Technical Report

Harry Baker, Madeleine Hardt, Marija Ivica

Summary:

The problem presented deals with classifying textual user inputs into 6 different method classes which indicate which method the user wants to use to solve a quadratic equation. The possible classes are: CompleteSquare, QuadraticFormula, FactorQuadratic and TakeSquareRoots, EliminationMethod, and SubstitutionMethod, where each class is associated with specific mathematical approach. A user might ask as for instructions (help) if they didn't know what to do. All other input is flagged as unrecognized if it doesn't match with any methods. The algorithm should also account for spelling errors. The program should take in textual input and return the name of a method, "help", or "unrecognized".

Assumed Input:

No input needed to run the demo. There will be 5 prompts at the end of the demo to test the neural network analyzer with your own strings. These prompts take a nonempty string of charactersm, for example: "use the quadratic formula". The string must be inside quotation marks. A quick warning though that the results from the user's query might not be as accurate as our examples, because the neural networks require a larger training corpus to understand a broader range of querise. To a certain extent we based our training sentances around the inputs we use as examples; while this isn't the best way to design a real world application of this, it shows the potential of doc2vec to understand semantic word associations.

Algorithm:

We used the doc2vec class of gensim, an open-source topic modeling toolkit, and semi-supervised learning to develop a model that associates certain words and phrases with one of the six method names, or for help.

We began by providing a training corpus. We compiled texts that discuss each of the six methods, as well as texts that attempt to capture certain phrases we believed might be used synonymously with certain methods. After cleaning these files so that they contained only lowercase letters, spaces, and periods, we passed them to gensim along with the associated model names so that gensim could learn what languge we expected it to associate with which methods.

Gensim's doc2vec class expects a sequence of sentences as input. It processes then discards one sentence at a time to avoid storing the whole document. Doc2vec iterates over each sentence twice. The first pass stores and counts each word to build a dictionary. The second pass trains the neural model by building semantic word associations.

We trained the model twice with each document in the training corpus. First we used labeled data for supervised learning by passing in a document and the name of its associated method. Then we used unlabaled data for unsupervised learning by passing in all the documents without specifying which method they are associated with. This trains the model with a broad knowledge base while also

targeting its learning. It teaches the model to associate languge with the general topic of quadratic equations as well as associating it with individual methods of solving quadratic equations.

The parameters of doc2vec can be adjusted to alter the training speed and quality, for example, larger size, which corresponds to degress of freedom, can lead to more accurate training data, but requires a larger corpus. The best way to evaluate the model is to use it on your intended data set. There is no good "objective" evaluation of a gensim model.

For our algorithm we created doc2vec neural networks for each method to heighten their weighted importance in the model when doing comparisons. When given a specific query, we run thorugh every topic and compare the query to the topic's associated method using the topic's asssociated neural model. The topic that returns the highest similarity score is returned as the assumed method of the query.

Resources:

Open source code for spelling checker (http://norvig.com/spell-correct.html), various google searches. Pages used are cited at the bottom of the file to which their content contributed.

Restrictions:

- Additional Package Installation instructions to be found in the README.
- The algorithm is trained with documents found online that have nothing to do with specific audience targeted (we didn't specify whether the input will be from a middle schooler, a PhD student and so forth). The performance of the algorithm should be significantly improved if more suited documents are used for its training (specific textbooks the users are learning from, user age etc.). Previous user inputs should also be used as training documents.
- The training document for spellchecker is a corpus for Sherlock Holmes, so better results for misspelling math words would be attained if there were a larger pool of mathematical documents that could be used for the training. For example "cumplit" is corrected into "culprit" rather than "complete".
- We haven't been able to develop an algorithm that would account for phonetic misspelling. For example, translating "da" into "the" ("cumplit da scware" into "complete the square") and such.
- Complete negation has been addressed, but semeantic negation would require more training. For example, "I want to use quadratic formula and not the complete the square method" will produce the result "quadratic formula," but "I want to use the quadratic formula because complete the square is too complicated" will be associated with both quadratic formula and complete the square.

*If two methods receive a similarity scores that are close together (for example, "quadratic" would likely receive high similarity scores for "quadratic formula" and "factor quadratic"), the model will return whichever is higher. The model could be easily tweaked to return every method with a similarity score over a certain threshold.