A Sample *Proceedings of the VLDB Endowment* Paper in LaTeX Format[*]

Ben Trovato[†]
Institute for Clarity in Documentation
1932 Wallamaloo Lane
Wallamaloo, New Zealand
trovato@corporation.com

G.K.M. Tobin[‡]
Institute for Clarity in Documentation
P.O. Box 1212
Dublin, Ohio 43017-6221
webmaster@marysville-ohio.com

Lars Thørväld[§]
The Thørväld Group
1 Thørväld Circle
Hekla, Iceland
larst@affiliation.org

Lawrence P. Leipuner
Brookhaven Laboratories
Brookhaven National Lab
P.O. Box 5000
lleipuner@researchlabs.org

Sean Fogarty
NASA Ames Research Center
Moffett Field
California 94035
fogartys@amesres.org

Charles Palmer
Palmer Research Laboratories
8600 Datapoint Drive
San Antonio, Texas 78229
cpalmer@prl.com

ABSTRACT

The abstract for your paper for the PVLDB Journal submission. The template and the example document are based on the ACM SIG Proceedings templates. This file is part of a package for preparing the submissions for review. These files are in the camera-ready format, but they do not contain the full copyright note. Note that after the notification of acceptance, there will be an updated style file for the camera-ready submission containing the copyright note.

[*]for use with vldb.cls
[†]Dr. Trovato insisted his name be first.
[‡]The secretary disavows any knowledge of this author's actions.
[§]This author is the one who did all the really hard work.

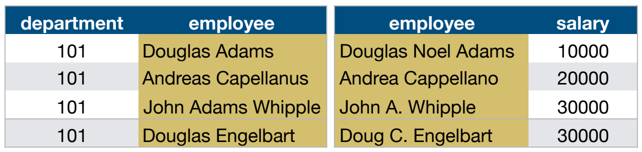
1. INTRODUCTION

Joining two datasets is a key step in preparing the data for subsequent operations such as performing business analytics, building predictive models etc. Data management systems have largely focussed solely on equi-joins, which is based on exact equality of strings or numeric values. However, in many circumstances, exact equality is inadequate because the same entity may in many cases be expressed with slight variations of the same name, as shown in Figure 1 (e.g., *Douglas Adams* and *Douglas Noel Adams*). Although Figure 1 illustrates the problem for people's names, the same problem occurs for other entity types, such as company names, addresses, states in countries, and countries.

One common technique to automating joins of the sort described in Figure 1 is string matching. String matching approaches use variants of string similarity measures such as edit-distance, Jaro-Winkler and TF-IDF (e.g., [2]) to perform matching. Typically to scale the problem to a large number of rows in each table, a filtering or blocking strategy is used to reduce the number of pairs to be considered for string matching. For instance, prefix, string length, or suffix based filtering are often applied to the strings in the two tables to identify potentially joinable pairs that need to be matched. For the examples shown in Figure 1, the prefix filtering step will ensure that *Douglas Adams* will be compared only with *Douglas Noel Adams* and *Doug C. Engelbart* to determine their string similarity, assuming the prefix used for filtering is *Doug*.

More recently, data driven approaches have emerged as a powerful alternative to string matching techniques for the join problem described in Figure 1. Data driven approaches mine patterns in the data to determine the 'rules' for joining a given entity type. One example of such an approach is illustrated in [4], which determines which cell values should be joined based on whether those cell values co-occur on the same row across disparate tables in a very large corpus of data. Such a system can perform 'semantic joins' such as mapping country names to country codes in new unseen tables. Another example of a data driven approach is work by [8] that uses program synthesis techniques to learn the right set of transformations needed to perform the entity matching operation, based on a numerous examples. Unlike string matching algorithms, which apply the same generic operator to a cell value regardless of entity type, the approach outlined in [8] can learn different programs to transform cell values for different entity types.

In this paper, we propose a novel data driven approach to the problem of joining semantically different representations of data. Our approach relies on building supervised deep learning models to automatically learn the correct set of transformations needed to compute the equality of two cell values. Because a deep learning model can pick up the appropriate features of what should be used for matching from correlations in the data, it should in theory be able to generalize better across join problems than the data driven approaches outlined in [4]. Conceptually, our approach is similar in spirit to the approach described in [8], but we use deep neural networks to learn the right function for the join operation instead of synthesizing different programs for different entity types. The advantage of building such a function over the approach described in [8] is that we can use this function in novel ways to completely eliminate the filtering that is typically needed to identify the set of potentially joinable rows, as we describe below.



department	employee
101	Douglas Adams
101	Andreas Capellanus
101	John Adams Whipple
101	Douglas Engelbart

employee	salary
Douglas Noel Adams	10000
Andrea Cappellano	20000
John A. Whipple	30000
Doug C. Engelbart	30000

Figure 1: Example of a join problem

Our specific solution to the join problem involves building a deep neural network that learns to produce a small distance estimate for elements of a name pair that represent the same entity (e.g., *Douglas Adams-Douglas Noel Adams* should produce a distance estimate d that is closest to 0), and a much larger distance estimate for elements of the name pair that do not represent the same entity (e.g., *Douglas Adams-John Adams* should produce a distance estimate d that is greater than some margin m). The function learnt by such a network (often called a 'siamese network') is conceptually one that maps input vectors for the same entity closer together in vector space, while mapping input vectors for a different entity to a distance that is at least m distance away from the vectors for the same entity, as shown in Figure 1. This sort of function can be used to determine join equality on a subset of string pairs that are considered joinable using some sort of filtering step, as is often done in approaches to the semantic join type problem (e.g. [8]).

However, our observation is that one can actually exploit what the siamese network produces to eliminate this filtering step altogether. Specifically, the last hidden layer of the siamese network is effectively a 'vector embedding' for the same versus different estimate. That is, the last layer is a vector in a lower dimensional space that contains the critical features needed for computing the distance estimate. We can in fact take these vector embeddings for all vectors in the two tables to be joined, and use approximate nearest neighbors algorithms to find the nearest neighbors. Ideally, the nearest neighbor that has a distance lower the margin m should be the correct match. Because approximate nearest neighbor algorithms have been applied successfully to millions of items, this approach should scale just as well as filtering/join algorithms, but have the added advantage that the filtering step is also data driven, and hence can adapt to different entity types very effectively.

Our key contributions to the join problem are as follows:

- We demonstrate that we can use siamese networks to learn a distance function that successfully discriminates same pairs from distance pairs with an F score of .9. Our results suggest that deep learning models can be used successfully as another mechanism for data driven joins, assuming some sort of filtering approach has been applied to the data.

- We explore whether it is possible to extend this further, which is to see if the output of the function can in fact be used in an approximate nearest neighbor algorithm to identify neighbors to match with, without any filtering.

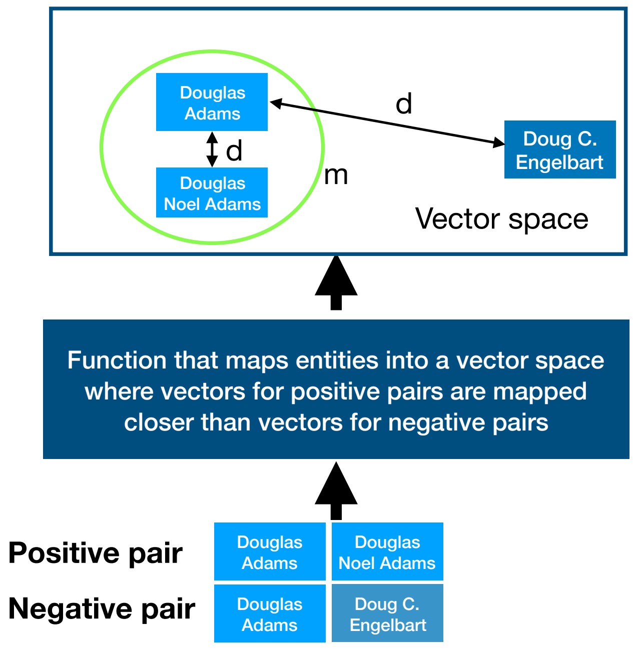


Figure 2: Conceptual overview of neural network approach to matching

2. ALGORITHM

2.1 Dimensionality Reduction by Learning an Invariant Mapping

The problem of dimensionality reduction, mapping a high dimensional group of entities onto a low dimensional manifold, is an important problem in many areas of machine learning. Dimensionality Reduction by Learning an Invariant Mapping is a technique for solving this problem by mapping points onto the manifold using a set of previously known matches and non-matches described by [3]. We used this technique not for reducing the dimensionality, but for logically mapping points onto a vector space. The goal of this technique is to minimize the distance between matched pairs and maximize the distance between unmatched pairs. This is accomplished using a network architecture know as a *siamese* network. In this architecture there are two deep neural networks that share weights attached to a final layer that determines the distance between the output of the two hidden networks and minimizes the loss function. To train the network a set of matched and non-matched pairs of inputs is sent through the network and the weights are adjusted to maximize or minimize distance accordingly. This is accomplished using the following loss function

$$L(W, Y, \vec{X}_1, \vec{X}_2) = (1-Y)\frac{1}{2}(D_W)^2 + (Y)\frac{1}{2}\{max(0, m-(D_W))\}^2$$

Where D_W is the euclidean distance between the two outputs, m is a radius beyond which we ignore dissimilar points, and Y is the similar or dissimilar label.

By mapping the inputs onto meaningful locations on a manifold, this method clusters similar inputs close to each other. Using this property we can use the distance layer to decide if two inputs are matched. We simply set a limit for euclidean distance and match if the outputs of the hidden layers are within that limit, as in Figure 4. This lets us use this method for our problem of matching, we treat items as similar to each other if they are aliases for the same entity and different otherwise. To begin with we created the non-matched pairs by choosing other entities with at least one word in common, to avoid the network learning a useless function. This successfully brought the matches close together in the input but failed to properly distance pairs that should be different, as shown in Figure 3. Since we never fed the network pairs that were completely different it did not learn to distinguish them from matches. To combat this we created two types of different pairs: ones that share at least one word in common and ones that are completely different.

2.2 Approximate Nearest Neighbors

We used the ANNOY package to find the nearest neighbors in output of the hidden layer computed in the previous section.[1] This package approximates the nearest neighbors by first calculating binary trees using the randomized k-d forest method.[5] These trees are calculated by splitting the vector space by drawing random hyperplanes. The algorithm then continues to split each subspace recursively until no more than a predetermined number of items are in each subsection. Some of the neighbors can be found by looking at the items in the same subspace, or subspaces near the target item in the tree. The processes of building a tree is repeated a number of times. By taking a union of the

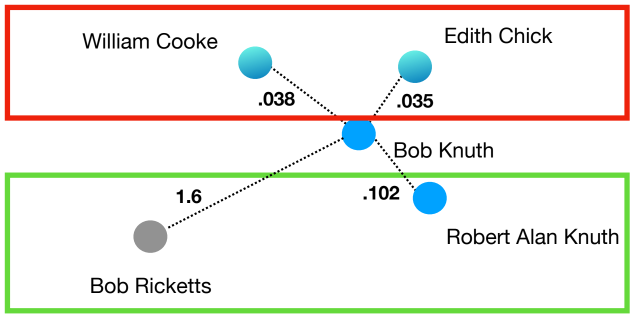


Figure 3: The model preformed well on somewhat similar names and badly on completely different names

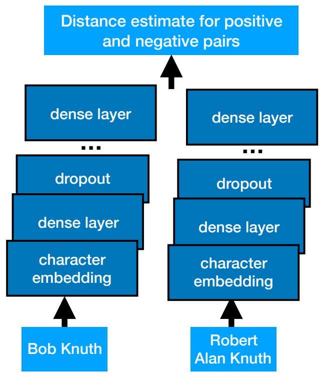


Figure 4: A siamese network has two networks which share weights which feed into a distance estimator

Table 1: Table 1

Name	Match
Douglas Adams	Douglas Noel Adams
Andreas Capellanus	Andrea Cappellano
John Adams Whipple	John A. Whipple

points from all the trees, a reasonable neighborhood is approximated. Once we have a neighborhood of points, we can calculate distance and sort the subset of points. Since much of the time goes into creating the trees, queries can be executed quickly, in logarithmic time.

3. ALGORITHM

We started our process with a simple rule based method for matching and another one for blocking. We then replaced individual pieces with a machine learning method and compared the performance. In the final product we have an elegant completely machine learning based method.

3.1 Rule-based Matching

In the rule-based method, we block based on items that have one word in common. This is the simplest reasonable blocking strategy. Table 2 illustrates what the blocking of the three names from Table 1 would look like. Out of NUMBER1 pairs, NUMBER2 ended up in the same bucket as all their correct matches. NUMBER3 had at least one correct match in the same bucket. To perform the actual matching, we used three simple rules see Table 3:

1. If a word is unique to two items, we match them. (‘John Adams Whipple’ matches ‘John A. Whipple’ since ‘Whipple’ only apears in those two names)
2. If, when all spaces are removed, one of the items is a substring of the other, we match them. (‘the flower company’ matches ‘theflowercompany.com’ since ‘theflowercompany’ is a subset of ‘theflowercompany.com’)
3. If we treat each name as a set of words, and one set is a subset of the other set, we match them. (‘Douglas Adams’ matches ‘Douglas Noel Adams’ since $\{$‘Douglas’, ‘Adams’$\} \subseteq \{$‘Douglas’, ‘Noel’, ‘Adams’$\}$)

Even using these three rules in combination resulted in too many false negatives. Therefore we decided to treat all matches resulting from any of the three rules as valid. If we treat it as a success if there is a correct match in any of the first three slots, we get an F_1 of FSCORE1. Since the issue is mostly false negatives, and therefore even some of the first three slots were often empty, so even if we accept any of the top 1000 as a success, we get FSCORE2, not much better.

3.2 Siamese Netwok

The first part we replaced was the matcher. To compare the two matchers we created a rule-based matcher using the rules from the rule-based system. To replace it we created a siamese network (figure 1) INSERT FIGURE 1. A Siamese network is two deep neural networks that share weights. One entity is fed into each of the two networks. The output of the two networks is fed into a final layer which determines the distance between the two outputs and accepts or rejects. Siamese networks have been shown to work well for image matching.[3] Before feeding the entities into the networks, we used Kazuma charecter embeddings to encode each entity as a vector. We trained the network on our NUMBER1*0.95 pairs and withheld NUMBER1*0.05 for testing. We chose an equal number of negitive pairs to train the model on. We chose negitive pairs that had at least one word in common with the positive ones. We found that the fscore on the training data was FSCORE3 and FSCORE4 on the test date. This was better than the rule-based matcher which had an f-score of FSCORE5.

We next approached the blocking problem with machine learning. Since the hidden layer of the siamese network is optimized for matching, it should output an embedding of the entity which has its essential qualities with regaurds to matching (figure 2). We then used an aproximate nearest neighbors algorithm to find the nearest neighbors. EXPLAIN MORE Using these techniques we can find the most similar items without needing to block at all. The problem we ran into is only PERCENT1 of the correct matches were in the 10 nearest neighbors. The problem seems to be that since we trained the model using negative pairs that have at least one word in common, it did not learn that names that are completly different should be mapped seperatly (figure 3). We could try to fix this by feeding the model better negative pairs, but we found a more elegant solution. Instead of trying to aproximate the correct location using a siamese network trained for matching, we can use a triplet loss function and teach it to maximize the distance to the closest false pair and minimize the distance to the true matches. To find these closest entities we use our charecter embedings. This allowed us to get an f-score of FSCORE6 on the training set and FSCORE7 on the test set reading the 10 closest entities. Notice that at this point we can do away with the matching algorithm alltogether. We simply use the closest entiies as our matches.

4. RELATED WORKS

There have been a number of attemts to solve the fuzzy join problem. Many of the attemts use varios string matching algorithms for example [7]. The main issue with using string matching comparisons is that they do not work across a wide variaty of enties ASK DR. SRINIVAS. To our knowledge no serios machine learning aproach has been used for this problem. The problem of blocking has also been dealt with using MapReduce [6]. This approach requires alot of computing power and, since they use a string matching approach in the end, is hard to generalize accross many types of entities.

5. CONCLUSIONS

6. ACKNOWLEDGMENTS

7. ADDITIONAL AUTHORS

Additional authors: John Smith (The Thørväld Group, email: `jsmith@affiliation.org`) and Julius P. Kumquat (The Kumquat Consortium, email: `jpkumquat@consortium.net`)

8. REFERENCES

[1] E. Bernhardsson. Nearest neighbors and vector models part 2 algorithms and data structures, Oct 2015.

Table 2: Table 2

Bucket Name	Contents
A	{John A. Whipple}
Adams	{Douglas Adams, Douglas Noel Adams, John Adams Whipple}
Andreas	{Andreas Capellanus}
Andrea	{Andrea Cappellano}
Cappellano	{Andrea Cappellano}
Capellanus	{Andreas Capellanus}
Douglas	{Douglas Adams, Douglas Noel Adams}
John	{John Adams Whipple, John A. Whipple}
Noel	{Douglas Noel Adams}
Whipple	{John Adams Whipple, John A. Whipple}

[2] W. W. Cohen, P. Ravikumar, and S. E. Fienberg. A comparison of string distance metrics for name-matching tasks. In *Proceedings of IJCAI-03 Workshop on Information Integration*, pages 73–78, August 2003.

[3] R. Hadsell, S. Chopra, and Y. Lecun. Dimensionality reduction by learning an invariant mapping. *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 2 (CVPR06)*.

[4] Y. He, K. Ganjam, and X. Chu. Sema-join: Joining semantically-related tables using big table corpora. *Proc. VLDB Endow.*, 8(12):1358–1369, Aug. 2015.

[5] M. Muja and D. G. Lowe. Scalable nearest neighbor algorithms for high dimensional data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(11):22272240, Jan 2014.

[6] R. Vernica, M. J. Carey, and C. Li. Efficient parallel set-similarity joins using mapreduce. *Proceedings of the 2010 international conference on Management of data - SIGMOD 10*, 2010.

[7] J. Wang, G. Li, and J. Fe. Fast-join: An efficient method for fuzzy token matching based string similarity join. *2011 IEEE 27th International Conference on Data Engineering*, 2011.

[8] E. Zhu, Y. He, and S. Chaudhuri. Auto-join: Joining tables by leveraging transformations. International Conference on Very Large Databases (VLDB), May 2017.

8.1 References

Generated by bibtex from your .bib file. Run latex, then bibtex, then latex twice (to resolve references) to create the .bbl file. Insert that .bbl file into the .tex source file and comment out the command `\thebibliography`.

APPENDIX