Robo Brain: Massive Knowledge Base For Robots

Gabriel Kho, Christina Hung, Hugh Cunningham
{*gdykho, chung888, hughec*}@stanford.edu

December 12, 2014

Abstract

The RoboBrain project is a collaborative effort to build a knowledge engine that learns and shares knowledge representations. These knowledge representations are learned from a variety of sources, including interactions that robots have while performing perception, planning and control, as well as natural language, semi-structured knowledge and visual data from the Internet. This representation needs to consider several modalities including symbols, natural language, visual or shape features, haptic properties, and so on. The knowledge that Robo-Brain accumulates is stored in a graph database. The structure lends well to a data visualization. Through the data visualization, we crowd-source comments on the graph to find problems with the data. This paper details an attempt to create a learning algorithm based off of those comments so that we can predict fault nodes in the graph and alert them to administrators.

1 Introduction

The Robo-Brain project is knowledge engine that will share and learn knowledge representations. The knowledge is stored in a graph structure and created by crawling knowledge repositories on the internet like OpenCyc[1] and WordNet[2][4][6]. However, as with any automatic crawler, errors in the graph can pop-up. First, any errors in OpenCyc and WordNet will propagate to Robo-Brain's knowledge engine. Another important problem is that as data is being coaslesced from multiple sources, incorrect representations will appear. For example, a node "plant" might have connections to the nodes "roots," "leaves," and "nuclear energy." What this entails is that the node "plant" needs to be disambiguated into two different nodes, one for the biological and plant and another for a power plant.

In order to quickly see these problems, a data visualization was created since the structure of a graph database lends to well to directed force layout. Unlike row-store databases, a graph database in based on graph theory. Each item in the database contains a pointer to its adjacent item. Thus, we can think of a tuple as a node, edge and node. Each node and edge can have their own set of properties. In Robo-Brain, each node symbolizes a concept, such as the noun "car" or a verb "slip". The name of these nodes is stored in the attribute handle of these nodes. Other relevant attributes include the source urls that this node was generated from, upvotes and downvotes on the source, and the lines to other edges. The only attributes we really care about on our edges is the pointer to the source node and the pointer to the target node.

The solution we propose is to crowd-source feedback on the graph. Humans can intuitively tell if there is a problem with a subsection of the graph. Previous work has been done in trying courdsource the information humans intuitively know and try to incorporate it into robots[3]. Thus, the data visualization built in D3[7] was created with options for users to comment on the nodes and edges of the graph. Afterwards, the algorithm is planned to be put into a system to alert administrators to potentially problematic nodes. FOr this paper, we did all our implementation in Scikit, a Python library [5].

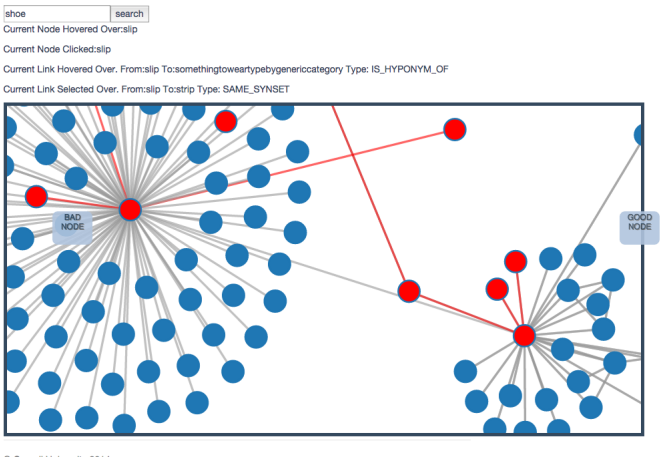

Figure 1. Visualizing the structure of the graph database

2 Dataset

Our dataset contains 707 comments over 225 nodes and 207 over 122 of edges. Most of these are collected by walking through the graph starting at the "shoe" node. A lot of the data has been collected by getting feedback from people involved in the project. Unfortunately, complications outside the classroom led to delays of the data visualization being accessible to people around the world, thus resulting in a much smaller dataset than we anticipated.

The feedback itself from the the graph was stored in a mongoDB. The only we differentiated a node from an edge was though the *node_id*. Otherwise, they had the same structure.

A challenging part of this dataset is we have conflicting data. Some feedback might list a node as good while others might list it as problematic. In addition, we have no idea what the underlying distribution of the feedback is.

For a node, the feedback structure was such that there was an *id_node*, which was a unique id that identified the node. The $feedback\_type$ was just a coarse level on whether the user thought the node was a good node or bad node. In this case, it either had the value of agree or disagree. Then we had a *node_handle* which was the name of the node. Due to the fact that nodes hadn't been disambiguated yet, this handle identifier was unique at the time of the writing of this paper. However, to keep this code easily integratable for the future version of the graph, we store id also which will remain unique. The last field in the feedback is a $feedback\_type$ which is the type of feedback. For the nodes, this feedback is either "GOOD NODE," "SPLIT NODE," "REMOVE NODE" or "RENAME NODE." The idea behind this was that when a node is good, everything is good, but when a node is bad, different things could be bad.

For an edge, the *node_id* takes the form of "link(sourceNodeHandle)to(targetNodeHandle)". The $feedback\_type$ is the same. The *node_handle* contains the handle of the link, which takes values like $ISAHOMONYMOF$ or $ISSAMESYSNET$. These values describe the relationship between the source and target node and are not unique. The last field $feedback\_type$ is really either $GOODLINK$ or $BADLINK$, because when a link is bad, it should just be removed.

3 Features and Preprocessing

We use different features for node prediction and link prediction. The features used in node prediction are the length of the handle name, number of non-alphabetical characters, belief, number of links, sources used for creation. We use length of handle name because some nodes come out with a name like "artifictwornbyhumans," our logic is that some nodes had their handles name incorrectly generated and a length of the handle might be a good indicator if that happened. We then use number of non-alphabetical characters because we have nodes like $HEATMAP_12$ and *shoe.jpg*, so we think that the number of non-alphabetical characters would a good indicator of that.

We also want to try to predict if two nodes have been merged such as the "plant" and "power plant" example mentioned above. We think the best indicator of this one is the number of edges a node has. A high number of edges is probably a good indicator that two distinct nodes have been merged.

The source that a node is created from is used as a feature. We use an indicator variable to indicate if a node used a particular source during its creation. This is possible because the number of sources used in Robo-Brain right now is very limited. The rationale behind this is that some sources are more trustworthy than others. In fact, down the line, we might be able to use this to rank sources.

Now, belief is a preprocessed feature that is generated by the pipeline at the creation of the node. It represents the probability that a node is good. A lot of sources contain upvotes and downvotes on the concepts that these nodes are created from. Thus, a belief is created by taking a feature vector containing upvotes, downvotes, weight of a project, and the log normalized number of times the feed (the source where the concept was generated from) appeared, and then calculating the dot product with a weight vector. Upvotes and downvotes are smoothed using Laplace smoothing, and then the result of the dot product is thrown into a sigmoid function and this results in the probability. The reason we use this preprocessed feature instead of doing more sophisticated learning on the raw data of the upvotes and downvotes is a practicality reason. Due to the way that data is stored, using the beliefs reduces our calculation time from $O(n^2)$ to $O(n)$, thus even though it might not be the most sophisticated way of doing the calculation, we use it.

In classifying whether a link is faulty or not, we build features outwards from a link through the nodes that it connects. In particular, we use the product of the belief in each of the two nodes as well as the number of links each connected node has. The intuition for the first feature is that if a link connects two nodes that are believed to be faulty, then the link itself is likely to be faulty as well. Additionally, as the number of links on a node increases, it may become more likley that links are contain faulty knowledge. This latter intuition is roughly equivalent to the inclusion of the number of links as a feature for classifying whether a node is faulty, where a high number of links might indicate that a node should be SPLIT. These few features represent an early baseline for classification of faulty links, and more analysis will be needed in the future to develop additionaly features in this space.

4 Models

We model the problem of predicting faulty nodes and links in the knowledge graph as a classification task and consider both binary and multiclass classification. In the binary case, our prediction models distinguish between "GOOD" nodes and links and "BAD" nodes and links. For the multiclass formulation of the problem the "BAD" node class is divided into clases for "SPLIT," "REMOVE," and "RENAME." There is no multiclass formulation for predicting faulty links since the only feasible action when altering a faulty link is to remove the link.

We use Support Vector Machines to perform classification in both the binary and multiclass formulations of the problem, and compare various parameter combinations for SVM classifiers with a baseline Multinomial Naive Bayes classifier. The SVM classifier is well-suited to our task since the low dimensionality of our feature set avoids the training time drawbacks of SVMs and allows us to easily capture any non-linear relationships between our features. Grid search over the SVM parameters kernel function, slack cost C, and γ (used in RBF and other kernel functions) finds that the linear kernel $K = \langle x, x' \rangle$ and a slack cost parameter $C = 0.1$ minimized classification error for predicting faulty nodes in the dataset. Ablation analysis reveals that features derived from node handles contribute most to classifier performance.

5 Results and Discussion

Given the limited size of our dataset, we use leave one out cross validation (LOOCV) to evaluate the performance of our models. We would ideally hold out a set of graph annotations on a separate part of the knowledge graph to assess the generalizability of our models, but, as mentioned previously, complications in the deployment of the wide release of the crowdsourcing interface has hindered our efforts towards data collection. In addition, the graph is ever changing its nodes and edges, so we figured it would be best to conduct our experiments on an isolated section of the graph. We report the average LOOCV misclassification error of our classification models for predicting faulty nodes in the table below:

Model	Binary	Multiclass
Multinomial Naive Bayes	0.320	0.369
SVM, Linear Kernel, $C = 1$	0.262	0.223
SVM, Linear Kernel, $C = 0.1$	0.204	0.233

Table 1. Misclassification error for predicting faulty nodes

The SVM classifiers achieve low misclassification error on the nodes in the dataset, but perhaps the more significant metric of success is the precision of the classifiers with respect to "BAD" nodes and links. For nodes, the SVM using a linear kernel and a slack cost parameter $C = 0.1$ achieves 0.888 precision with respect to the "BAD" class. A high precision indicates that the proportion of predicted faulty nodes and links that are truly faulty is high. Thus, using the predictions of a highly precise classifier to alter the graph structure would result in alterations to mostly faulty nodes and links rather than already correct ones.

The following table shows somewhat preliminary results for classification error with regard to predictiing faulty links:

Model	Classification Error
Multinomial Naive Bayes	0.263
SVM, Linear Kernel, $C = 1$	0.295

Table 2. Misclassification error for predicting faulty links

For predicting faulty edges, the baseline naive bayes classifier outperforms the baseline SVM model in achieving a lower misclassification error. With a precision of 0.737 with respect to BAD links, the precision of the naive bayes classifier seems promising, but short of where we would like our model to be before utlizing predictions for augmenting the knowledge base.

6 Conclusions

The Robo-Brain crowdsourcing project was ultimately a challenging project. Real world complications with the Robo-Brain project made data collection challenging. In addition, a lot of verification in this learning algorithm would have benefited from human evaluation, which just wasn't possible given the man power we had.

Ultimately though, our algorithm does its job of having a low number of false positives. By positives, we mean classifying a node as faulty. The reason this is important is because most of an administrator's time should not be spent evaluating nodes that aren't problematic. In that sense, this algorithm is a success, and just needs more testing.

7 Future

Robo-Brain is currently being modified as we speak. In fact, on the subgraph we examined, a new node with high number of edges appeared last week. In addition, more studies need to be done on the HCI level. For example, there is an imbalance of data related to the positive and negative examples. We hypothesize that people just might not comment if a node or edge is good, and only comment on the bad ones to save their time. However, to be sure of this, studies with a statistically revelant number of users has to be conducted.

In addition, some of the handles on the edges might not be human-readable to someone unfamiliar with the project. Thus, there might be discrepancies. This means that some of these might need to be changed.

In addition, some feedback might be because some users just don't understand some of the words presented to them. For example, "gaucherie" is connected to "rusticity." The comments on such nodes might be random noise.

Further work will be done to try and incorporate this learning algorithm into the Robo-Brain pipeline so that it can alert administrators to potentially problematic nodes. As the knowledge base grows it will become increasingly important to maintain a healthy knowledge graph, and developing additional features for capturing relationships between potentially fault nodes and edges will similarly become important for improving the performance of these models over an increasingly disparate dataset.

8 References

[1] A.Jain et al., PlanIt: "A Crowdsourcing Approach for Learning to Plan Paths from Large Scale Preference Feedback." *arXiv*:1406.2616, 2014.

[2] A.Jain et al., RoboBrain: "Large-Scale Knowledge Engine for Robots." *arXiv*:1412.0691, 2014.

[3] George A. Miller. WordNet: "A Lexical Database for English." *Communications of the ACM* Vol. 38, No. 11: 39-41, 1995.

[4] Pederosa et al., "Scikit-learn: Machine Learning in Python," *JMLR* 12, pp.2825-2830, 2011.

[5] Sw.opencyc.org, 'OpenCyc for the Semantic Web,' 2014. [Online]. Available: http://sw.opencyc.org/. [Accessed: 13- Dec- 2014].

[6] Y. Zhu et al., "Reasoning About Object Affordances in a Knowledge Base Representation." *Stanford University*, 2014.