Learning Distributed Representations of Relational Data using Linear Relational Embedding

Alberto Paccanaro
Geoffrey E. Hinton
Gatsby Computational Neuroscience Unit
University College London, London, UK
{alberto,hinton}@gatsby.ucl.ac.uk

Abstract

Linear Relational Embedding (LRE) is a new method of learning a distributed representation of concepts from data consisting of binary relations between concepts. The final goal of LRE is to be able to generalize, i.e. to infer new relations among the concepts. The version presented here is capable of handling incomplete information and multiple correct answers. We present results on two simple domains, that show an excellent generalization performance.

1 Introduction

Let us consider a world in which we have objects related to each other through some relations. Let us look for now only at *binary* relations, i.e. relations involving two objects. The way in which these objects relate by means of these relations can be written out as triplets, of the kind {*object1*, *Relation*, *object2*}. We are interested in the following problem: *given a few of these triplets, is it possible to infer the other ones?*

Clearly this problem is hopeless if the relations associate objects in a random fashion. Fortunately this is unlikely to happen if the data is taken from the real world where, in general, a lot of structure is present since real objects relate to each other depending on "what" they are, i.e. according to their features. Think for example, of the relations that relate the members of a family. The relation *"has mother"* relates any person to another which must have the features of being *older* than that person (by exactly one generation) and *female.*

This gives us a starting point to try to solve our problem: if objects relate to each other according to their features, then if we could obtain the features of each object which are relevant for the set of relations at hand and the correct rules for how these features interact, it should possible to infer how objects relate. Our approach to solving this problem is therefore to assume structure in the world, to learn a set of features for each object and the rules on how they interact, and then to use these features to infer new triplets. We want to find a mapping from the objects into a feature-space and our idea is to do this by imposing the constraint that relations in this feature-space can be modeled by linear operations. We represent each object as a learned vector in a Euclidean space and each relationship between objects as a learned matrix that maps an object into an approximation of another object. We call this technique *Linear Relational Embedding* (LRE).

Several methods already exist for learning sensible distributed representations from relational data. Multidimensional Scaling [2, 9] finds a representation of concepts as vectors in a multi-dimensional space, in such a way that the dissimilarity of two concepts is modeled by the Euclidean distance between their vectors. Unfortunately, dissimilarity is the only relationship used by multidimensional scaling so it cannot make use of the far more specific information about concepts contained in a triplet like (`Mary`, `has_father`, `John`).

Latent Semantic Analysis (LSA) [3, 4] assumes that the meaning of a word is reflected in the way in which it co-occurs with other words. LSA finds features by performing singular value decomposition on a large matrix and taking the eigenvectors with the largest eigenvalues. Each row of the matrix corresponds to a paragraph of text and the entry in each column is the number of times a particular word occurs in the paragraph or a suitably transformed representation of this count. Each word can then be represented by its projection onto each of the learned features and words with similar meanings will have similar projections. Again, LSA is unable to make use of the specific relational information in a triplet.

Hinton (1986) showed that a multilayer neural network trained using backpropagation could make explicit the semantic features of concepts and relations present in the data. This was done on a simple task called the *family tree problem*. In this problem, the data consists of people and relations among people belonging to two families, one Italian and one English, shown in figure 1 [1]. All the information in these trees can be represented in simple propositions of the form *(person1, relation, person2)*. Using the relations *father, mother, husband, wife, son, daughter, uncle, aunt, brother, sister, nephew, niece* there are 112 such triplets in the two trees. Unfortunately, the system had problems in generalizing when many triplets were missing from the training set.

Finally, it should be mentioned that there has been also a different, more logic-based, approach to the problem of learning relational data. FOIL [8] assumes that relational information can be represented as a set of predicates, i.e. mappings from k-tuples of objects onto truth values. Given data consisting of these mappings for a particular set of concepts (and under the closed-world assumption) FOIL learns a definition of each predicate in terms of the other ones and itself. This is particularly interesting when the data contains a set of basic predicates: FOIL is then able to learn other predicates that can be expressed as a combination of the basic ones. The definitions which are learned are very similar to Horn clauses and will then hold for any other set of data. A limitation of FOIL is that it can learn the definition of a predicate only when

[1]The names of the Italian family have been altered from those originally used in Hinton (1986) to match those of one of the author's family.

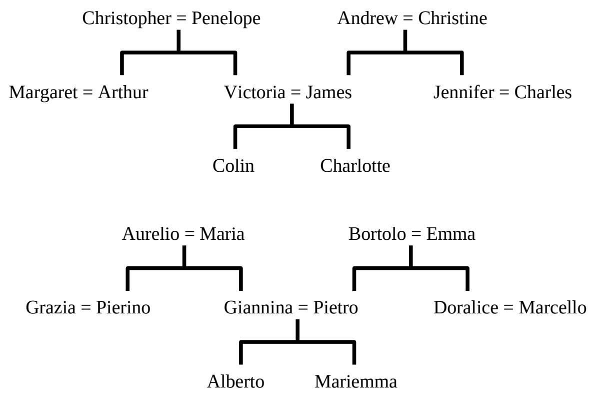


Figure 1: Two isomorphic family trees. The symbol "=" means "married to".

it is possible to define it in terms of the predicates available.

The next section presents the details of Linear Relational Embedding, showing how it overcomes the limitations of the systems which we have presented here.

2 Linear Relational Embedding

Let us assume that our data consists of C triplets (*concept1, relation, concept2*) containing N distinct objects and M binary relations. We shall call this set of triplets $\mathcal{C}$. As anticipated in section 1, the main idea of Linear Relational Embedding is to represent each concept with an n-dimensional vector, and each relation with an $(n \times n)$ matrix. We shall call: $\mathcal{V} = \{\mathbf{v}_\infty, ..., \mathbf{v}_\mathcal{N}\}$ the set of vectors corresponding to the N objects, and $\mathcal{R} = \{\mathcal{R}_\infty, ..., \mathcal{R}_\mathcal{M}\}$ the set of matrices corresponding to the M relations. Often we shall need to indicate the vectors and the matrix which correspond to the concepts and the relation in a certain triplet c. In this case we shall denote the vector corresponding to the first concept with $\mathbf{a}$, the vector corresponding to the second concept with $\mathbf{b}$ and the matrix corresponding to the relation with R. We shall therefore write the triplet c as $(\mathbf{a}^c, R^c, \mathbf{b}^c)$ where $\mathbf{a}^c, \mathbf{b}^c \in \mathcal{V}$ and $R^c \in \mathcal{R}$. The operation that relates a pair $(\mathbf{a}^c, R^c)$ to a vector $\mathbf{b}^c$ is the matrix-vector multiplication, $R^c \cdot \mathbf{a}^c$, which produces an approximation to $\mathbf{b}^c$.

Here we discuss how to learn an embedding given a set of data, that is how to find suitable vectors and matrices such that for each triplet $(\mathbf{a}^c, R^c, \mathbf{b}^c)$, $\mathbf{b}^c$ is as close as possible to $R^c \cdot \mathbf{a}^c$. The obvious approach is to minimize the sum of squared distances between $R^c \cdot \mathbf{a}^c$ and $\mathbf{b}^c$ over all the triplets

$$S = \sum_{c=1}^{C} ||R^c \cdot \mathbf{a}^c - \mathbf{b}^c||^2 \qquad (1)$$

with respect to all the vector and matrix components. Unfortunately minimizing S (almost) always causes all of the vectors and matrices to collapse to

0. In order to be able to learn a linear embedding, we need a way to minimize S, while at the same time keeping away from the trivial **0** solution. To achieve this we used a discriminative function: for each triplet, c in addition to minimizing the squared distance between $R^c \cdot \mathbf{a}^c$ and $\mathbf{b}^c$ we also maximize the squared distances to other concept vectors that are nearby. This keeps the vectors different from each other, and thus away from the **0** solution. One way of writing such a discriminative function is to think of $R^c \cdot \mathbf{a}^c$ as a noisy version of one of the concept vectors, and to maximize the probability that it is a noisy version of the correct answer, $\mathbf{b}^c$, rather than any of the other possibilities. We imagine that a concept has an average location in the space but that each "observation" of the concept is a noisy realization of this average location. Assuming spherical Gaussian noise with a variance of 1/2 on each dimension, the probability that a realization of concept i would occur at $R^c \cdot \mathbf{a}^c$ is proportional to $\exp(-\|R^c \cdot \mathbf{a}^c - \mathbf{v}_i\|^2)$. So the posterior probability that $R^c \cdot \mathbf{a}^c$ matches concept $\mathbf{b}^c$ given that it must match one of the known concepts is:

$$P(\mathbf{b}^c | R^c \cdot \mathbf{a}^c) = \frac{e^{-\|R^c \cdot \mathbf{a}^c - \mathbf{b}^c\|^2}}{\sum\limits_{\mathbf{v}_i \in \mathcal{V}} e^{-\|R^c \cdot \mathbf{a}^c - \mathbf{v}_i\|^2}} \tag{2}$$

A discriminative goodness function D, that corresponds to the log probability of getting the right answer, summed over all training triplets is therefore:

$$D = \sum_{c=1}^{C} \frac{1}{k_c} \log \frac{e^{-||R^c \cdot \mathbf{a}^c - \mathbf{b}^c||^2}}{\sum\limits_{\mathbf{v}_i \in \mathcal{V}} e^{-||R^c \cdot \mathbf{a}^c - \mathbf{v}_i||^2}} \tag{3}$$

where k_c is the number of triplets in $\mathcal{C}$ having the first two terms equal to the ones of c, but differing in the third term. Learning based on maximizing D has given good results, and has proved successful in generalization as well. Examples of this learning can be found in Paccanaro and Hinton (2000a, 2000b, 2001). However, when we learn an embedding by maximizing D, we are not making use of exactly the information that we have in the triplets. For each triplet c, we are making the vector representing the correct completion $\mathbf{b^c}$ more likely than any other concept vector given $R^c \cdot \mathbf{a}^c$ assuming Gaussian noise, while the triplet states that $R^c \cdot \mathbf{a}^c$ must be *equal* to $\mathbf{b^c}$. The numerator of D does exactly that, but then there is the denominator, that is necessary in order to stay away from the **0** solution. We noticed that the denominator is critical at the beginning of the learning, when the vector and matrices could easily shrink to **0**. But as they differentiate, we could gradually lift this burden, allowing $R^c \cdot \mathbf{a}^c = \mathbf{b^c}$ to become the real goal of the learning. To do this we propose here a modified discriminative function that includes a parameter α, which is annealed from 1 to 0 during learning:

$$G = \sum_{c=1}^{C} \frac{1}{k_c} \log \frac{e^{-\|R^c \cdot \mathbf{a}^c - \mathbf{b}^c\|^2}}{[\sum\limits_{\mathbf{v}_i \in \mathcal{V}} e^{-\|R^c \cdot \mathbf{a}^c - \mathbf{v}_i\|^2}]^\alpha} \tag{4}$$

This function G (for Goodness) is maximized during learning. At the beginning, when $\alpha = 1$, maximizing G is equivalent to maximizing D. Then annealing α amounts to gradually changing the error function, until we end up minimizing S. This gives a much better generalization performance than the one obtained by just maximizing D [2].

The results presented in the next sections were obtained by maximizing G using gradient ascent. All the vector and matrix components were updated simultaneously at each iteration. One effective method of performing the optimization is conjugate gradient. Learning was fast, usually requiring only a few hundred updates. In general, different initial configurations and optimization algorithms caused the system to arrive at different solutions, but these solutions were almost always equivalent in terms of generalization performance.

3 Results

Here we shall present the results we obtained using LRE on the *family tree problem* [1] presented earlier, as well as on a simpler task, which we called the *number problem*. In the number problem, the objects are the integers in the set $\mathfrak{D} = [\mathtt{0} \ldots \mathtt{m} - \mathtt{1}]$ and the relations are the operations
$\mathfrak{L} = \{\mathtt{+1}, \mathtt{-1}, \mathtt{+2}, \mathtt{-2}, \mathtt{+3}, \mathtt{-3}, \mathtt{+4}, \mathtt{-4}, \mathtt{+0}\}$. The data then consists of all or some of the triplets (num_1, op, num_2) where $num_1, num_2 \in \mathfrak{D}$, $op \in \mathfrak{L}$, and num_2 is the result of applying operation op to number num_1 — when the result of the operation is outside $[\mathtt{0} \ldots \mathtt{m} - \mathtt{1}]$ we omit the corresponding triplet from the data set.

Fig. 2 show the layout of the vectors obtained for a run of the algorithm for the two problems. In the solution for the family tree problem, notice how the Italians are linearly separable from the English people. From the Hinton diagram, we can see that each member of a family is symmetric to the corresponding member in the other family. The third component of the vectors is almost a feature for the nationality.

The interesting question is whether the system can correctly complete triplets on which it has not been trained. The obvious method for completing triplet $(\mathbf{a}, R, ?)$ is to choose as completion the concept which is closest to $R \cdot \mathbf{a}$. After a concept, say $\mathbf{k}$, has been chosen, we can check whether the triplet $(\mathbf{a}, R, \mathbf{k})$ belongs to the data set. If we use this method for evaluating the system, results are very good. For example, the solution to the number problem shown in figure 2 was trained using only 1/3 of the total number of triplets (143) and still the system was able to complete correctly all of them (430). For the family tree problem the system was generally able to complete correctly all 112 triplets even if 28 of them had been left out during training. These results on the Family Tree problem are much better than the ones obtained by Quinlan (1990) and Hinton (1986) on the same problem.

[2] For one-to-many relations we must not decrease the value of α all the way to 0, because this would cause some concept vectors to become coincident. This is because the only way to make $R^c \cdot \mathbf{a}^c$ equal to k_c different vectors, is by collapsing them onto a unique vector.

This method for completing the triplets works well if all the relations are one-to-one. It clearly cannot deal properly with one-to-many relations, in which we may have multiple correct answers. In the Family Tree problem, `Colin` has 2 aunts, namely `Jennifer` and `Margaret`. But if we complete the triplets as described above, the system will always output only one of them, the one which is closest to the dot product of matrix `''aunt''` and vector `''Colin''`. Although the answer given is correct, we would like to obtain *all* the correct answers.

Moreover, here we argue that for this kind of problem it is not sufficient to test generalization by merely testing the completion of those valid triplets which have not been used for training. The proper test for generalization is to see how the system completes any triplet of the kind $(\mathbf{a}, R, ?)$ where $\mathbf{a}$ ranges over the concepts and R over the relations. This is because in real world applications we cannot expect to know in advance that triplets like (`Christopher, father, ?`) cannot be completed, because our tree is incomplete (and `Christopher` belongs to the first generation). More generally, we should not suppose to have knowledge of which completions can be asked, and which ones cannot. To our knowledge this issue has never been analyzed before (even though FOIL should be able to correctly handle this problem).

The system needs a way to indicate for those triplets that do not admit a completion that the correct answer is "don't know". To accomplish this, although at the end of the learning we were optimizing S, once the learning has finished we switch back to a probabilistic model. This model is constituted, for each relation, of a mixture of N identical spherical Gaussians, each centered on a vector, and a Uniform distribution. The Uniform distribution will take care of the "don't know" answers, and will be competing with all the other Gaussians, each representing a vector constituting an answer in our dataset. For each relation the Gaussians have different variances and the Uniform a different height. The parameters of this probabilistic model are, for each relation R, the variances of the Gaussians σ_R and the relative density under the Uniform distribution, which we shall write as $\exp(-\frac{r_R^2}{2\sigma_R^2})$. These parameters are learned using a validation set, which will be the union of a set of completable triplets $\mathcal{P}$ and a set of pairs which cannot be completed $\mathcal{N}$; that is $\mathcal{P} = \{\mathbf{a}, \mathcal{R}, \mathbf{b}\}_{=}^{\mathcal{P}}$ and $\mathcal{N} = \{\mathbf{a}, \mathcal{R}, \perp\}_{=}^{\mathcal{Q}}$ where $\perp$ indicates the fact that the result of applying relation R^q to $\mathbf{a}^q$ does not belong to $\mathcal{V}$. This is done by maximizing the following discriminative goodness function F over the validation set :

$$\begin{aligned} F &= \sum_{q=1}^{Q} \log \frac{\exp(-\frac{r_R^2}{2\sigma_R^2})}{\exp(-\frac{r_R^2}{2\sigma_R^2}) + \sum_{\mathbf{v}_i \in \mathcal{V}} \exp(-\frac{\|R^q \cdot \mathbf{a}^q - \mathbf{v}_i\|^2}{2\sigma_R^2})} \\ &+ \sum_{p=1}^{P} \frac{1}{k_p} \cdot \log \frac{\exp(-\frac{\|R^p \cdot \mathbf{a}^p - \mathbf{b}^p|^2}{2\sigma_R^2})}{\exp(-\frac{r_R^2}{2\sigma_R^2}) + \sum_{\mathbf{v}_i \in \mathcal{V}} \exp(-\frac{\|R^p \cdot \mathbf{a}^p - \mathbf{v}_i\|^2}{2\sigma_R^2})} \end{aligned} \tag{5}$$

with respect to the σ_R and r_R parameters, while everything else is kept fixed. As before, k_p is the number of triplets in $\mathcal{P}$ having the first two elements equal to the ones of p, but differing in the third one. Thus for each triplet in $\mathcal{P}$, we maximize the probability that the vector $R^p \cdot \mathbf{a}^p$ is generated from the Gaussian centered on $\mathbf{b}^p$ while minimizing the probability that it comes from any other Gaussian or the Uniform distribution. At the same time, for each triplet in $\mathcal{N}$, we maximize the probability that the vector $R^q \cdot \mathbf{a}^q$ is generated from the Uniform distribution while minimizing the probability that it comes from any of the Gaussians.

Having learned these parameters, in order to complete any triplet $(R, \mathbf{a}, ?)$ we compute the probability distribution over each of the Gaussians and the Uniform distribution given $R \cdot \mathbf{a}$. The system then chooses a vector $\mathbf{v}_i$ or the "don't know" answer according to those probabilities, as the completion to the triplet.

When we tried this method on the number problems, the system always assigned a probability of 1 to the correct answers and 0 to the wrong answers for each of the completable triplets. It also assigned a probability of 1 to the "don't know" symbol for every triplet for which a completion did not exist. In other words, it answered correctly to all 450 possible completions.

For the family tree problem we first picked 12 triplets for the test set, out of the 112 total set of triplets. These were chosen at random, but such that the test set had a triplet for each relation. The same operation was done to obtain another 12 completable triplets for the validation set. Then another 12 uncompletable triplets, 1 per relation, were added to the validation set. So the training set was constituted by 88 triplets, the validation set by 24 triplets and the test set by 12. Once the embedding was learned using the training set, we learned the values of σ_R and r_R for each relation by maximizing F, using the validation set, as described above. We then tested the system by attempting to complete all the 288 possible triplets $(R, \mathbf{a}, ?)$. Figure 3 shows the distribution of the probabilities assigned to each of the 24 concept vectors representing a person, together with the probability assigned to the "don't know" answer when completing the 12 completable triplets in the test set. We can see that the system gives the correct answer, but sometimes it also communicates that "it is not very sure about it" by assigning some probability to the "don't know" answer as well. When tested on uncompletable triplets the system almost always assigned a probability of 1 to the "don't know" symbol, and only very seldom it assigned a small probability to one of the people in the dataset.

4 Conclusions

Linear Relational Embedding is a new method for discovering distributed representations of concepts and relations from data consisting of binary relations between concepts. On the task on which we tried it, it was able to learn sensible representations of the data, and this allowed it to generalize better than any other published method on the datasets used. The main open question is how well it will scale up to problems of bigger size, even if preliminary results on larger data sets seem to be encouraging. LRE can handle any kind of binary relations, learning is fast, generalization is very good. A major limitation at present is that relations need to be binary. We are working on a version that uses higher order tensors to be able to handle higher arity relations.

References

[1] Geoffrey E. Hinton. Learning distributed representations of concepts. In *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*, pages 1–12. Erlbaum, NJ, 1986.

[2] J. B. Kruskal. Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika*, 29, 1:1–27, 1964.

[3] Thomas K. Landauer and Susan T. Dumais. A solution to Plato's problem: The latent semantic analysis theory of acquisition, induction and representation of knowledge. *Psychological Review*, 104, 2:211–240, 1997.

[4] Thomas K. Landauer, Darrel Laham, and Peter Foltz. Learning human-like knowledge by singular value decomposition: A progress report. In Michael I. Jordan, Michael J. Kearns, and sara A. Solla, editors, *Advances in Neural Processing Information Systems 10*, pages 45–51. The MIT Press, Cambridge Massachusetts, 1998.

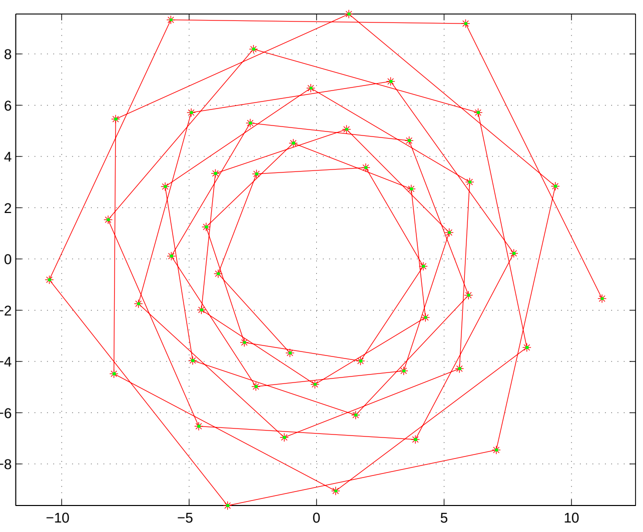
[5] Alberto Paccanaro and Geoffrey E. Hinton. Extracting distributed representations of concepts and relations from positive and negative propositions. In *Proceedings of the International Joint Conference on Neural Networks, IJCNN 2000.* 2000.

[6] Alberto Paccanaro and Geoffrey E. Hinton. Learning distributed representations by mapping concepts and relations into a linear space. In Pat Langley, editor, *Proceedings of the Seventeenth International Conference on Machine Learning, ICML2000*, pages 711–718. Morgan Kaufmann Publishers, Stanford University, San Francisco, 2000.

[7] Alberto Paccanaro and Geoffrey E. Hinton. Learning distributed representation of concepts using linear relational embedding. *to appear in IEEE Trans. on Knowledge and Data Engineering - special issue on 'Connectionists Models for Learning in Structured Domains*, 2001.

[8] J. R. Quinlan. Learning logical definitions from relations. *Machine Learning*, 5:239–266, 1990.

[9] F. W. Young and R. M. Hamer. *Multidimensional Scaling: History, Theory and Applications.* Hillsdale, NJ: Lawrence Erlbaum Associates, Publishers,, 1987.



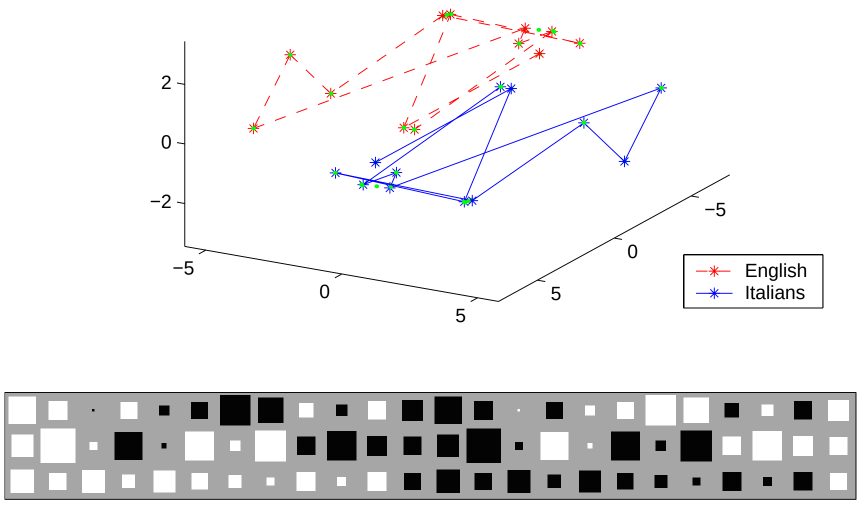


Figure 2: (a)Vectors obtained after learning the number problem with numbers $\in [1 \ldots 50]$ in two dimensions. Vector endpoints are marked with stars and a solid line connects the ones representing consecutive numbers. Vectors corresponding to smaller numbers are closer to the center of the spiral. The training set was constituted by 143 triplets randomly chosen from the total 430 correct triplets. (b)Top: layout of the vectors in 3D space obtained for the family tree problem. Vectors endpoints are indicated by *, the ones in the same family tree are connected to each other. All 112 triplets were used for training. Bottom: Hinton diagrams of the vectors in 3D space obtained for the family tree problem. The vector of each person is a column, in the following order from left to right: 1=Christopher; 2=Andrew; 3=Arthur; 4=James; 5=Charles; 6=Colin; 7=Penelope; 8=Christine; 9=Margaret; 10=Victoria; 11=Jennifer; 12=Charlotte; 13=Aurelio; 14=Bortolo; 15=Pierino; 16=Pietro; 17=Marcello; 18=Alberto; 19=Maria; 20=Emma; 21=Grazia; 22=Giannina; 23=Doralice; 24=Mariemma.

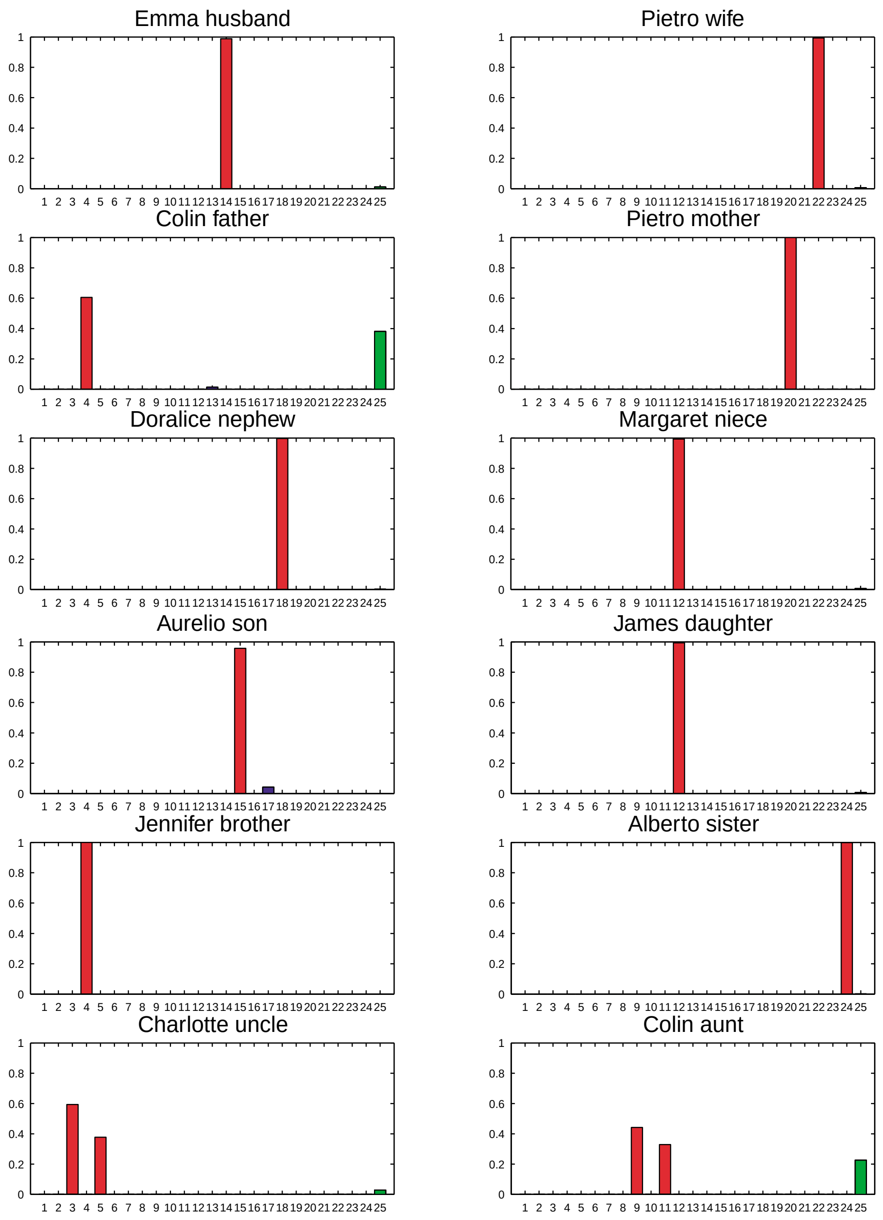


Figure 3: Distribution of the probabilities assigned to each concept for each of the triplet in the test set, written above each diagram. Black bars from 1 to 24 are the probabilities of the people in the order specified in fig. 2. The last grey bar on the right, is the probability of the "don't know" answer.