How Well Do Computers Solve Math Word Problems? Large-Scale Dataset Construction and Evaluation

Danqing Huang[1]*, **Shuming Shi**[2], **Chin-Yew Lin**[2], **Jian Yin**[1] and **Wei-Ying Ma**[2]
[1] Sun Yat-sen University
{huangdq2@mail2,issjyin@mail}.sysu.edu.cn
[2] Microsoft Research
{shumings,cyl,wyma}@microsoft.com

Abstract

Recently a few systems for automatically solving math word problems have reported promising results. However, the datasets used for evaluation have limitations in both scale and diversity. In this paper, we build a large-scale dataset which is more than 9 times the size of previous ones, and contains many more problem types. Problems in the dataset are semi-automatically obtained from community question-answering (CQA) web pages. A ranking SVM model is trained to automatically extract problem answers from the answer text provided by CQA users, which significantly reduces human annotation cost. Experiments conducted on the new dataset lead to interesting and surprising results.

1 Introduction

Designing computer systems for automatically solving math word problems is a challenging research topic that dates back to the 1960s (Bobrow, 1964a; Briars and Larkin, 1984; Fletcher, 1985). As early proposals seldom report empirical evaluation results, it is unclear how well they perform. Recently, promising results have been reported on both statistical learning approaches (Kushman et al., 2014; Hosseini et al., 2014; Koncel-Kedziorski et al., 2015; Zhou et al., 2015; Roy and Roth, 2015) and semantic parsing methods (Shi et al., 2015). However, we observe two limitations on the datasets used by these previous works. First, the datasets are small. The most frequently used dataset (referred to as Alg514 hereafter) only contains 514 algebra problems. The Dolphin1878 dataset (Shi et al., 2015), the largest collection among them, contains 1878 problems. Second, the diversity of problems in the datasets is low. The Alg514 collection contains linear algebra problems of 28 types (determined by 28 unique equation systems), with each problem type corresponding to at least 6 problems. Although the Dolphin1878 collection has over 1,000 problem types, only number word problems (i.e., math word problems about the operations and relationship of numbers) are contained in the collection.

Due to the above two limitations, observations and conclusions based on existing datasets may not be representative. Therefore it is hard to give a convincing answer to the following question: How well do state-of-the-art computer algorithms perform in solving math word problems?

To answer this question, we need to re-evaluate state-of-the-art approaches on a larger and more diverse data set. It is not hard to collect a large set of problems from the web. The real challenge comes from attaching annotations to the problems. Important annotation types include equation systems (required by most statistical learning methods for model training) and gold answers (for testing algorithm performance). Manually adding equation systems and gold answers is extremely time-consuming[1].

In this paper, we build a large-scale and diverse dataset called Dolphin18K [2], which contains over 18,000 annotated math word problems. It is constructed by semi-automatically extracting problems, equation systems and answers from community question-answering (CQA) web pages. The source data we leverage are the (question, answer text) pairs in the math category of Yahoo! An-

*Work done while this author was an intern at Microsoft Research.

[1] According to our experience, the speed is about 10-15 problems per hour for a person with good math skills.

[2] Available from http://research.microsoft.com/en-us/projects/dolphin/.

Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, pages 887–896,
Berlin, Germany, August 7-12, 2016.

swers[3]. Please note that the answer text provided by CQA users cannot be used directly in evaluation as gold answers, because answer numbers and other numbers are often mixed together in answer text (refer to Figure 1 of Section 3). We train a ranking SVM model to identify (structured) problem answers from unstructured answer text.

We then conduct experiments to test the performance of some recent math problem solving systems on the dataset. We make the following main observations,

1. All systems evaluated on the Dolphin18K dataset perform much worse than on their original small and less diverse datasets.

2. On the large dataset, a simple similarity-based method performs as well as more sophisticated statistical learning approaches.

3. System performance improves sub-linearly as more training data is used. This suggests that we need to develop algorithms which can utilize data more effectively.

Our experiments indicate that the problem of automatic math word problem solving is still far from being solved. Good results obtained on small datasets may not be good indicators of high performance on larger and diverse datasets. For current methods, simply adding more training data is not an effective way to improve performance. New methodologies are required for this topic.

2 Related Work

2.1 Math Word Problem Solving

Previous work on automatic math word problem solving falls into two categories: symbolic approaches and statistical learning methods. In symbolic approaches (Bobrow, 1964a; Bobrow, 1964b; Charniak, 1968; Charniak, 1969; Bakman, 2007; Liguda and Pfeiffer, 2012; Shi et al., 2015), math problem sentences are transformed to certain structures (usually trees) by pattern matching, verb categorization, or semantic parsing. Math equations are then derived from the structured representation. Addition/subtraction problems are studied most in early research (Briars and Larkin, 1984; Fletcher, 1985; Dellarosa, 1986; Bakman, 2007; Yuhui et al., 2010). Please refer to Mukherjee and Garain (2008) for a review of symbolic approaches before 2008.

[3]https://answers.yahoo.com/

Statistical machine learning methods have been proposed to solve math word problems since 2014. Hosseini et al. (2014) solve single step or multi-step homogeneous addition and subtraction problems by learning verb categories from the training data. Kushman et al. (2014) and Zhou et al. (2015) solve a wide range of algebra word problems, given that systems of linear equations are attached to problems in the training set. Seo et al. (2015) focuses on SAT geometry questions with text and diagram provided. Koncel-Kedziorski et al. (2015) and Roy and Roth (2015) target math problems that can be solved by one single linear equation.

No empirical evaluation results are reported in most early publications on this topic. Although promising empirical results are reported in recent work, the datasets employed in their evaluation are small and lack diversity. For example, the Alg514 dataset used in Kushman et al. (2014) and Zhou et al. (2015) only contains 514 problems of 28 types. Please refer to Section 3.4 for more details about the datasets.

Recently, a framework was presented in Koncel-Kedziorsk et al. (2016) for building an online repository of math word problems. The framework is initialized by including previous public available datasets. The largest dataset among them contains 1,155 problems.

2.2 Answer Extraction in CQA

Our work on automatic answer and equation extraction is related to the recent CQA extraction work (Agichtein et al., 2008; Cong et al., 2008; Ding et al., 2008). Most of them aim to discover high-quality (question, answer text) pairs from CQA posts. We are different because we extract structured data (i.e., numbers and equation systems) *inside* the pieces of answer text.

3 Dataset Construction

Our goal is to construct a large and diverse problem collection of elementary mathematics (i.e., math topics frequently taught at the primary or secondary school levels). We build our dataset by automatically extracting problems and their annotations from the mathematics category of the Yahoo! Answers web site. A math problem post on Yahoo! Answers consists of the raw problem text and one or multiple pieces of answer text provided by its answerers (refer to Figure 1).

Please note that posts cannot be used directly as our dataset entries. For example, for training statistical models, we have to extract equation systems from the unstructured text of user answers. We also need to extract numbers (56,000 and 21,000 in Figure 1) from the raw answer text as gold answers. We perform the following actions to the posts,

- Removing the posts that do not contain a math problem of our scope (Section 3.1)
- Cleaning problem text (Section 3.1)
- Extracting gold answers (Section 3.2)
- Extracting equation systems (Section 3.3)

In Section 3.4, we report some statistics of our dataset and compare them with previous ones.

3.1 Preprocessing

We crawl over one million posts from the mathematics categories of Yahoo! Answers. They are part of the posts submitted and answered by users since year 2008. By examining some examples, we soon find that many of them do not contain math problems of our scope. We discard or ignore the posts with the following types:

1. Containing a general math-related question but not a typical math problem. For example, "*Can anyone teach me how to set up two equations for one problem, and then how to solve it after?*".
2. College-level math
3. Containing multiple math problems in a single post. They are discarded for simplifying our answer and equation system extraction process.

As the size of a set of one million problems is large for human annotation and many of them belong to the above three types, we need a way to automatically filter out undesired problems. We manually annotate 6,000 posts with the speed of about 150 posts per hour per person. Then a logistic regression classifier of posts is trained with a precision of 80% and a recall of 70%. The post collection after classification is reduced to 120,000 posts.

Then we randomly sample 46,000 posts from the reduced post collection to perform two actions manually: post classification and problem text cleaning. Please note that, since the precision of the automatic classifier is only 80%, we rely on manual classification to remove the remaining 20% undesired posts. Problem text cleaning is for removing sentences like "please help" and "Son's 6th grade math" (refer to Figure 1). The problem text after cleaning is just like that appearing in a formal math test in an elementary or secondary school.

Question part:
Son's 6th grade math? The number of cans produced in one day by two companies A and B were in ratio 8:3 and their difference was 35,000. How many cans did each company produce that day?

Answer part:
Answer 1: Let can produced by 1 company be 8x and the other 3x. so 8x - 3x = 35000. 5x = 35000. x = 7000. So the first company produced 8 x 7000 = 56,000 cans, and the other produced 3 x 7000 = 21,000 cans.
Answer 2: From the ratio: 3A=8B or A=(8/3)B. From the difference: A-B=35000. By substituting for A, we get (8/3)B-B=35000 and further to B = 21000. From the difference: A = 21000+35000=56000.
Answer 3: It's 56000 and 21000.
Answer 4: what the hell thats not 6th grade math!!!

Figure 1: An example post from Yahoo! answers

Eight annotators participated in the manual post classification and problem text cleaning, at an average speed of about 80 posts per hour per person.

3.2 Automatic Answer Extraction

Compared to post classification and problem text cleaning, it is much more time consuming to manually assign gold answers and equation systems to a problem (10-15 problems per hour per person vs. 80 posts per hour per person). In addition, the latter has higher requirements of the math skills of annotators. Since manually annotating all problems exceeds our budget, we choose to train a high precision model to automatically extract numbers as gold answers from the answer part of a post.

In our dataset, the gold answer to a problem is one or a set of numbers acting as the solution to the problem. We define *answer dimension* as the

count of numbers required in the gold answer. For example, the gold answer to the problem in Figure 1 is {56000, 21000}, with dimension 2.

Extracting gold answers from the answer part of a post is nontrivial. We tried an intuitive approach called last-number-majority-voting, where the last number in each answer of the post is chosen as a candidate and then majority voting is performed among all the candidates. We got a low accuracy of 47% on our annotated data. Thus, we turn to a machine learning model for better utilizing more features in the posts.

Notations: Let χ denote the set of training problems. For each problem x_i in χ, $N_{ij} = \{n_{ij}^1, n_{ij}^2, \ldots, n_{ij}^m\}$ denote the set of all unique numbers given the *j*th answer, where *m* represents the size of N_{ij}. For each N_{ij}, we generate possible subsets of numbers as candidate answers to the problems. Please pay attention that the gold answer to a problem may contain multiple numbers (in the case that the answer dimension is larger than 1). We use Y_i to denote all the candidate answers in problem x_i.

Model: We define the conditional probability of $y_{ik} \in Y_i$ given x_i:

$$p(y_{ik}|x_i;\nu) = \frac{\exp(\nu \cdot f(x_i, y_{ik}))}{\sum_{y'_{ik} \in Y_i} \exp(\nu \cdot f(x_i, y'_{ik}))}$$

where ν is a parameter vector of the model and $f(x_i, y_{ik})$ is the feature vector. We apply the Ranking SVM (Herbrich et al., 2000) to maximize the margin between the correct instances and the negative ones. Constructing the SVM model is equivalent to solving the following Quadratic Optimization problem:

$$\min_{\nu} M(\nu) = \frac{1}{2}\|\nu\|^2 + C\sum_i \xi_i$$

$$s.t.\ \xi_i \geq 0, \nu \cdot \langle f(x_i, y_{ik})^+ - f(x_i, y_{il})^- \rangle \geq 1 - \xi_i$$

where subscript "+" indicates the correct instance and "-" indicates the false ones.

Features: Features are extracted from the answer part of each post for model training. We design features based on the following observations. In Yahoo! answers, users tend to write down correct answers at the beginning of the answer text, or at the end after providing the solving procedure. Surrounding words also give hints for finding correct solutions. For example, numbers that are close to the word *"answer"* are more likely to be in the gold answer. Given a post, numbers appearing in the answer text of more users are more likely to be the correct solution. Some words in the question sentence help determine answer dimension. For example, *"How far does Tom run?"* requires a one-dimension answer while *"How much do they each earn?"* indicates multiple dimensions. Main features are listed in table 1.

Table 1: Features for automatic answer extraction

Local context features
Relative position in the procedure
On right side of the symbol "="?
On left side of the symbol "="?
Close to *"ans"*, *"answer"*, *"result"*, or *"therefore"*
Global features
Is in the text of the first answer (the first answer is often marked as the best answer in Yahoo! answers)?
Is in problem text?
Frequency in the text of all answers for this problem
Frequency in the first position of all answers
Frequency in the last position of all answers
Number value features
Is positive?
Is an integer?
Its value is between 0 to 1?
Equals to the predicted solution in automatic equation extraction?
Number set features
Are numbers at same line of answer text?
Are numbers at consecutive lines of answer text?
Frequency of the numbers at same line in all answers
Frequency of the numbers at consecutive lines in all answers
Dimension features
Has singular verb in question?
Has plural noun in question?
Has special words (e.g., *and, both, each, all*) in question?

Inference: After we train the model to get parameter vector ν, the predicated gold answer is selected from the candidate number subsets by maximizing $\nu \cdot f(x_i, y_{ik})$. Formally, the predicated gold

answer is,

$$\arg\max_{y_{ik} \in Y_i} \nu \cdot f(x_i, y_{ik})$$

About 3,000 problems are manually annotated with answers and equations by the human annotators we hire. Then we train and evaluate our model using 5-fold cross validation. The extractor's performance is shown in Figure 2. To preserve an accuracy rate of 90%, we use score = 3 as the threshold and only keep problems with predicted confidence score >= 3.

Please note that precision is more important than recall in our scenario. We need to guarantee that most extracted answers are correct. Lower recall can be tackled by processing more posts.

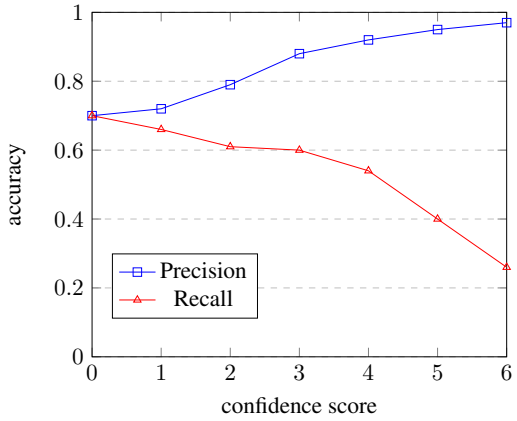

Figure 2: Accuracy of answer extraction

3.3 Automatic Equation Annotation

Now we illustrate how to extract equation systems automatically from the unstructured answer text of a post. The input is the answer text of n answers:

$$T = \{T_1, T_2, \ldots, T_n\}$$

For example in Figure 1, there are four answers, each corresponding to a piece of answer text T_i.

The task is not easy, because variables and equations may not be in standard formats in answer text. In addition, equations may be duplicate (like those in Answer 1 of Figure 1). Our algorithm is a two-phase procedure:

Candidate extraction: We extract an equation system from each piece of answer text T_i. In processing T_i, we first extract a list of equations by regular expression matching. Then the equations are added to the equation system by the order of their occurrences in the text. Before adding an equation, we check whether it can be induced by the already-added equations. If so, we skip it. Duplicate equations are effectively reduced in this way.

Voting by solution: We solve each equation system obtained from the first phase and build a (equation system, solution) bipartite graph. We then choose the equation system that has the maximum degree as our output. For example, if three equation systems return the solution {24} and the fourth returns {-1}, we will choose one from the first three equation systems. To improve precision, we do not return any equation system if the maximal degree is less than 2.

We evaluate our equation extractor on 3,000 manually annotated problems [4]. For an equation system extracted for a problem, we say it is correct if the annotated gold answer is a subset of the solutions to the equation system. For example, if the gold answer is {16, 34} and the solution to the equation system is {16, 34, 100}, then the equation system is considered correct. Evaluation results show a precision of **91.4%** and a recall of **64.7%**.

3.4 Datasets Summary

Below are a list of previous benchmark datasets for math word problem solving.

Alg514 is introduced in Kushman et al. (2014) and also used in Zhou et al. (2015) for evaluation. It consists of 514 algebra word problems from algebra.com[5], with each problem annotated with linear equations. The *template* (explained later) of each problem has to appear at least six times in the whole set.

Verb395 (Hosseini et al., 2014): A collection of addition/subtraction problems.

Dolphin1878: A collection built by Shi et al. (2015), containing 1,878 number word problems obtained from algebra.com and Yahoo! answers.

DRAW (Upadhyay and Chang, 2015): Containing 1,000 algebra word problems from algebra.com, each annotated with linear equations.

SingleEQ: By Koncel-Kedziorski et al. (2015), containing 508 problems, each of which corresponds to one single equation.

Before comparing the datasets, let's first introduce the concept of *equation system templates*, which are first introduced in Kushman et al. (2014)

[4] the same set of problems as we used in training and evaluating answer extraction

[5] http://www.algebra.com

Table 2: Comparison of different datasets

Dataset	# Problems	# Templates	# Sentences	# Words	Problems types
Verb395	395	3	1.13k	12.4k	homogeneous addition or subtraction problems
Alg514	514	28	1.62k	19.3k	algebra, linear
Dolphin1878	1,878	1,183	3.30k	41.4k	number word problems
DRAW	1,000	232	2.67k	35.3k	algebra, linear
SingleEQ	508	31	1.38k	13.8k	single equation, linear
Dolphin18K	18,460	5,871	49.9k	604k	linear + nonlinear

for math word problem solving. A *template* is a unique form of equation system. For example, the following is a template of two equations:

$$n_1 \cdot x_1 + x_2 = n_2$$
$$x_1 + n_3 \cdot x_2 = n_4$$

The following equation system corresponds to the above template,

$$3 \cdot x_1 + x_2 = 5$$
$$x_1 + 7 \cdot x_2 = 15$$

Table 2 shows some statistical information of our dataset and previous ones. It can be seen that our dataset has a much larger scale (about 10 times the size of the Dolphin1878 collection and more than 17 times larger than the others) and higher diversity (in terms of both problem types and the number of templates contained).

We split our dataset into a development set and an evaluation set. The development set is used for algorithm design and debugging, while the evaluation set is for training and testing. Any problem in the evaluation set should be invisible to the people who design an automatic math problem solving system. Statistics on our dataset are shown in Table 3, where *dev* and *eval* represent the development set and the evaluation set respectively. Most problems are assigned with both equation systems and gold answers. Some of them are annotated with answers only, either because annotators feel it is hard to do so, or because our equation extraction algorithm returns empty results.

As most previous systems only handle linear equation systems, we summarize, in Table 4, the distribution of linear problems in the evaluation set by template size. In the table, the size of a template is defined as the number of problems corresponding to this template. Between the two numbers in each cell, the first one is the number of problems, and the second number (or the one in parentheses) is the number of templates in this category. For example, in the automatically annotated evaluation set, 166 templates have size 6 or larger. They correspond to 4,826 problems.

Table 3: Annotation statistics for our dataset

		Equations + answer	Answer only	Sum
dev	Manual	909	67	976
	Auto	2,245	507	2,752
	All	3,154	574	3,728
eval	Manual	3,605	321	3,926
	Auto	8,754	2,052	10,806
	All	12,359	2,373	14,732

4 Experiments

4.1 Systems for evaluation

We report the performance of several state-of-the-art systems on our new dataset.

KAZB: A template-based statistical learning method introduced in Kushman et al. (2014). It maps a problem to one equation template defined in the training set by reasoning across problem sentences. KAZB reports an accuracy of 68.7% on the Alg514 dataset.

ZDC: Proposed in Zhou et al. (2015) as an improved version of KAZB. It reduces the search space by not modeling alignment between noun phrases and variables. It achieves an accuracy of 79.7% on Alg514.

SIM is a simple similarity-based method implemented by us. To solve a problem, it calculates the lexical similarity between the problem and each problem in the training set. Then the equation system of the most similar problem is ap-

Table 4: Problem distribution by template size (for linear problems only)

Template size	Manual	Auto	All
(all linear templates)	2,675 (876)	7,969 (2,609)	10,644 (3,158)
>=2	2,036 (237)	5,956 (596)	8,229 (743)
>=5	1,678 (98)	4,979 (196)	7,081 (254)
>=6	1,578 (78)	4,826 (166)	6,827 (216)
>=10	1,337 (43)	4,329 (96)	6,216 (130)
>=20	1,039 (22)	3,673 (48)	5,392 (68)
>=50	634 7	2,684 18	4,281 30

plied to the new problem. In a little more details, a test problem P_T is solved in two steps: template selection, and template slot filling. In the first step, each problem is modeled as a vector of word TF-IDF scores. The similarity between two problems is calculated by the weighted Jaccard similarity between their corresponding vectors. We choose, from the training data, problem P_1 that has the maximal similarity with P_T and use the equation template T of P_1 as the template of problem P_T. In the second step, the numbers appearing in problem P_T are mapped to the number slots of template T (which has been identified in the first step). The mapping is implemented by selecting one problem P_2 from all the training problems corresponding to template T so that it has the minimum word-level edit-distance to P_T. Then the number mapping of P_2 is borrowed as the number mapping of P_T. For example, for the following test problem,

An overnight mail service charges $3.60 for the first six ounces and $0.45 for each additional ounce or fraction of an ounce. Find the number of ounces in a package that cost $7.65 to deliver.

Assuming that a problem P_1 has maximum Jaccard similarity with the above problem and its corresponding equation template is as follows, this template will be identified in the first step,

$$n_1 + n_2 * (x - n_3) = n_4$$

Assume that P_2 has the minimum edit-distance to P_T among all the training problems corresponding to template T. Suppose the numbers in P_2 are (by their order in the problem text),

$$3.5, 5, 0.5, 6.5$$

Also suppose P_2 is annotated with the following equation system,

$$3.5 + 0.5 * (x - 5) = 6.5$$

Then we will choose P_2 and borrow its number mapping. So the mapping from numbers in the above test problem to template slots will be,

$$3.60/n_1; 6/n_3; 0.45/n_2; 7.65/n_4$$

In implementing SIM, we do not use any POS tagging or syntactic parsing features for similarity calculation. This method gets an accuracy of 71.2% on Alg514 and 49.0% on SingleEQ.

Systems not included for evaluation: Although the system of Shi et al. (2015) achieves very high performance on number word problems, we do not include it in our evaluation because it is unknown how to extend it to other problem types. The system of Hosseini et al. (2014) is not included in our evaluation because it only handles homogeneous addition/subtraction problems. The systems of Koncel-Kedziorski et al. (2015) and Roy and Roth (2015) are also not included because so far they only supports problems with one single linear equation.

4.2 Overall Evaluation Results

Table 5 shows the accuracy of various systems on different subsets of our dataset. In the table, Manual.Linear contains all the manually annotated problems with linear equation systems. It contains 2,675 problems and 876 templates (as shown in Table 4). Auto.LinearT6 (containing 4,826 problems) is the set of all the automatically annotated problems with a template size larger than or equal to 6. Similarly, LinearT2 means the subset of problems with template size ≥ 2. For each system on each subset, experiments are conducted using 5-fold cross-validation with 80% problems randomly selected as training data and the remaining 20% for test.

In the table, "-" means that the system does not complete running on the dataset in three days. Since KAZB and ZDC only handle linear equation systems, they are not applicable to the datasets

	Systems		
Dataset	KAZB	ZDC	SIM
Manual.Linear	10.7%	11.1%	13.3%
Manual.LinearT2	12.8%	13.9%	17.3%
Manual.LinearT6	17.6%	17.1%	18.8%
Auto.Linear	-	17.2%	17.4%
Auto.LinearT2	-	20.1%	19.2%
Auto.LinearT6	-	19.2%	18.4%
All.Linear	-	17.9%	18.4%
All.LinearT2	-	20.6%	20.3%
All.LinearT6	-	21.7%	20.2%
All (Dolphin18K)	n/a	n/a	16.7%
Alg514	68.7%	79.7%	71.2%

Table 5: Overall evaluation results

containing nonlinear problems. An "n/a" is filled in the corresponding cell in this case.

The results show that all three systems (KAZB, ZDC, and SIM) have extremely low performance on our new datasets. Surprisingly, no system achieves an accuracy rate of over 25%. Such results indicate that automatic math word problem solving is still a very challenging task.

Another surprising observation is that KAZB and ZDC do not perform better than SIM, a simple similarity-based method which runs much faster than the two statistical learning systems.

By comparing the results obtained from the manual version of the datasets with their corresponding auto version (for example, Manuall.Linear vs. Auto.Linear), we can see larger accuracy scores on the auto versions [6]. This demonstrates the usefulness of the automatically annotated data. Considering the huge cost of manually assigning equation systems and gold answers, automatic annotation has good potential in constructing larger datasets.

4.3 Why Different from Previous Results

The last line of Table 5 displays the results on Alg514. All three systems perform well on Alg514 but poorly on Dolphin18K. To study the reason of such a large gap, we derive two small datasets from All.Linear by referring to the equation templates in Alg514.

Small.01: The set of all problems in All.Linear that correspond to one of the 28 templates in Alg514. The dataset contains 2,021 problems.

[6]Please note that the auto versions are more than 2 times larger.

Small.02: A subset of Small.01, constructed by randomly removing problems from Small.01 so that each template contains similar number of problems as in Alg514. In other words, Small.02 and Alg514 have similar problem distribution among templates.

	Small.01	Small.02
KAZB	29.9%	50.0%
ZDC	30.1%	52.7%
SIM	33.7%	43.0%

Table 6: The case of fewer number of templates

We still use 5-fold cross validation to test and compare system performance on the two small datasets. Evaluation results are displayed in Table 6. We now obtain higher accuracy scores for each system, but there is a big difference between the results on Small.01 and Small.02. As mentioned in (Upadhyay and Chang, 2015), Alg514 has a skewed problem distribution, with a few templates covering almost 50% problems. This may be the main reason why all three systems achieve high accuracy on this dataset and on Small.02. From all of the above results, we see at least two factors which affect system performance: number of templates in the dataset, and the distribution of problems among the templates. For a small dataset, the distribution of problems among templates have a huge impact on evaluation results.

4.4 Effect of Training Data Size

Now we investigate the performance change of various systems when the size of training data changes. The goal is to check whether the accuracy increases quickly when more training data are added. This is important: If it is the case, we can tackle this task by simply adding more training data, either manually or automatically. Otherwise, we have to discover new approaches.

We conduct experiments in two settings: fixed-test-set, and increasing-test-set. In the first setting, we randomly choose 1/2 of the problems from the Manual.Linear subset to form a fixed test-set (with size 1330). Then the other problems in All.Linear forms a candidate training collection (containing 9314 elements). We construct training sets of different scales by doing random sampling from the candidate training collection.

In the second setting (i.e., increasing-test-set), we construct datasets (training set plus test set) of

Training data source	All.LinearT6					All.Linear				
Training data size	138	434	1024	2940	5771	500	1000	2000	5000	9000
Test set size	1330	1330	1330	1330	1330	1330	1330	1330	1330	1330
KAZB accuracy (%)	6.7	7.2	-	-	-	7.1	8.3	-	-	-
ZDC accuracy (%)	6.1	7.5	8.6	11.4	12.6	5.5	9.2	10.5	12.5	13.1
SIM accuracy (%)	5.5	8.7	11.0	13.7	15.9	6.5	10.8	12.2	14.9	18.4

Table 7: System performance with different training data size (setting: fixed-test-set)

Training data size	400	800	1600	4000	8516
Test set size	100	200	400	1000	2128
KAZB	5.4%	6.7%	11.7%	-	-
ZDC	5.8%	7.6%	12.9%	17.0%	17.9%
SIM	7.4%	10.0%	13.3%	16.9%	18.4%

Table 8: System performance with different training data size (setting: increasing-test-set)

different scales by doing random sampling from All.Linear, and then conduct 5-fold cross validation on each dataset. In each fold, 80% problems are chosen at random for training, and the other 20% for testing.

The results in the two settings are reported in Tables 7 and 8 respectively. Both tables show that the accuracy of all the three systems improves steadily but slowly along with the increasing of training data size. So it is not very effective to improve accuracy by simply adding more training data.

4.5 Results Summary

In summary, the following observations are made from the experiments on our new dataset. First, all systems evaluated on the Dolphin18K dataset perform much worse than on the small and less diverse datasets. Second, the two statistical learning methods do not perform better than a simple similarity-based method. Third, it seems not promising for the current methods to achieve much better results by simply adding more training data. Automatic math word problem solving is still a very challenging task so far.

5 Conclusion

We have constructed Dolphin18K, a large dataset for training and evaluating automatic math word problem solving systems. The new dataset is almost one order of magnitude larger than most of previous ones, and has a much higher level of diversity in term of problem types. We reduce human annotation cost by automatically extracting gold answers and equation systems from the unstructured answer text of CQA posts.

We have also conducted experiments on our dataset to evaluate state-of-the-art systems. Interesting and surprising observations are made from the experimental results.

Acknowledgments

We would like to thank the annotators for their efforts in annotating the math problems in our dataset. Thanks to the anonymous reviewers for their helpful comments and suggestions.

References

Eugene Agichtein, Carlos Castillo, Debora Donato, Aristides Gionis, and Gilad Mishne. 2008. Finding high-quality content in social media. In *First ACM International Conference on Web Search and Data Mining (WSDM'08)*.

Yefim Bakman. 2007. Robust understanding of word problems with extraneous information. http://arxiv.org/abs/math/0701393.

Daniel G. Bobrow. 1964a. Natural language input for a computer problem solving system. Technical report, Cambridge, MA, USA.

Daniel G. Bobrow. 1964b. Natural language input for a computer problem solving system. Ph.D. Thesis.

Diane J. Briars and Jill H. Larkin. 1984. An integrated model of skill in solving elementary word problems. *Cognition and Instruction*, 1(3):245–296.

Eugene Charniak. 1968. Carps, a program which solves calculus word problems. Technical report.

Eugene Charniak. 1969. Computer solution of calculus word problems. In *Proceedings of the 1st International Joint Conference on Artificial Intelligence*, pages 303–316.

Gao Cong, Long Wang, Chin-Yew Lin, Young-In Song, and Yueheng Sun. 2008. Finding question-answer pairs from online forums. In *Proceedings of 31st International ACM-SIGIR Conference on Research and Development in Information Retrieval (SIGIR'08)*, pages 467–474.

Denise Dellarosa. 1986. A computer simulation of children's arithmetic word-problem solving. *Behavior Research Methods, Instruments, & Computers*, 18(2):147–154.

Shilin Ding, Gao Cong, Chin-Yew Lin, and Xiaoyan Zhu. 2008. Using conditional random fields to extract context and answers of questions from online forums. In *Proceedings of the 46th Annual Meeting of the ACL: HLT (ACL 2008)*, pages 710–718, Columbus, USA.

Charles R. Fletcher. 1985. Understanding and solving arithmetic word problems: A computer simulation. *Behavior Research Methods, Instruments, & Computers*, 17(5):565–571.

Ralf Herbrich, Thore Graepel, and Klaus Obermayer, 2000. *Large Margin Rank Boundaries for Ordinal Regression*, chapter 7, pages 115–132.

Mohammad Javad Hosseini, Hannaneh Hajishirzi, Oren Etzioni, and Nate Kushman. 2014. Learning to solve arithmetic word problems with verb categorization. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, October.

Rik Koncel-Kedziorsk, Subhro Roy, Aida Amini, Nate Kushman, and Hannaneh Hajishirzi. 2016. Mawps: A math word problem repository. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*.

Rik Koncel-Kedziorski, Hannaneh Hajishirzi, Ashish Sabharwal, Oren Etzioni, and Siena Dumas Ang. 2015. Parsing algebraic word problems into equations. *Transactions of the Association for Computational Linguistics*, 3:585–597.

Nate Kushman, Yoav Artzi, Luke Zettlemoyer, and Regina Barzilay. 2014. Learning to automatically solve algebra word problems. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*.

Christian Liguda and Thies Pfeiffer. 2012. Modeling math word problems with augmented semantic networks. In *Natural Language Processing and Information Systems. International Conference on Applications of Natural Language to Information Systems (NLDB-2012)*, pages 247–252.

Anirban Mukherjee and Utpal Garain. 2008. A review of methods for automatic understanding of natural language mathematical problems. *Artificial Intelligence Review*, 29(2):93–122.

Subhro Roy and Dan Roth. 2015. Solving general arithmetic word problems. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1743–1752. The Association for Computational Linguistics.

Minjoon Seo, Hannaneh Hajishirzi, Ali Farhadi, Oren Etzioni, and Clint Malcolm. 2015. Solving geometry problems: Combining text and diagram interpretation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*.

Shuming Shi, Yuehui Wang, Chin-Yew Lin, Xiaojiang Liu, and Yong Rui. 2015. Automatically solving number word problems by semantic parsing and reasoning. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*.

Shyam Upadhyay and Ming-Wei Chang. 2015. Draw: A challenging and diverse algebra word problem set. Number MSR-TR-2015-78, October.

Ma Yuhui, Zhou Ying, Cui Guangzuo, Ren Yun, and Huang Ronghuai. 2010. Frame-based calculus of solving arithmetic multistep addition and subtraction word problems. *Education Technology and Computer Science, International Workshop*, 2:476–479.

Lipu Zhou, Shuaixiang Dai, and Liwei Chen. 2015. Learn to solve algebra word problems using quadratic programming. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*.