needs to be minimised during knowledge graph embedding model training in order to provide the best result on test data [7].

2 Related work

Answer type prediction systems take form in many ways and pull on many parts of the question. Bogatyy [8] applies a transition-based neural network to parse syntactic structure of a question and using this infers the answer type. To do this, Bogatyy's model annotates parts of the query using a type inventory (specifically Freebase, the precursor to Wikidata) to represent the types they may be associated with. For example, 'who' is annotated with 'people'. This is done both at a coarser level - as with 'who' - and at a finer level, as with a phrase such as 'horror movie'. The syntactic structure of the query is then annotated using globally normalized transition-based neural network parser. This model outperforms a logistic regression baseline that only uses the type inventory and doesn't take into account syntactic structure of the queries.

Answer type prediction can be used to improve semantic parsing, as in Yavuz et. al [9]. Their model uses a bidirectional LSTM model to infer answer types in conjunction with semantic parsing, which maps a natural language question into its semantic representation – logical form that relates to meaning stored structurally in knowledge bases. It does this by recursively computing vector representations for the entities present in the question. Syntactic features are also used here in the form of dependency trees, before constructing a bidirectional LSTM neural network over this final representation of the question and predicting the answer type. Again, a combination of different methods, models and parsers improve the accuracy of the question answering system.

Knowledge bases may suffer from lack of completion and potentially outdated or missing information. To counteract this, Sun. et. al [10] use knowledge bases alongside information mined directly from the Web to successfully improve question answering F1 score. Another source of inaccuracy from lack of information may be due to infrequency of answers and answer types within test data. Murdock et. al [11] adapted to the challenge associated with finding specialist answers to questions that featured on the quiz show Jeopardy. They predicted fine-grain answer types using DeepQA to analyse the question and produce candidate answers independent of answer type before using type coercion to choose the candidate answer most likely to be correct.

3 The answer type prediction system

The steps within the development of this system involve the parsing of the natural language questions, the reuse of vectors using knowledge graph embeddings and/or word

embeddings and the parsed parts of the questions, the training of MLP classifiers using these vectors and the evaluation of the classifiers. The system is implemented in Python, using pre-existing libraries and knowledge bases: fastText [12], DBpedia [2] and SPARQL endpoint [13].

The system uses word and knowledge graph embeddings on the parsed questions - the word embeddings of the key 'wh' part of the question (what/when/why etc.), word embeddings of the parsed nouns, knowledge graph embeddings of word embeddings of the parsed noun phrases and word embeddings of the types of the knowledge graph entities. A knowledge graph access interface [16] is used to query DBpedia via SPARQL, the standard query language for Linked Open Data and semantic graph databases. Multi-layer perceptron classifiers, one per type/category, are trained on the concatenated vector, and the similarly vectorised test data is given a probability estimate for each type by the related classifier. The top ten types in terms of probability are the type results. The categories are found in a similar way, with only the top type in terms of probability provided for evaluation.

Additional heuristics are also applied to the system to increase performance. Some of which are provided by the SMART challenge [1]:

- If the category is "resource", answer types are ontology classes from the DBpedia ontology.
- If the category is "literal", answer types are either "number", "date", "string" or "Boolean" answer type.
- If the category is "Boolean" the answer type is always "Boolean".

3.1 Question parsing

The question is first parsed for nouns using the Natural Language Toolkit (NLTK) [14] word_tokenize() and pos_tag() functions, which splits the question up into words and tags them with parts-of-speech taggers. The NN (common nouns), NNP (proper nouns), NNS (common noun plural forms) and NNPS (proper noun plural forms) tags are all collectively treated as nouns and added to a noun list for each question. Noun phrases (names etc.) are then parsed using the noun phrases function in TextBlob, a Python library for processing textual data.

Finally, the ‘wh’ part of the question is extracted ('why', 'where', 'when', 'how', 'which' etc.) using a search function. This is done to extract the option most likely to be important to the question as judged by the list above, e.g., ‘where’ is prioritised above ‘when’, ‘when’ above ‘how’. This encompasses all the questions so the majority of them return a question keyword, while acknowledging how some of the questions were worded. For example, in the question “Where is the Empire State Building?” the ‘where’ is most important, whereas in “Is the Empire State Building in New York?” the ‘is the’ is the most important, although both questions contain ‘is the’. This list was generated by iteratively looking through the questions in the question data to check the existing list covered them, and if it didn’t a new keyword from that question was added to the list. The ordering of the list was subjective.

3.2 Building positive and negative samples

Additional positive and negative training samples are created in order to optimize performance of the classifiers. Negative training samples are created in two ways. The first: by shuffling the answer categories and types independent of the questions to get a random answer category/type associated with the questions. This is a coarse way of generating negative samples, and in likelihood leads to a less precise and accurate classifier than swapping out answer categories and types close to the original, according to the class hierarchy.

The second method of creating negative samples involves using sibling types of types associated with the parsed question, accessed through the ontology. For the example type ‘gymnast’, the ancestor type ‘athlete’ is accessed and then a disjoint type, descendant from the ancestor such as ‘basketball player’ is returned as a sibling type. This is an example of more fine-grained negative samples, which are useful for fine-tuning the classifiers.

Both methods of creating negative samples are used in the system, future work could use these individually to compare which has a greater effect on the results of the classification accuracy.

Additional positive training data is created by getting alternative but similar entities to those associated with the parsed question, using the SPARQL endpoint. For example:

- The original entity: http://dbpedia.org/resource/Serena_Williams
- The associated type: http://dbpedia.org/ontology/TennisPlayer
- The similar entity: http://dbpedia.org/resource/Roger_Federer

This is implemented using the endpoint class in the KG access framework [16], which uses an API to access related entities and classes in the knowledge graph. Here, the getEntitiesForDBPediaClass() function is used.

The vectors used to train the classifier consist of several concatenated vectors based on the above parsed parts of the question and word and knowledge graph embedding applications. The structure of the final concatenated vector is as follows:

- First position: the word embedding of the ‘wh’ question part.
- Second position: the word embedding of the set of nouns.
- Third position: the knowledge graph embedding of the noun phrases.
- Fourth position: the word embedding of the types (e.g. 'http://dbpedia.org/ontology/OfficeHolder') of the found KG entities.

3.3 Training and using classifiers

A separate binary classifier is built for each semantic type and category in the data, the type in this case being the most fine-grained type associated with the question. The system then uses a one-vs-all strategy, in which binary classification for each task is used collectively as multi-class classification. Given a classification problem with N possible classes, N binary classifiers are trained - one for each possible outcome. These classifiers are trained on the previously described positive and negative specific and general samples built using pre-computed word and knowledge graph embeddings on the parsed questions, in the form of a stack of word vectors.

The model used in this project to train the classifiers is a multi-layer perceptron, a feed-forward, supervised neural network composed of several layers of neurons. An MLP model has a minimum of three layers: input, output and at least one hidden layer of neurons which extract appropriate features and weight components of the input layer. An MLP can be made more complex by increasing the number of hidden layers, should that be required.

Once all classifiers are trained, test data can be passed through the classifiers and generate probabilities of the test data belonging to that category/type. The higher probability categories/types are considered those most likely to be associated with that test data point.

Heuristics In addition to the classifiers, two sets of heuristics are applied to the system: the first applying rules based on the question keyword part of the question, and the second based on the rules described by the SMART challenge described above [1]. This helped improve the performance of the classifiers.

3.4 Evaluation

The evaluation of the system uses the evaluation script provided by the challenge. Answer category predication is considered as a multi-class classification problem and accuracy score as the key performance metric, as there are only three answer categories and little ambiguity between them. Accuracy is calculated via a direct comparison between the predicted type as returned by the system and the actual type.

The pre-written evaluation code uses hierarchical target type identification for assessing accuracy, in which the system provides a list of ten most likely target types and these are evaluated by judging the semantic distance between them and the actual types in the test data, applying decay as the ranking increases. The specific metric used is the metric lenient NDCG@k with a linear decay [15]. This allows several ranked potential types to be evaluated as a whole, giving a tailored overview of the accuracy of the classifier.

4 Experiments

4.1 System parameters

System parameters to consider included length of the vector, amount of training data per classifier, ratio of positive and negative samples per classifier. The lengths of the vectors were 300 floats long for word embeddings and 200 floats long for the knowledge graph embeddings, the concatenated vectors being however long the vectors were individually combined. The amount of training data depended on how much was available for each type/category but was always more than or equal to 22 vectors, due to the creation of additional training data. Ideally this would have been greater, but the time taken to create more positive vectors constrained this. The ratio of positive and negative samples per classifier was always a 1:1 ratio.

The hyperparameters of any MLP classifier are the number of hidden layers, the momentum and the learning rate. The number of hidden layers was set to the default of

the scikit-learn MLP classifier, which was a single hidden layer with 100 neurons. The momentum and learning rates were also the defaults. The maximum number of iterations (in case of convergence taking longer) was set to 300 for each classifier.

4.2 Original vs. additional training data

Comparing the results from just using the original training data versus using the original training data with additional positive samples, and new negative samples allowed evaluation of the quality of the manufactured training data and discussion on whether the original training data was sufficient. The original training data was fairly limited in terms of spanning the distribution of types it did - with 17,528 rows and 310 types sometimes a type only had one or two samples with which to train a classifier. The creation of new positive training data allowed this to be expanded somewhat, although due to time limitations not as much additional training data was created as originally desired. 158,264 new positive samples were created and 28,557 new negative samples. While the overall data pot was still unbalanced, this increase in training data allowed each classifier to be trained on a balanced sample of positive and negative vectors that was larger than could be used with just the original training data. The more relevant negative samples also allowed the classifiers to be more fine-tuned.

4.3 Vector structure comparisons

Tests were carried out in which different parts of the vectors were used to train the classifiers. This allowed for a better understanding of the quality of different parts of the vectors, for example the word embedding of the noun phrases was relatively unsuccessful compared to the knowledge graph embedding of the noun phrases or word embedding of the nouns, so this vector was not used in the final classifier training experiments and was deleted from the final concatenated vector. The results of these tests are in the tables below.

5 Results

Table 1. Original training data results.

Vector component	Category accuracy	Type NDCG@5	Type NDCG@10
Word embedded ‘wh’	0.617	0.334	0.313
Word embedded nouns	0.529	0.427	0.402
KG embedded noun phrases	0.793	0.300	0.278
Word embedded types of KGE entities	0.581	0.405	0.379
WE ‘wh’ + WE nouns	0.487	0.471	0.440
WE ‘wh’ + KGE noun phrases	0.617	0.337	0.314
WE nouns + KGE noun phrases	0.546	0.412	0.388

WE ‘wh’ + WE nouns + KGE noun phrases	0.488	0.471	0.440
WE ‘wh’ + KGE noun phrases + WE types of KGE entities	0.574	0.385	0.359
All vectors combined	0.512	0.441	0.414

Table 2. Additional training data results.

Vector component	Category accuracy	Type NDCG@5	Type NDCG@10
Word embedded ‘wh’	0.616	0.350	0.330
Word embedded nouns	0.617	0.343	0.318
KG embedded noun phrases	0.616	0.336	0.334
Word embedded types of KGE entities	0.598	0.400	0.380
WE ‘wh’ + WE nouns	0.570	0.382	0.359
WE ‘wh’ + KGE noun phrases	0.617	0.342	0.318
WE nouns + KGE noun phrases	0.606	0.359	0.340
WE ‘wh’ + WE nouns + KGE noun phrases	0.593	0.365	0.342
WE ‘wh’ + KGE noun phrases + WE types of KGE entities	0.580	0.374	0.351
All vectors combined	0.562	0.396	0.372

6 Discussion and conclusions

6.1 Discussion of results

The better the category accuracy results are, the worse the type NDCG values, with a negative correlation of -0.89. This is unexpected, as one would expect a correct answer category prediction to facilitate a more accurate answer type prediction. It also makes it difficult to know which vector combination performs best overall. Also unexpected is that the additional created training data seems to produce no benefit in terms of classifier accuracy gain. The answer category prediction accuracy from the classifiers trained on the additional training data is a little higher than those trained on only the original training data, but the answer type prediction results are better with the original training data.

Individually, the classifiers trained on knowledge graph embedded noun phrases vectors from the original training data produced the highest accuracy for answer category prediction, while the classifiers trained on the word embeddings from the parsed question keyword and nouns parsed from the original training data produced the highest NDCG values for answer type prediction. Across all combinations, the NDCG@5 values are always higher than the NDCG@10 values - this makes sense considering the more types predicted by the system, the less relevant they become as their ranking gets lower.

The final test set results submitted to the task returned an answer category prediction accuracy of 0.79 and answer type NCDG values of 0.31 and 0.30 respectively. These

results were gathered from their own test set so are not the same as those in the tables included here. The category prediction accuracies for the other participants in the challenge ranged from 0.74 to 0.98, while the NDCG values ranged from 0.54 to 0.79. For category accuracy, this places this system within a good range of results, while the type prediction under-performed compared to other submissions.

6.2 Further conclusions and future work

Given longer to work on the project, I would experiment with optimising hyperparameters of the models and experimenting with different models to see if performance improved.

Due to the generalised question data, the wide ontology and the large knowledge base it uses, I believe this system could be similarly applied to other natural language questions and achieve similar results. The results are reproducible across different test data sets.

There are several unanswered questions pertaining to the system results I would like to investigate further, for example the ineffectiveness of additional training data in terms of performance, and why an increase in answer category prediction accuracy is correlated with a decrease in NDCG values for the answer type prediction. A good starting point could be to compare the strategies that were used to generate more samples in terms of their effects on the accuracy of the classification system.

References

1. Mihindukulasooriya, N., Dubey, M., Gliozzo, A., Lehmann, J., Ngonga Ngomo, A.-C., Ricardo, U., SeMantic AnsweR Type prediction task (SMART) at ISWC 2020 Semantic Web Challenge. CoRR/arXiv/abs/2012.00555 (2020).
2. DBpedia, https://wiki.dbpedia.org/about, last accessed 2020/10/20.
3. Joulin, A., Grave, E., Bojanowski, P., Douze, M., Jégou, H., Mikolov, T.: FastText.zip: Compressing text classification models. arXiv:1612.03651 [cs] (2016).
4. Gupta, P., Gupta, V.: A Survey of Text Question Answering Techniques. In: International Journal of Computer Applications 53, no. 4: 1–8 (2012).
5. Cortes, E., Woloszyn, V., Barone, D. A. C.: When, Where, Who, What or Why? A Hybrid Model to Question Answering Systems. In: Computational Processing of the Portuguese Language, 136–46. Lecture Notes in Computer Science. Cham: Springer International Publishing (2018).
6. Liu, K., and Feng, Y.: Deep Learning in Question Answering. In: Deep Learning in Natural Language Processing, p. 185–217. Singapore: Springer, (2018).
7. Bianchi, F., Rossiello, G., Costabello, L., Palmonari, M., Minervini, P. 'Knowledge Graph Embeddings and Explainable AI'. ArXiv:2004.14843 [Cs] (2020).
8. Bogatyy, I., Predicting answer types for question-answering, (2016).
9. Yavuz, S., Gur, I., Su, Y., Srivatsa, M. & Yan, X., Improving Semantic Parsing via Answer Type Inference, In: Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, 149-159, (2016).

10. Sun, H., Ma, H., Yih, W., Tsai, C., Liu, J., Chang, M., Open Domain Question Answering via Semantic Enrichment, In: WWW '15: Proceedings of the 24th International Conference on World Wide WebMay, 1045–1055, (2015).
11. Murdock, J.W., Kalyanpur, A., Welty, C., Fan, J., Ferrucci, D. A., Gondek, D. C., Zhang, L. & Kanayama, H., Typing candidate answers using type coercion, In: IBM J. Res & Dev. Vol. 56, (2012).
12. Bojanowski, P., Grave, E., Joulin, A., Mikolov, T., Enriching Word Vectors with Subword Information, arXiv:1607.04606, (2016).
13. W3.org, SparqlEndpoints, https://www.w3.org/wiki/SparqlEndpoints, last accessed 2020/11/30.
14. Bird, S., Klein, E. & Loper, E., Natural Language Processing with Python, O'Reilly Media (2009).
15. Balog, K., Neumayer, R., Hierarchical target type identification for entity-oriented queries. In: Proceedings of the 21st ACM International Conference on Information and Knowledge Management - CIKM ’12. (2012).
16. Ruiz, E.J., Tabular Data Semantics for Python, https://github.com/ernestojimenezruiz/tabular-data-semantics-py, last accessed 2020/10/20.