AUTOMATIC ASSESSMENT GENERATION VIA MACHINE LEARNING

A Design Project Report

Presented to the School of Electrical and Computer Engineering of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Master of Engineering, Electrical and Computer Engineering

Submitted by

Arjun Jauhari

MEng Field Advisor: Christoph Studer

MEng Outside Advisor: Igor Labutov

Degree Date: May 2016

Abstract

Master of Engineering Program

School of Electrical and Computer Engineering

Cornell University

Design Project Report

Automatic Assessment Generation via Machine Learning

Arjun Jauhari

Abstract

This project is part of Machine Learning and Education Research aimed to build an automatic system for generation of test/assessment. The project uses data dumps from 130+ stack-exchange websites, with $\sim$5 million users $\sim$30 million posts, to learn interesting hidden variables like difficulty of question, quality of answers, ability of users in a particular specialization. The goal is to use these learned parameters to build applications like generating an assessment, tracking the learning curve of a student, ranking all the users. Data on these 130+ stack-exchange websites is huge, therefore a careful review is done to understand how data is arranged, what different attributes mean. It is then processed to extract out the important attributes. Various approaches like Linear system of equations, Naive Bayes, Probabilistic Graphical models have been explored to learn the hidden variables. It is found that Graphical models give best results on both simulation and real world data. We are able to achieve $\sim$ 20% better than the baselines.

Executive Summary

The goal of this design project was to solve the problem of assessment generation. Currently available method for generating a test requires an excessive amount of human (generally a teacher or domain expert) effort. Our method provides an automated way of generating test with almost no human effort. Moreover, our method can generate new assessments instantly and even personalize them. It can have a mix of both theoretical as well as practical questions. To achieve this goal the main challenge was to learn interesting latent variables like difficulty of question, quality of answers, and ability of users from the observed clicks captured in the StackExchange dataset.

The dataset was huge, which was beneficial for learning task but posed serious processing issues which required critical thinking during the design process of the system. We developed a new algorithm, which is slightly modified version of MapReduce algorithm, to process these data files with million of lines. We successfully processed these files to extract useful attributes and that too in very efficient and fast manner. For instance, we were able to process `Votes.xml` file from StackOverflow.com with $\sim$99 million lines in $\sim$15 minutes and memory usage of $\sim$3 GB. Our modified MapReduce algorithm can trade-off very effectively between time and memory requirements.

The above algorithm compressed the files by throwing away attributes which were not useful to us but still the data was not in format which can be used directly by any learning algorithm. Therefore, next we formatted data into various useful Data Structures. Most of these Data Structures were based on HashMap.

For the learning task, we explored couple of models but spent most of the time on Probabilistic Graphical Model based approach. Graphical models are natural choice for Data Science problems similar to the one we are trying to solve. This design choice proved to be quite successful and we were able to achieve good results. We used a simple graphical model which captures the intuition behind the data generation. The intuition it captures is that every user has some ability (ϕ) and he generates answers (θ) which are normally (Gaussian) distributed around the user's ability. Now the observed clicks are modelled with intuition that a better answer is more probable of getting an UpVote. We used SoftMax function to capture this intuition.

Even for this simple looking model the implementation was very complex as the number of parameters in the model are equal to number of answers + number of users, which for StackOverflow are in the range of $\sim$10 million even after filtering. This poses both memory and time constraint. Therefore, we first chose dataset like Maths, Physics and Chemistry where the number of parameters were in range of thousands. We used Maximum Likelihood Estimation to learn the most likely parameters given the observed data. To solve this optimization problem we used gradient descent. Specifically, we used L-BFGS for smaller datasets and for larger dataset we used AdaGrad which is an adaptive stochastic gradient descent method.

The planning and implementation of the project went well and smoothly. We were able to train the model and do predictions from the learned model with an accuracy of order $\sim$50% which was 20% better than random guessing. This accuracy is quite good as this problem is a varying multi-class classification problem.

Contents

1 Introduction

Recently, use of Internet has become ubiquitous in Education. There is a rapid growth in volume of academic texts available online lecture excerpts from courses, books chapters, Scholarpedia pages, and sources like technical blogs, Stack Exchange posts, Wikipedia. All these sources available today help a learner(student) to enhance their knowledge without depending much on other human but there are not sufficient ways available to assess one's knowledge in a particular discipline without using an expert(teacher) generated test. On the other hand, there is no scarcity of quality questions and corresponding answers on the Internet, specifically on websites like Stackoverflow.com. Goal of this project is to use this huge pool of questions & answers to develop a novel system which can create assessments automatically with minimal human effort. Moreover, this system can generate new assessments almost instantly and personalize them for a particular user.

Through this system we are providing an automated tool that can generate an assessment which a user can use to test his/her skills. Currently available methods for generating a test requires an excessive amount of human (generally a teacher or domain expert) effort. Also, the traditional approach to assessment generation does not scale to test in multiple fields or even personalized tests. Our approach mitigate all these issues and provides a more scalable solution.

The proposed solution uses machine learning to automatically generate assessment using already available million of questions and answers from websites like Stackoverflow.com. The main challenge was to learn interesting hidden variables like difficulty of question, quality of answers, and ability of users.

2 Dataset

We use the data available on various Stack Exchange websites to generate tests automatically. But the scope of project is not limited to Stack Exchange only it can easily be extended to other similar websites like Quora.
Some of the key aspects of stackexchange dataset are listed below:

- 130+ stackexchange websites, most famous one is stackoverflow.com
- Number of users = 5,277,830 ($\sim$ 5 million), file size 1.5 GB
- Number of posts (Question + Answers) = 29,499,662 ($\sim$ 30 million), file size 45 GB
- Number of votes = 98,928,934 ($\sim$ 99 million), file size 9 GB

As can be seen the dataset size is huge, which is both good and bad. More data is good for the machine learning algorithm to learn and generalize better to different scenarios. But on the other hand, it becomes very challenging to process such huge amount of data which poses both time and memory constraints. To solve this challenge we used simplified version of MapReduce algorithm which is described in more details later.

Stack Exchange data is structured into following 8 files:

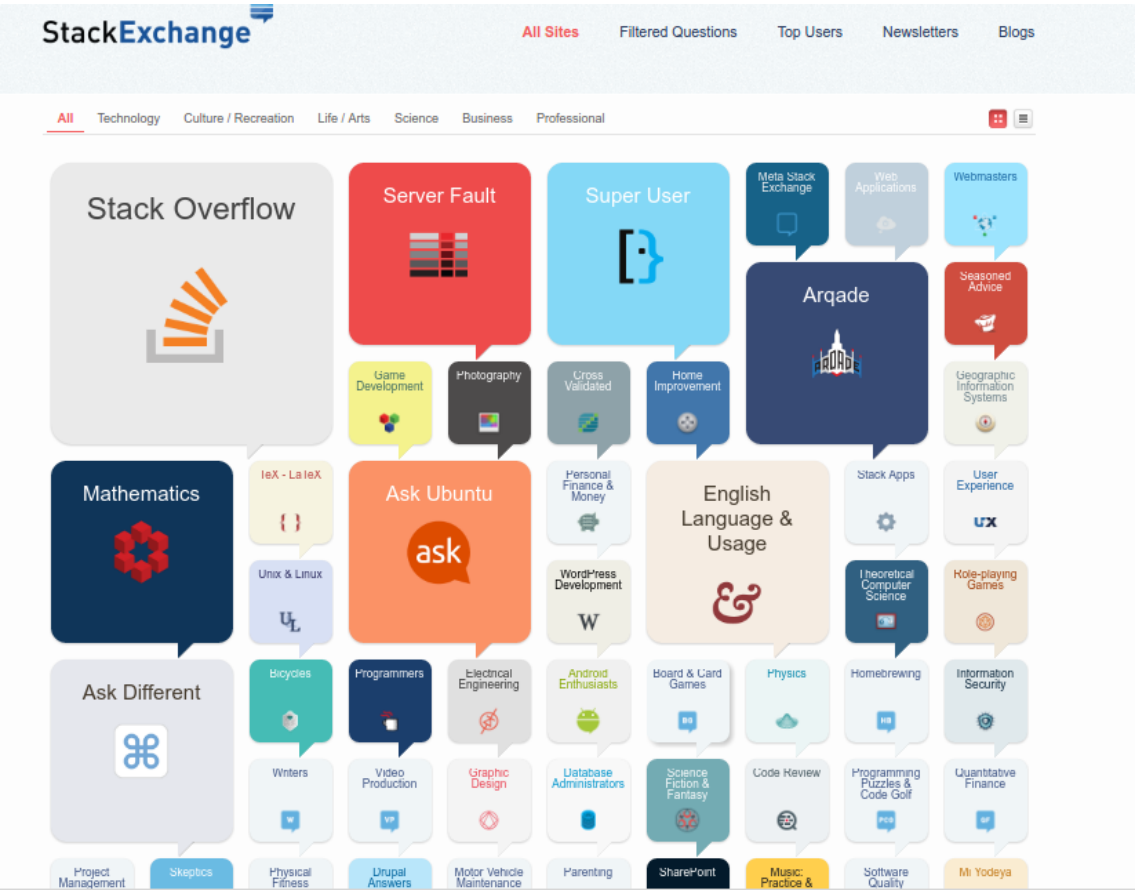

Figure 1: Dataset: Various StackExchange websites

- Badges.xml
- **Comments.xml**
- PostHistory.xml
- PostLinks.xml
- **Posts.xml**
- Tags.xml
- **Users.xml**
- Votes.xml

Although all of these files contains important information but the one's which are in bold are most relevant for the implementation of this project. Each line in these files corresponds to a unique Post, User, etc defined by a row tag. All information is stored under attributes of each row. For example a sample row looks like -

$< row$**Id** $=$ "1"**PostTypeId** $=$ "1"**CreationDate** $=$ "$2015-02-03T16:40:26.487$"**Score** $=$ "22"**ViewCount** $=$ "307"**Body** $=$ "$SampleQuestion$"**OwnerUserId** $=$ "2"**LastEditorUserId** $=$ "2"**LastEditDate** $=$ "$2015-02-03T17:51:07.583$"**LastActivityDate** $=$ "$2015-02-03T21:05:27.990$"**Title** $=$ "$SampleTitle$"**Tags** $=$ "$line-numbers$"**AnswerCount** $=$ "2"**CommentCount** $=$ "0"**FavoriteCount** $=$ "3"$/>$

3 Pre-Processing of Dataset

3.1 Extracting Attributes

To extract the important attributes from the above mentioned data files we wrote shell scripts. Below is a list of attributes used for the project:

- From Posts.xml file
  - Id
  - PostTypeId
  - OwnerUserId
  - ParentId(onlyforans)
  - Score
  - AcceptedAnswerId
  - CreationDate
  - AnswerCount
- From Users.xml file
  - Id
  - DisplayName
  - Reputation
  - UpVotes
  - DownVotes
- From Votes.xml file
  - Id
  - PostId
  - VoteTypeId
  - CreationDate

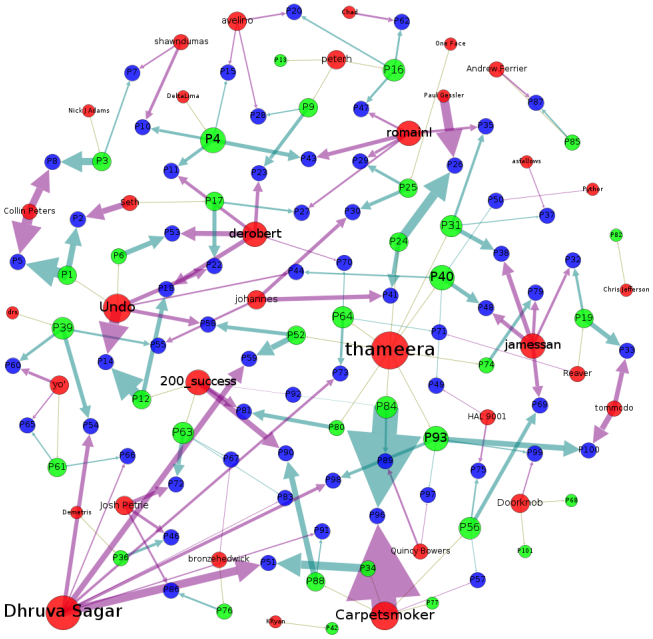

Figure 2: Interaction graph of Users with Posts

3.2 Generating Gephi graphs

To visualise a small subset of data, we generated graphs using Gephi. This helped us in visualizing how Users are connected to Questions and Answers. Figure 2 above shows one graph on a very small subset of data. Legend for the graph-

- Red Nodes: User
- Green Nodes: Questions
- Blue Nodes: Answers
- Green Edge: User2Question(u2q)
- Blue Edge: User2Answer(u2b)
- Grey Edge: Question2Answer(q2a)

The edges connect each user to their post. Post can be a question(green edge) or an answer(blue edge). Also, each question is connected to its answers through grey edge.

Important details about the graph -

- Node size represents, how active is that particular user. For example, user with name 'thameera' is the one of the most active user followed by user 'Dhruv Sagar'.
- The thickness of these q2a edges tells the quality of an answer relative to all the other answers for same parent question. For example, user 'Carpet Smoker' has give one of the best answer.

4 Statistical Model Design

4.1 Modelling as System of Linear Equations

This is a very simple approach where we model the whole system through set of linear equations as described below in detail.

4.1.1 Subscripts

- i is subscript for question.
- j is subscript for answer.
- k is subscript for user.

4.1.2 Notations

1. u_k: Quality measure of the kth user.
2. q_i: Quality measure of the ith question.
3. va_{ij}: Normalized votes corresponding to jth answer of ith question.
   Calculated as: $va_{ij} = \frac{|sa_{ij}|}{\sum_j |sa_{ij}|}$
   where sa_{ij} is the actual votes(score) read from data dump.
4. a_{ijk}: Quality measure of a_{ij}th answer given by the kth user.
5. f_{acc}^{ij}: Boolean flag telling if this answer was Accepted, read from data dump.
6. r_k: Reputation of the kth user, read from data dump.
7. N_a^i: Number of answer to ith question, read from data dump.
8. vq_i: Number of votes to ith question, read from data dump.

4.1.3 Equations

Below equations model the relation/dependence between the above defined parameters. Bold values are known features. All the features were scaled between 0 & 1

1. $a_{ijk} = f_a(u_k, \mathbf{va_{ij}}, \mathbf{f_{acc}^{ij}})$
   $a_{ijk} = 1/3 * u_k + 1/3 * \mathbf{va_{ij}} + 1/3 * \mathbf{f_{acc}^{ij}}$
2. $u_k = f_u(\{a_{ijk}\}_{ij}, \{q_i\}_k, \mathbf{r_k})$, where $\{a_{ijk}\}_{ij}$ is set of all answers by user k
   $u_k = 1/3 * mean\{a_{ijk}\}_{ij} + 1/3 * mean\{q_i\}_k + 1/3 * \mathbf{r_k}$
3. $q_i = f_q(u_k, \mathbf{N_a^i}, \mathbf{vq_i}, \sum_{\mathbf{j}} |\mathbf{sa_{ij}}|)$, where $\sum_j |sa_{ij}|$ is sum of votes for all the answers of ith question
   $q_i = 1/4 * u_k + 1/4 * \mathbf{N_a^i} + 1/4 * \mathbf{vq_i} + 1/4 * \sum_{\mathbf{j}} |\mathbf{sa_{ij}}|$

4.1.4 Solving the model

All the above 3 set of equations are combined together to form a system of linear equations.
$Ax = B$, where A is the coefficient matrix and B is the vector of all the known quantities.
x is the vector of all the unknowns namely : a_{ijk}, u_k, q_i

4.1.5 Results from the model

Structure of typical A matrix (Coefficient Matrix) looks like: (white = positive, black=negative, grey = zero)

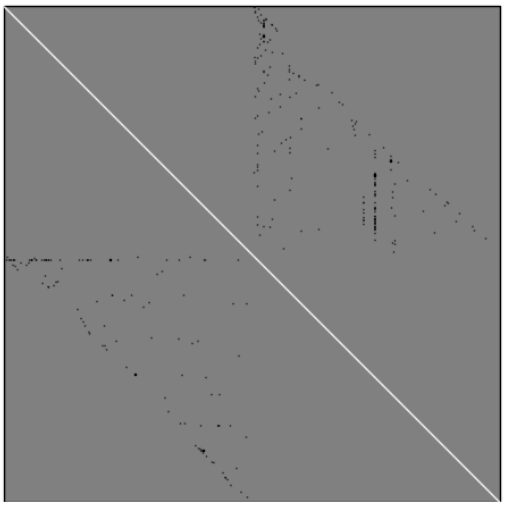

Figure 3: Coefficient matrix

Evaluation Metric:
We used Precision at K to evaluate our results against the ranking of users listed by stackoverflow.com. P@K is a metric which is used to measure the similarity between two ranks over the same set of users. It counts the number of user common in both the ranks in top K users and divide it by K.

$$P@K = \frac{R1 \cap R2}{K}$$

Precision at K plot, where K is number of Users being considered.
Figure 5 below shows histogram of User quality and Question quality. It can be seen that most of the user does not get good quality score, primarily because they don't have much

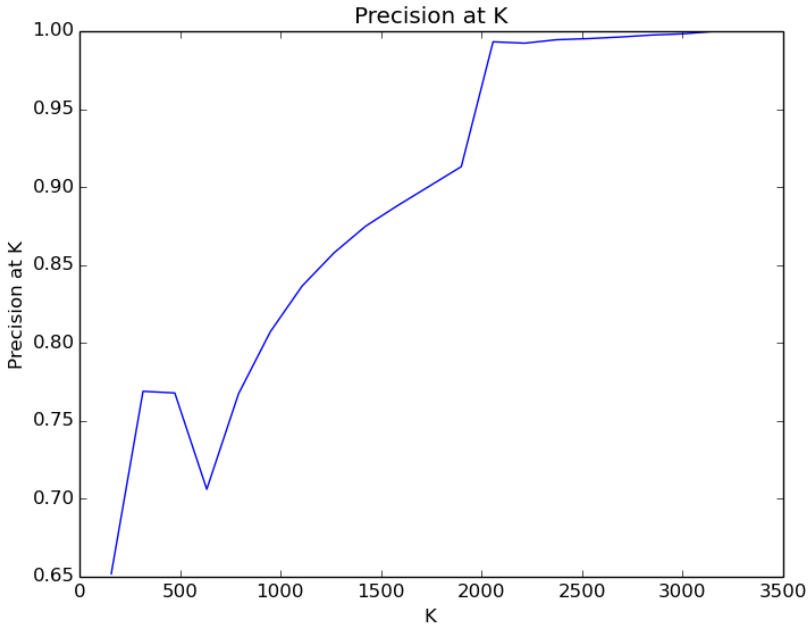

Figure 4: Precision at K

contribution. We want to filter such users out as we can't learn any meaningful insight about them from the data.

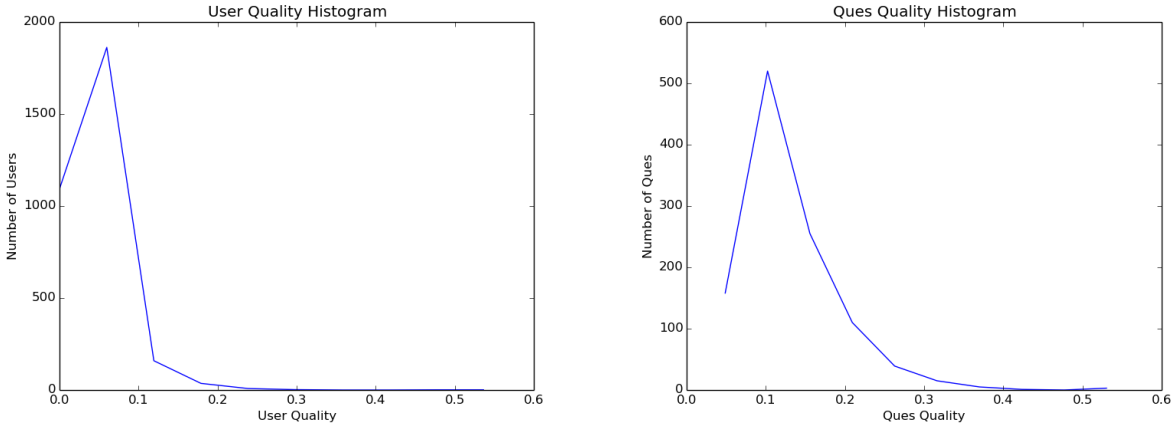

Figure 5: User & Question Quality Histogram

4.2 Probabilistic Graphical Model

4.2.1 Preprocess data

Pre-processed data files to structure them into useful data structures which can be used to train the model.

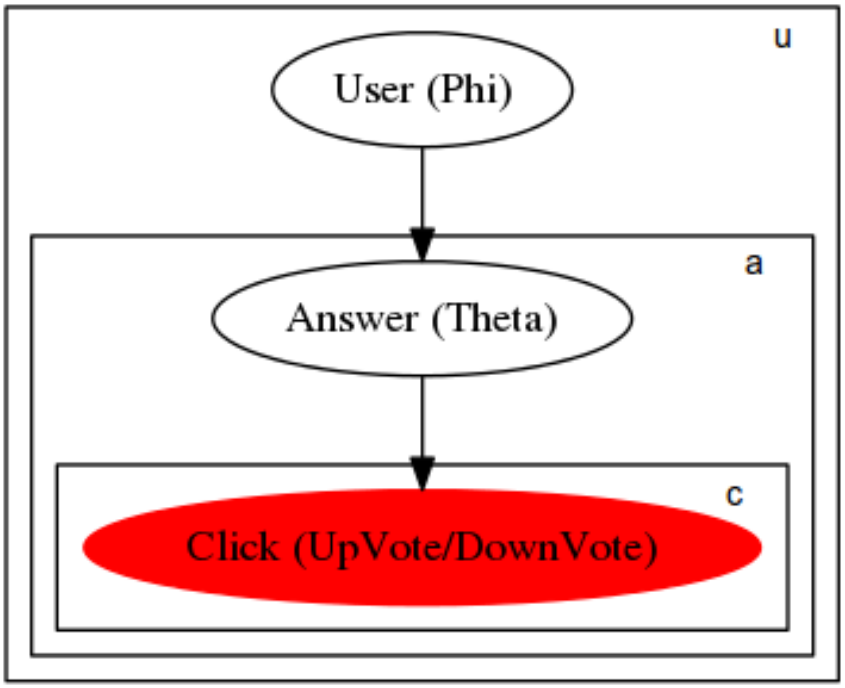

Figure 6: Graphical Model

One of the main data structure was a dictionary mapping each question to its click history. For example, {Q1 : list of click History}.
Further click History was structured as list of list
$[[vote1, [listofanswerspresent]], [vote2, [...]], ...]$
Note: Each click can either be an UpVote or DownVote.

4.2.2 Graphical Model

Figure 6 shows the graphical model used for modelling the relation between various random variables. Red node color corresponds to observed variable.

4.2.3 Conditional Probability Distribution (CPD)

Goal is to find the probability $P(\theta, \phi|clicks)$
And by Bayes rule,

$$P(\theta, \phi|clicks) \propto P(clicks|\theta, \phi) * P(\theta, \phi)$$
$$P(\theta, \phi|clicks) \propto P(clicks|\theta, \phi) * P(\theta|\phi) * P(\phi)$$

Now, we define the conditional probability by SoftMax function over all the answers choices available for a particular question when this click event occurred. This function also captures the information present in the time line of the clicks. Therefore, every click

is different and has different weight.

$$P(click = k|\theta_1, ..., \theta_n) = \frac{exp(\theta_k)}{exp(\theta_1) + ... + exp(\theta_n)}$$

and

$$P(\theta_i|\phi_j) \sim \mathcal{N}(\phi_j, \sigma^2)$$
$$P(\theta_i|\phi_j) = \frac{1}{\sqrt{2\pi\sigma^2}} exp(\frac{-|\theta_i - \phi_j|^2}{2\sigma^2})$$

and

$$P(\phi_j) \sim \mathcal{N}(0, \sigma^2)$$

4.2.4 Maximum Likelihood Estimation based Objective Function

The objective function is the product of the 3 terms defined above for every user(ϕ), answer(θ) and click(k).

$$\text{Objective Function} = P(clicks|\theta) * P(\theta|\phi) * P(\phi)$$
$$\text{Objective}_{MLE} = \underset{\theta,\phi}{\text{argmax}}\, NLL(P(clicks|\theta) * P(\theta|\phi) * P(\phi))$$

The goal is to minimize the negative log-likelihood (NLL) of this function and we used gradient descent based optimization methods.
L-BFGS : Limited-memory BFGS (L-BFGS or LM-BFGS) is an optimization algorithm in the family of quasi-Newton methods that approximates the BroydenFletcherGoldfarb-Shanno (BFGS) algorithm using a limited amount of computer memory. It is a popular algorithm for parameter estimation in machine learning.
AdaGrad : AdaGrad (for adaptive gradient algorithm) is a modified stochastic gradient descent with per-parameter learning rate, first published in 2011. Informally, this increases the learning rate for more sparse parameters and decreases the learning rate for less sparse ones. This strategy often improves convergence performance over standard stochastic gradient descent in settings where data is sparse and sparse parameters are more informative.

4.2.5 Model Verification

To verify the implementation of the model, we generated the data following the relation described by our model. We used that data to train our model. We compared the parameters learned by model with the actual parameters used to sample the data. Specifically, we compared the ϕ parameter learned by our model to the actual ϕ parameter used to create the data. We used **KendallTau** metric for this comparison.
KendallTau, is a statistic used to measure the ordinal association between two measured quantities. It is a measure of rank correlation: the similarity of the orderings of the data when ranked by each of the quantities. KendallTau is defined as:

$$\tau = \frac{(\text{number of concordant pairs}) - (\text{number of discordant pairs})}{n(n-1)/2}$$

where a pair of users is said to concordant if the relative ranking of both users is same in both the rankings, it is said discordant otherwise.
Properties The denominator is the total number of pair combinations, so the coefficient must be in the range $1 \leq \tau \leq 1$.

- If the agreement between the two rankings is perfect (i.e., the two rankings are the same) the coefficient has value 1.
- If the disagreement between the two rankings is perfect (i.e., one ranking is the reverse of the other) the coefficient has value 1.
- If the two rankings are independent, then we would expect the coefficient to be approximately zero.

Figure 7 shows the KendallTau plotted against Number of generated clicks. It can be seen that as the number of simulated clicks increases the correlation between the parameters learned by the model and actual ones also increases. This verifies that the implementation of the model is accurate.

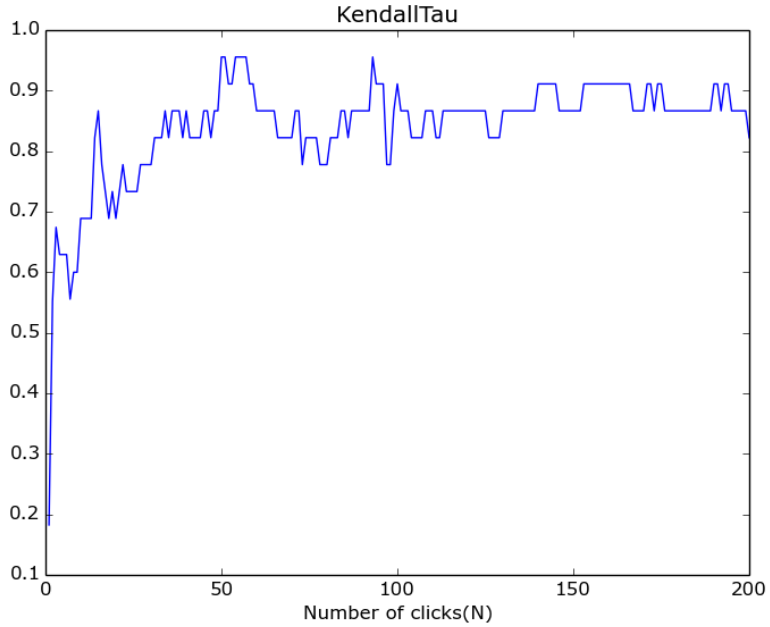

Figure 7: KendallTau vs Number of clicks

5 Moving to Big Data

Initially, we started with our experiments with smaller stack exchange websites like Vim and Datascience. In these websites the Number of Users were around 13,000 and number

of Posts were around 7,000 and the data set size was $\sim$ 15MB.

Once the model was tested on smaller websites, we moved to our final target i.e. bigger websites like Physics, Maths, Biology, and StackOverflow. For instance, Stackoverflow has

- Number of Users = 5277830 ($\sim$ 5 million), file size 1.5GB
- Number of Posts(Question + Answers) = 29499662 ($\sim$ 30 million), file size 45GB
- Number of Votes = 98928934 ($\sim$ 99 million), file size 9GB
- Total dataset size of $\sim$ 55GB

To process this huge dataset, we used approach similar to MapReduce algorithm. We broke each file into multiple smaller files and processed one at a time as processing the whole big file will require far more memory than available of a commodity computer. Figure 8 below visualise the algorithm.

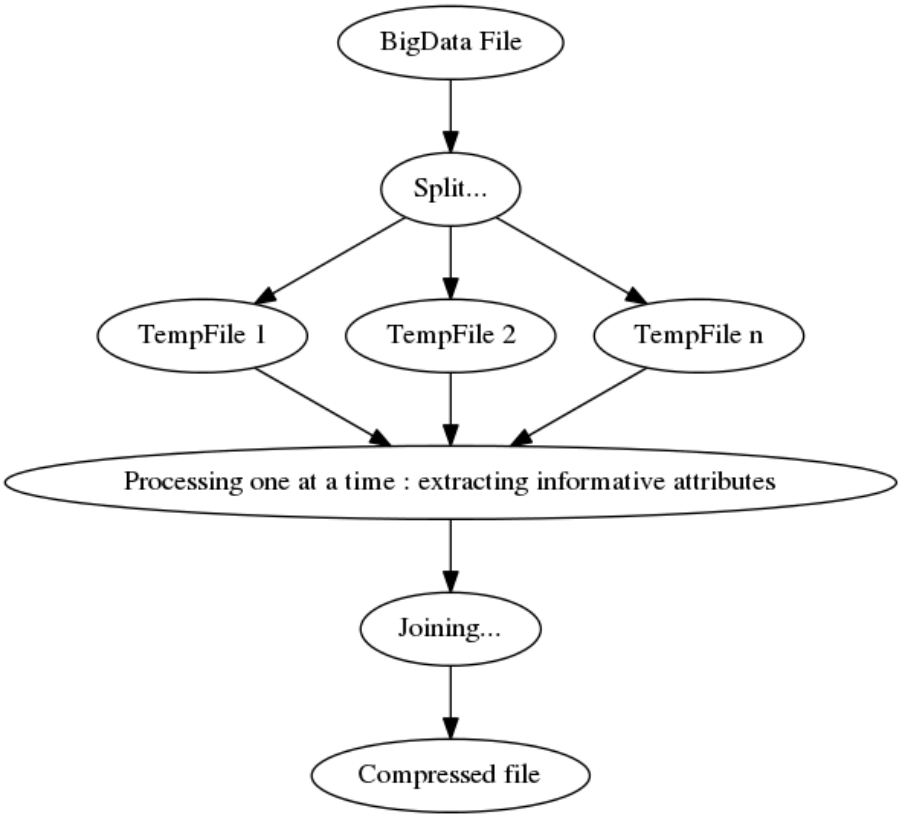

Figure 8: Simplified MapReduce Algorithm description

6 System design and Implementation

Figure 9 shows the entire system. The whole system is implemented in Python. The main function is implemented in `main.py`. Moreover, the whole system can be broken down into 4 separate module as described in the sections below.

```
usage: main.py [-h] [-a] [-c] [-f] [-p MINPOST] [-u MINUSER] [-s] -k KFOLD
               [-r]

Main script to run the full model. From compression to optimization. Need to
call from datasets/<xyz>/ directory

optional arguments:
  -h, --help            show this help message and exit
  -a, --all             Equivalent to -cfsr.
  -c, --compress        Flag to enable compression. Breaks file into
                        multiple to handle BigData files.
  -f, --filter          Flag to enable filtering of files. Flag -p and -u
                        also needed.
  -p MINPOST, --minpost MINPOST
                        Min. number of post for user to qualify(inclusive)
  -u MINUSER, --minuser MINUSER
                        Min. number of qualified user in a Q/A
                        pair(inclusive)
  -s, --structure       Flag to structure data in dataStructures.
  -k KFOLD, --kfold KFOLD
                        Number of folds to create and run. k=1 is special
                        case where it just trains on full data.
  -r, --run             Run the optimization and dump solution.
```

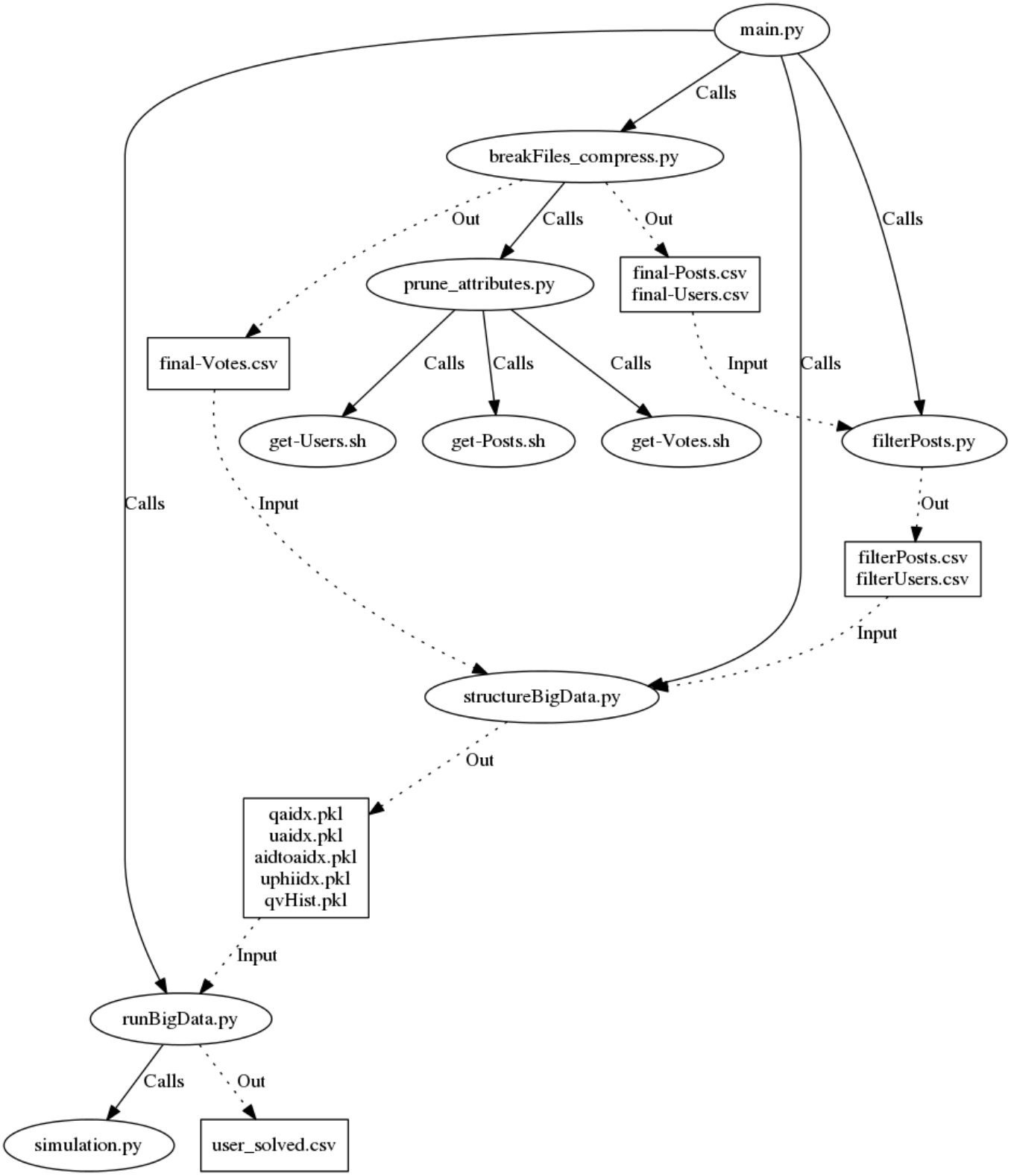

Figure 9: Flowchart: System design and Implementation

6.1 Break files and Compress

This is the first module of the overall system, in this we break the files into multiple small files and extract out the important attributes from each sub-file. This is essentially the implementation of simplified MapReduce algorithm as described above in Section 5. Implementation is in file `breakFiles_compress.py` and usage is as follows:

```
usage: breakFiles_compress.py [-h] [-u] [-p] [-v]

Script to break big data files into small and extract the important
attributes and put in to final-*.csv file. Need to call from datasets/<xyz>/
directory

optional arguments:
  -h, --help  show this help message and exit
  -u, --user  flag to break and process Users.xml file
  -p, --post  flag to break and process Posts.xml file
  -v, --vote  flag to break and process Votes.xml file
```

6.2 Filtration of compressed files

In this module we filter the compressed files based on two filtering criterion. First, we remove all the users with number of posts less than minimum number of post p. Next, we remove all the Q/A pairs in which the combined number of qualified users are less than minimum number of qualified user u. This module is implemented in file `filter_Posts.py`

```
usage: filterPosts.py [-h] -p MINPOST -u MINUSER

Script to filter Users and Posts based on minPosts and minUsers. Should be
called from datasets/<xyz> directory

optional arguments:
  -h, --help            show this help message and exit
  -p MINPOST, --minpost MINPOST
                        Minimum number of post for user to qualify(inclusive
                        )
  -u MINUSER, --minuser MINUSER
                        Minimum number of qualified user in a Q/A
                        pair(inclusive)
```

6.3 Forming Data structures

In this module, we take filtered files and structure them into useful dataStructures as described below. Implementation is in file `structureBigData.py`.

- qaidx: HashMap mapping Question id to its Answer (θ) index

- uaidx: HashMap mapping User id to its Answer (θ) index
- aidtoaidx: HashMap mapping Answer id to Answer (θ) index
- uphiidx: HashMap mapping User id to User (ϕ) index
- qvHist: HashMap mapping Question id to its click history

```
usage: structureBigData.py [-h] -k KFOLD

Script to structure data for running optimization. Should be called from
datasets/<xyz> directory

optional arguments:
  -h, --help            show this help message and exit
  -k KFOLD, --kfold KFOLD
                        Number of folds to create. k=1 is special case where
                        it just creates one training set of full data.
```

6.4 Training Model

In this module, we use the dataStructures formed in previous module to run the model and learn the hidden parameter i.e. θ and ϕ for every answer and user respectively. Implementation is in file `runBigData.py`.

```
usage: runBigData.py [-h] [-l] [-a] [-c] -k KFOLD

Script to load dataStructures and perform optimization using gradient
descent. Need to call from datasets/<xyz>/ directory

optional arguments:
  -h, --help            show this help message and exit
  -l, --lbfgs           Flag to use fmin_l_bfgs_b for optimization
  -a, --adagrad         Flag to use adagrad for optimization
  -c, --checkgrad       Flag to use check gradient
  -k KFOLD, --kfold KFOLD
                        Number of folds to train on. k=1 is special case
                        where it just trains on one training set made up
                        of full data.
```

6.5 Measuring model accuracy

This module use the trained model to predict the clicks and does K-fold cross validation to report a cross validation accuracy of the model on the dataset. Implementation is in file `modelAccuracy.py`.

```
usage: modelAccuracy.py [-h] -k KFOLD

Script for testing/making prediction on validation set. Should be called
from datasets/<xyz> directory

optional arguments:
  -h, --help            show this help message and exit
  -k KFOLD, --kfold KFOLD
                        Number of folds to run prediction. k=1 is not valid.
```

7 Visualization and Results

7.1 TimeLine of typical question in Real Data

- Figure 10 shows the entire vote and answer generation history of a particular question capturing the number of answers & in order they were generated
- Each dot captures a click event (up-vote or down-vote)

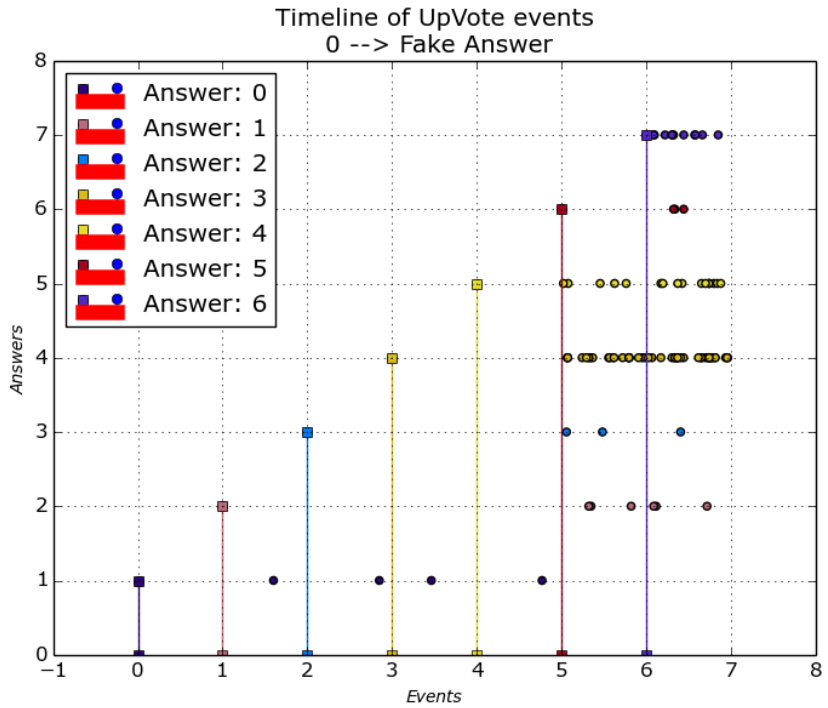

Figure 10: Timeline of typical question

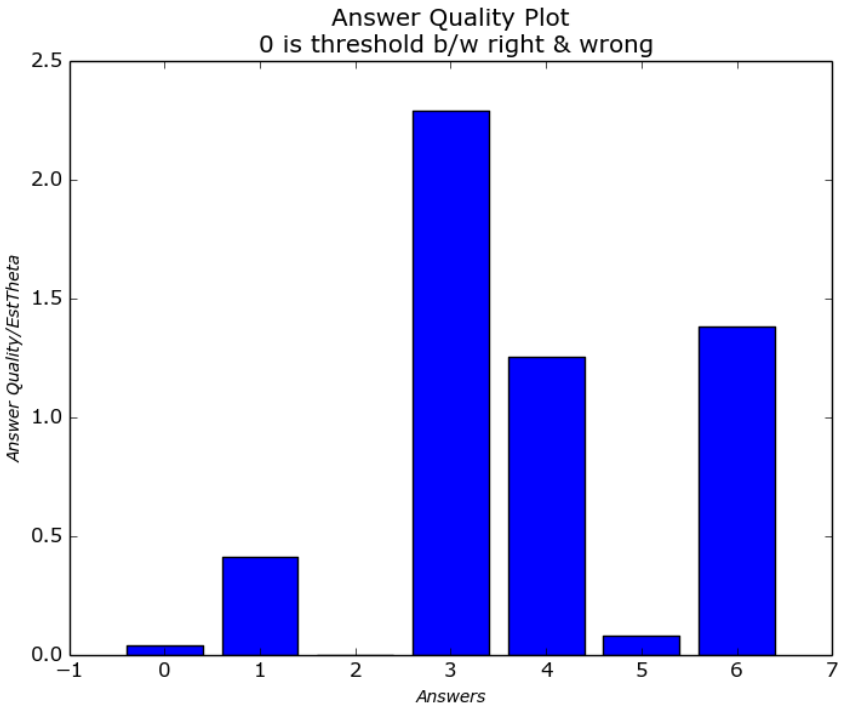

Figure 11: Quality of every answer in above Timeline

Figure 11 shows the learned quality corresponding to every answer in Figure 10.

7.2 Results: K-fold cross validation accuracy

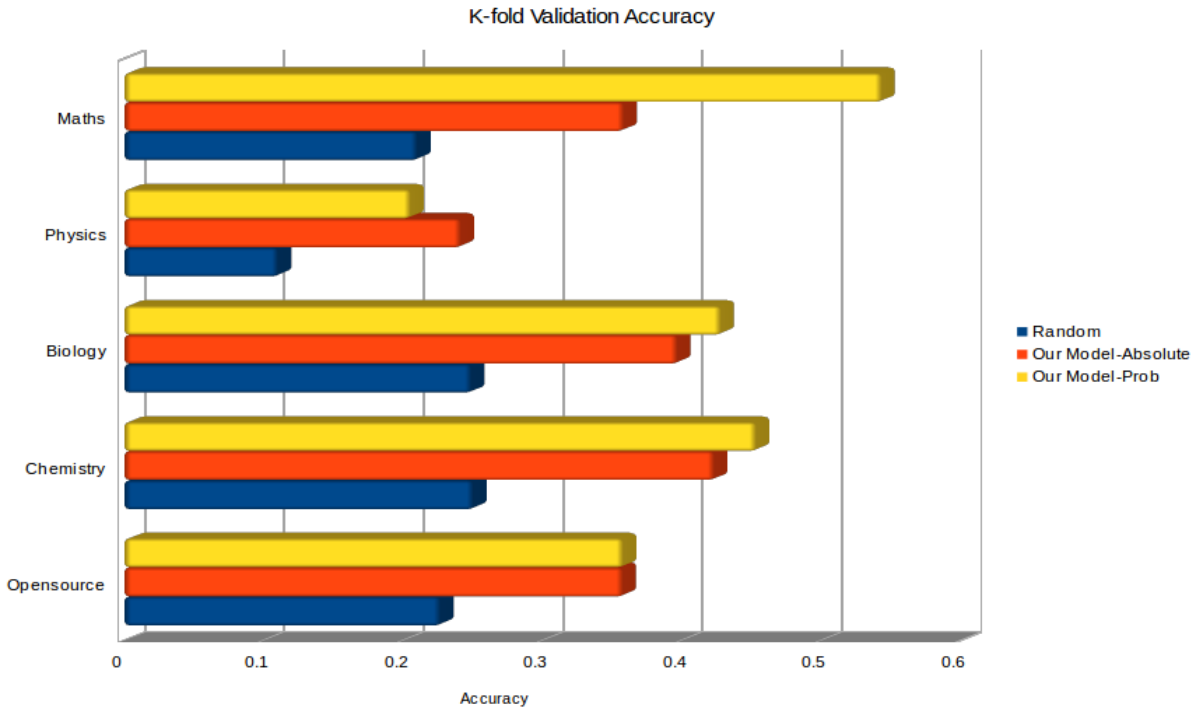

Figure 12: K-fold cross validation Accuracy on Real Datasets

Figure 12 above shows the K-fold cross validation accuracy results from our model. Below is list of few important points and takeaways from this result.

- We use K=10. We predict the clicks (UpVote/DownVote) made by Users and average over each fold.
- Note: Its a multiclass problem with varying number of classes.
- Evaluation Metric 1 : Absolute - $click = argmax(\phi_j; j \in X)$, X is list of answers for this question. This means we just sample a click for answer with highest θ parameter among all the answers for a particular question.
- Evaluation Metric 2 : Probabilistic - $click \sim P(\frac{exp(\phi_j)}{exp(\phi_1)+...+exp(\phi_n)})$. To calculate this metric we sample a click from the probability density function over all the θ parameters for a particular question.

8 Conclusion and Further Work

8.1 Conclusion

The proposed system in this report provides an automated tool that can generate an assessment which a user can use to test his/her skills. Currently available methods for generating a test requires an excessive amount of human (generally a teacher or domain expert) effort. Also, the traditional approach to assessment generation does not scale to test in multiple fields or even personalized tests. Our approach mitigate all these issues and provides a more scalable solution. The proposed solution uses machine learning to automatically generate assessment using already available million of questions and answers from websites like Stackoverflow.com. The main challenge was to learn interesting hidden variables like difficulty of question, quality of answers, and ability of users. The planning and implementation of the project went well and we were able to learn these hidden variables successfully. We were able to train the model and do predictions from the learned model with an accuracy of order $\sim$50% which was 20% better than random guessing. This accuracy is quite good as this problem is a varying multi-class classification problem.

8.2 Further Work

Currently we only take into account the user's ability and answers quality, we want to extend our graphical model to model the difficulty of question as well. The intuition we want to encode for question difficulty is that a user with ability parameter ϕ generates with high probability a question whose difficulty is higher than user's ability. Also, a user successfully answering a particular question has better ability than the question difficulty parameter.

We also plan to model the learning curve of user using the samilar model over different time periods monitoring how the understanding of user of a particular topic evolved over time.

9 Python Code

The full implementation code is available @ GitHub