Relevance Based Ranking of Answers in Community Question Answering Forums

Komal Talwar (1710110183), Manika Mittal (1710110202), Shivangi Goel (1710110309)

***ABSTRACT*- The internet is a rich resource for exchange of knowledge. Community question answering sites such as Yahoo!Answers, Quora and StackOverflow are flowing with questions, answers and a thread of comments. Yet, we often find ourselves trying to decide which answer correctly satisfies the question. Ranking of answers on these platforms is usually performed by users, by upvoting an answer that they prefer. However, this ranking seems to overlook the possibility that a newly added answer is of high importance. Along with that, there is a high possibility of upvoting being biased or for a user to vote incorrectly. Our paper overcomes these issues by using Linear Regression over Heuristic values and Topic Similarity Score calculated from a Topic Modelling algorithm, to produce correct ranking of answers, that keeps getting dynamically updated as new answers are added.**

I. Introduction

With the massive growth of online social media applications, community question answering platforms have a booming popularity among information seekers. These platforms, such as Quora and StackOverflow, allow a user to ask or answer just about any question that comes to mind. It is a type of information retrieval system which is fundamentally based on perspectives of knowledge sharing and collaborative learning. Knowledge is shared among members of a particular community and the results obtained ensures the satisfaction of the people, which is a specific way of informal learning. Since these platforms are open-ended and open-domain, there may be high variance of question and answer quality. These platforms have answers ranked on the basis of upvotes rather than relevance which may make a user think that newer answers at the bottom which have not received votes yet are of no relevance. Ranking of answers not only helps the user find relevant information in less time, but also allows quality of answers to be maintained in threads.

Many of the platforms available online for CQA require human interference for ranking of answers and may most often satisfy the information needs. But information need may differ among users and so a definitive relevance between question and answer is to be defined.

Several approaches have been tested out for ranking of answers on these platforms. Mainly, classification is done on answers to tag them as good, potentially useful or bad. The approaches differ among themselves on the kind of features extracted and put to use, classification models used for training, the type of score used for measuring and scoring answers, etc.

Forums contain a wealth of knowledge related to discussions and solutions to various problems and needs posed by various users. Therefore, mining such content is desirable as otherwise it is a painstaking process for users to manually search through many posts in various threads. This is true especially for long threads containing hundreds or even thousands of posts. Our framework applies topic modelling and some heuristics to sort the answers to a question according to relevance.

The following heuristics are taken into consideration:

1. Presence of links in answer
2. Presence of pictures in answer

These heuristics are used along with the topic modelling score and regressed upon to find the correct weightage.

In this paper, we build our model based on the above approach and then compare it with a reference Model, WordCount and check to see which performs better.

The remainder of this paper is organised as follows: Section II presents related work to our research, Section III presents proposed approach and implementation, Section IV presents a comparison between the two models, Section v contains results and our paper is finally concluded in Section VI.

II. RELATED WORK

With the aim of ranking answers, the authors of [1] have designed a **semantic search engine framework** to process software threads and recover relevant answers according to user queries. The framework infers semantic tags of posts in the CQA forum threads and utilizes these tags to recover relevant answer posts. A forum thread usually consists of an initiating post and a number of reply posts which could be answers to the question, new questions, clarifying information or some kind of feedback or comments. This structure of a discussion board is utilized to extract relevant information out of the abundant data available. For this, the framework consists of two main steps:

- An engine is proposed that infers a comprehensive set of tags of posts in forums: questions, answers, clarifying questions, clarifying answers, positive feedback, negative feedback and junk. In this step, once the dataset is collected and pre-processed, the posts for each forum are manually annotated so that we have sufficient amount of labelled/ tagged training data. Next, they extract features from these examples which simply include individual words from each piece of text to be classified and author information. Once the training posts are transformed into feature vectors based on the chosen features, a classification algorithm is used to train the classifier.
- In the next step, a semantic search engine is proposed to find relevant answers to queries by leveraging automatically inferred tags. The tag inference engine is embedded into the search engine framework for this purpose. An additional component called the filterer picks the top-k matching documents from the retrieval tool based on their inferred tags. It keeps the relevant solution posts while filters out the irrelevant posts, e.g., junk, feedback, etc. The tags could be used to extract answers which have been given positive feedback. Answers with only negative feedback can be pruned.

In [2], the authors have proposed an answer ranking framework based on type-matching scores from Named Entity Types and Semantic answer types of answer candidates. A Knowledge Base (KB) is usually a directed graph of multiple triples (relational tuples), which are composed of two entities (concepts) and their relation (property). For example, to express the knowledge, “Microsoft was founded by Bill Gates”, two entities (“Microsoft” and “Bill Gates”) and their relation (“was founded by”) are extracted and knowledge expressed like this is called a triple. **Named Entity Recognition** (NER) is used to find any domain-independent entities in the text. Named Entity Linking (NEL) based on SVM algorithm is used to link different surface forms into unique Universal Resource Identifiers (URI). The NEL classifies each named entity into one of 144 named entity types (NETs). The next step is to use the Distant supervision algorithm to find the relation between two entities. Using the aforementioned method, the knowledge bases as well as the questions are converted into a connected graph of triples. The focus which is what the answer should replace was extracted from a question using syntactic rules. Later, answer candidates for answer ranking replaced the focus. The top hundred documents sharing the highest number of named entities with query triples were selected as document candidates. Of the hundred document candidates, all named entities were considered potential answer candidates. This is followed by calculation of triple-matching score between the knowledge-triples for the answer candidates. To further improve accuracy, the approach uses a Semantic Answer Type (SAT) tool trained with heuristic rules and structural SVM. The answer candidates with the highest triple matching score and which have the highest number of shared types with the question SATs and NETs are ranked the highest.

The approach of [3] classifies answer threads as good, potential, bad and useless. To do the same, the author has defined two hierarchies of features, baseline features and thread-level features. In baseline features, lexical and syntactic similarity between the question and answers is measured. N-gram similarity is computed using greedy string tiling, Jaccard coefficient, etc. To check whether an answer is good or not, a set of heuristic features is designed. These include whether the answer contains URLs or not, contains words such as “yes”, “sure”, “no”, “can”, “neither”, “okay”, and “sorry”, as well as symbols ‘?’ and ‘@; starts with “yes; has been posted by the same user.

First each answer is compared on the basis of this. Further, thread-level features are used. These are classified on the basis of whether an acknowledgement is contained in the comments or not; whether a question is contained in the comments. All these features are mapped to a score on the basis of which the comparison is done. Classification is done using SVM and linear regression is also performed. The author, using his techniques, has observed that adding the thread-level features improves the score by two F1 scores.

The author’s approach in [4] required him to work on a set of features extracted such as word length, word number, part-of-speech, name entity, content tag, class tag, etc., to classify answers on the basis of good potential and bad dialogue. Two-level hierarchical classification was brought into play. Answers were classified as good potential and bad dialogue in the first level; classifying good potential answers as good and potential, classifying bad dialogue answers as bad and dialog separately in the second level. Ensemble learning which is strategically generating the classification models and combining them to improve the prediction, is performed next. It includes training and choosing top N best classifiers based on cross validation on training data, then using the N classifiers to vote for the final result. SVM classifier was used with a toolkit LIBLINEAR. It was found out that hierarchical classification actually proves out to be a better way to rank answers than other ways of ranking tried by the author in the paper

In [5], the author has ranked the answers on the basis of the answer- related Question Answer (QA) features and internal non-QA features. The QA features included a total count of answers and the best answer rate, where the best answer rate was based on the upvoting and downvoting of the answers done by the user (asker). There was a bias in asker’s best answer selection that it did not ensure the best relevant answer to be ranked the highest. The non-QA features included picturization (presence of pictures in the answers), self-description, answerer’s age and links or some external platform references. The self-description feature satisfied the best selection of answers by taking explicit feedback on a 5-point scale by the viewer but it did not take into account the user’s topical authority, expertise, etc. while evaluating the data obtained from the feedback. But they finally assumed that the scores provided by the asker or the community which although was more objective would be considered. The machine learning models used by the author for the best answer selection and ranking were L2R(Learning to Rank) which is an advanced version of a simple Regression model, SVM Rank (using features pairwise), ListNet, Random Forest and Naïve Bayes. Naïve Bayes was used only for the best answer rate and self-description features in which user involvement was required.

III. PROPOSED APPROACH AND ITS IMPLEMENTATION

a. Method

Topic modelling is an unsupervised machine learning technique that’s capable of scanning a set of documents, detecting word and phrase patterns within them, and automatically clustering word groups and similar expressions that best characterize a set of documents. This is known as

'unsupervised' machine learning because it doesn't require a predefined list of tags or training data that's been previously classified by humans. Topic modelling involves counting words **and grouping similar word patterns** to infer topics within unstructured data. The underlying assumptions are that every document comprises a statistical mixture of topics, i.e. a statistical distribution of topics that can be obtained by "adding up" all of the distributions for all the topics covered and the distributional hypothesis which states that similar topics make use of similar words. What topic modelling methods do is try to figure out which topics are present in the documents of the corpus and how strong that presence is. The most common topic modelling techniques **are Latent Dirichlet Allocation and Latent Semantic Analysis**. The main difference between LSA and LDA is that **LDA** assumes that the distribution of topics in a document and the distribution of words in topics are Dirichlet distributions. LSA does not assume any distribution This is the reason why in most cases, LDA is found to provide more accurate results than LSA and that is why we have used LDA in our model to perform the topic modelling task.
Latent Dirichlet Allocation (LDA), as explained in [6], is an unsupervised, statistical approach to document modelling that discovers latent semantic topics in a large collection of text documents. LDA ignores syntactic information and treats documents as bags of words. It also assumes that all words in the document can be assigned a probability of belonging to a topic. That said, the goal of LDA is to determine the mixture of topics that a document contains. Each document is represented as a distribution over a fixed number of topics, while each topic is represented as a distribution over words. **Figure 1 and 2** explain the working of LDA.

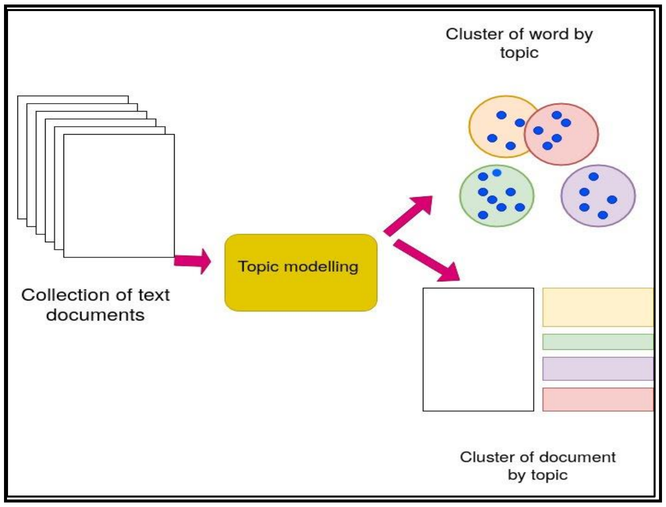
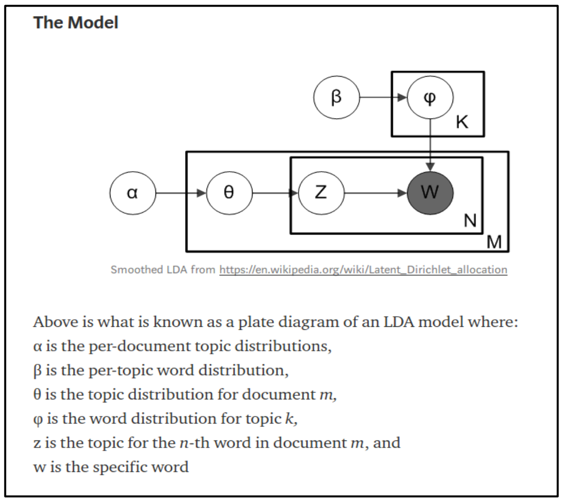


Figure 1: Working of a basic LDA model


The Model
β
φ
K
α
θ
Z
W
N
M
Smoothed LDA from https://en.wikipedia.org/wiki/Latent_Dirichlet_allocation

Above is what is known as a plate diagram of an LDA model where:
α is the per-document topic distributions,
β is the per-topic word distribution,
θ is the topic distribution for document *m*,
φ is the word distribution for topic *k*,
z is the topic for the *n*-th word in document *m*, and
w is the specific word

Figure 2: Plate notation for LDA with Dirichlet distributed topic word distribution

In the presented work, these two probability distributions are used to calculate the similarity between a question and all possible answers. We have constructed an LDA topic model using the content and subject of all questions that currently exist in a database. Therefore, topic distributions over all questions and word distributions over all topics are known. This model is then used to infer the distribution over topics of all the answers of a given question to find which answers are more similar to it. Apart from topic-based similarity, studies have shown that certain **heuristics** like **presence of URLs** and pictorial representations are indications of a good answer. Therefore, we have calculated these Boolean heuristic values (presence of URLs (Yes/No) and presence of images

(Yes/No) for all our answers. Now, that we have a **thematic similarity score** and heuristic values for all answers, we apply linear regression, using these three features, with the rank of the answers as the predicted value. For training, we have manually ranked the answers based on their relevance. Figure 2 provides a conceptual overview of our approach.

The answer which is the most thematically similar to the question and satisfies the heuristics is then proposed to be the best answer and the other answers are also ranked in the descending order of their predicted ranks. In this way, the model provides us with an ordering of all answers of a particular question, in the decreasing order of their importance, thereby saving the time and effort of any user, looking up the answer to the query as now he/she doesn't need to read up all answers to find all answers, to find the most relevant one. In most cases, the most suitable answers will be among the top 3-4 answers.

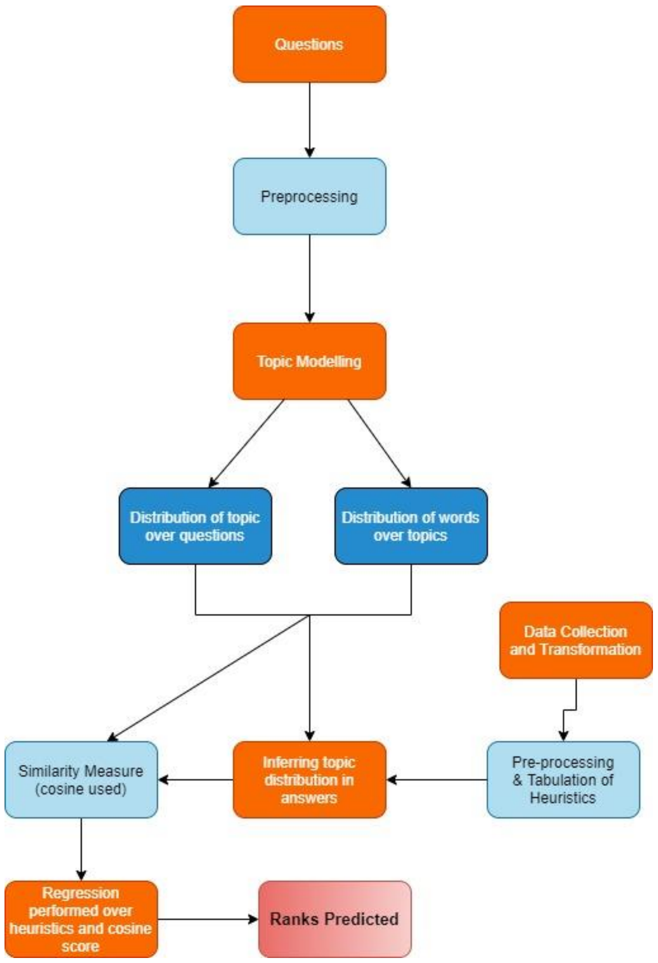

Questions
Preprocessing
Topic Modelling
Distribution of topic over questions
Distribution of words over topics
Data Collection and Transformation
Similarity Measure (cosine used)
Inferring topic distribution in answers
Pre-processing & Tabulation of Heuristics
Regression performed over heuristics and cosine score
Ranks Predicted

Figure 3: Conceptual overview of our ranking model

b. Topic Modelling

LDA's approach to topic modelling is it considers each document as a collection of topics in a certain proportion. And each topic as a collection of keywords, again, in a certain proportion. Once we provide the algorithm with the number of topics, all it does it to rearrange the topics distribution within the documents and keywords distribution within the topics to obtain a good composition of topic-keywords distribution. A topic is nothing but a collection of dominant keywords that are typical representatives.

The following are **key factors** to obtaining good segregation topics:

1. *The quality of text processing.*
2. *The variety of topics the text talks about.*
3. *The choice of topic modelling algorithm.*
4. *The number of topics fed to the algorithm.*
5. *The algorithms tuning parameters*.

The two main inputs to the LDA topic model are the dictionary and the corpus. Gensim package creates a unique id for each word in the document. The produced corpus shown is a mapping of (word_id, word_frequency). For example, (0, 1) above implies, word id 0 occurs once in the first document. Likewise, word id 1 occurs twice and so on. Now, we finally train the LDA model by providing the dictionary, corpus and number of topics. We found the optimal number of topics for our dataset by computing the coherence score of LDA models built over a varied number of topics. From the graph in Figure 4, **we found that the suitable number of topics for our model is 8. Model perplexity and topic coherence provide a convenient measure to judge how good a given topic model is.** Higher the coherence, and lower the perplexity, better is the model. For most purposes, coherence is found to be a more useful measure. **Our model recorded a coherence score of 0.51,** which is an indication of a good topic model.

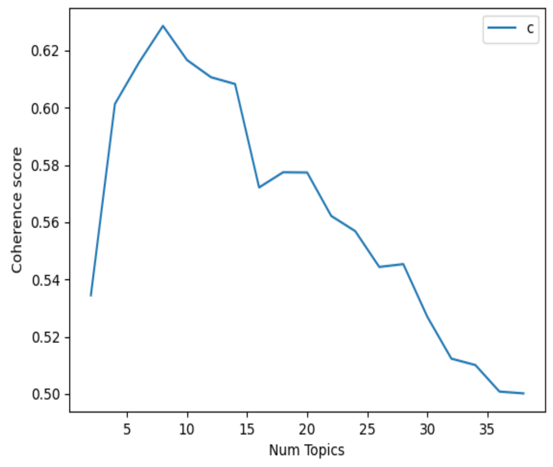


Figure 4. Coherence score vs Number of topics

Once the LDA model was built, we examined the produced topics and the associated keywords using pyLDAvis package's interactive chart tool. Figure 5 shows that each bubble on the plot represents a topic. The larger the bubble, the more prevalent is that topic. A good topic model has **fairly big, non-overlapping bubbles** scattered throughout the chart instead of being clustered in one quadrant. The figure shows that our topic model is decently constructed.

The above LDA model is built with 8 different topics where each topic is a combination of keywords and each keyword contributes a certain weightage to the topic.

We can see the keywords for each topic and the weightage(importance) of each keyword in Figure 7.

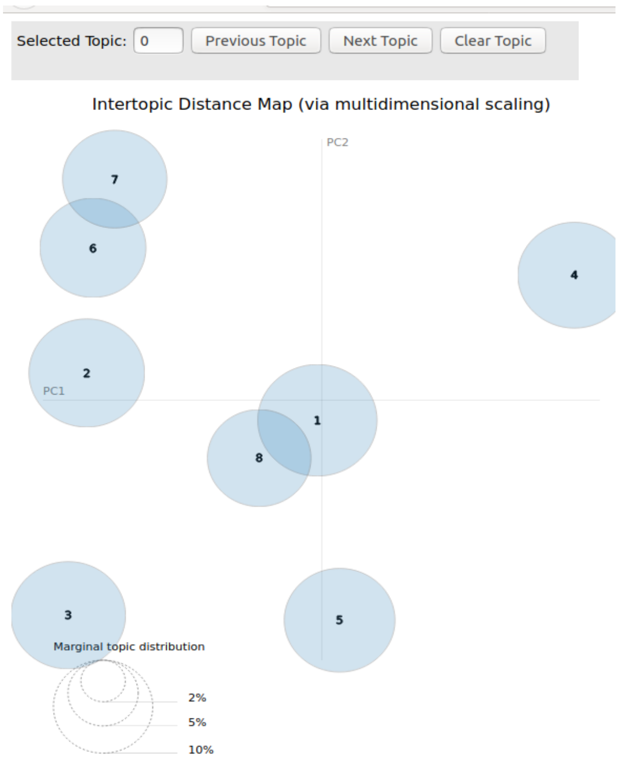

Selected Topic: 0
Previous Topic
Next Topic
Clear Topic
Intertopic Distance Map (via multidimensional scaling)
PC2
PC1
7
6
4
2
1
8
3
5
Marginal topic distribution
2%
5%
10%

Figure 5: Inter-topic distance map

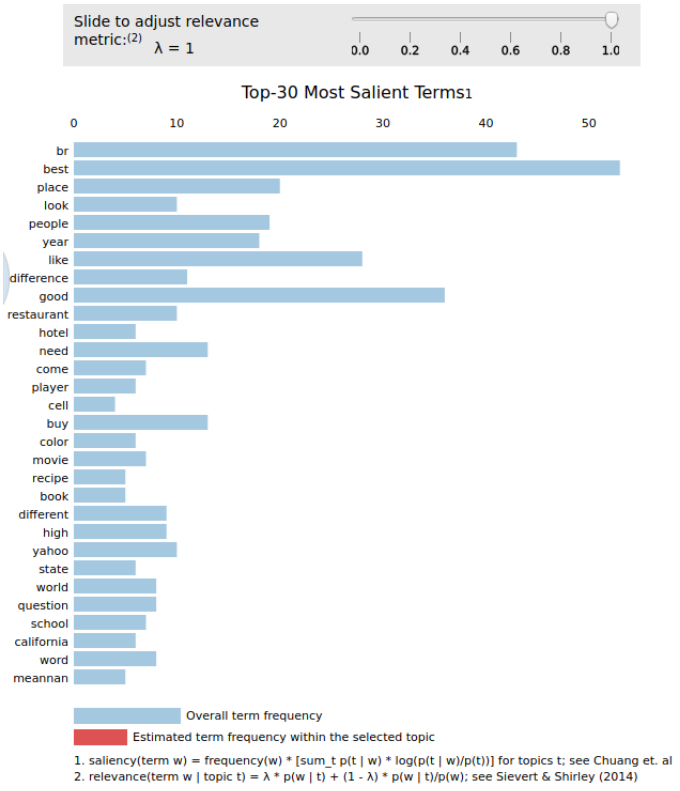
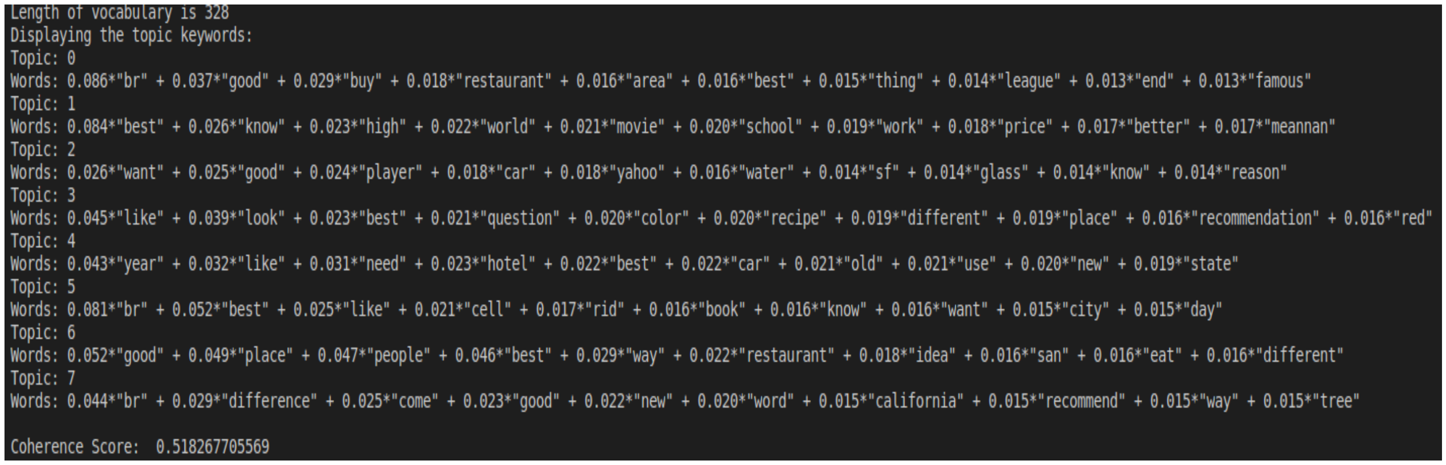


Figure 6: Top 30 most salient terms per topic



```
Length of vocabulary is 328
Displaying the topic keywords:
Topic: 0
Words: 0.086*"br" + 0.037*"good" + 0.029*"buy" + 0.018*"restaurant" + 0.016*"area" + 0.016*"best" + 0.015*"thing" + 0.014*"league" + 0.013*"end" + 0.013*"famous"
Topic: 1
Words: 0.084*"best" + 0.026*"know" + 0.023*"high" + 0.022*"world" + 0.021*"movie" + 0.020*"school" + 0.019*"work" + 0.018*"price" + 0.017*"better" + 0.017*"meannan"
Topic: 2
Words: 0.026*"want" + 0.025*"good" + 0.024*"player" + 0.018*"car" + 0.018*"yahoo" + 0.016*"water" + 0.014*"sf" + 0.014*"glass" + 0.014*"know" + 0.014*"reason"
Topic: 3
Words: 0.045*"like" + 0.039*"look" + 0.023*"best" + 0.021*"question" + 0.020*"color" + 0.020*"recipe" + 0.019*"different" + 0.019*"place" + 0.016*"recommendation" + 0.016*"red"
Topic: 4
Words: 0.043*"year" + 0.032*"like" + 0.031*"need" + 0.023*"hotel" + 0.022*"best" + 0.022*"car" + 0.021*"old" + 0.021*"use" + 0.020*"new" + 0.019*"state"
Topic: 5
Words: 0.081*"br" + 0.052*"best" + 0.025*"like" + 0.021*"cell" + 0.017*"rid" + 0.016*"book" + 0.016*"know" + 0.016*"want" + 0.015*"city" + 0.015*"day"
Topic: 6
Words: 0.052*"good" + 0.049*"place" + 0.047*"people" + 0.046*"best" + 0.029*"way" + 0.022*"restaurant" + 0.018*"idea" + 0.016*"san" + 0.016*"eat" + 0.016*"different"
Topic: 7
Words: 0.044*"br" + 0.029*"difference" + 0.025*"come" + 0.023*"good" + 0.022*"new" + 0.020*"word" + 0.015*"california" + 0.015*"recommend" + 0.015*"way" + 0.015*"tree"

Coherence Score:  0.518267705569
```

Figure 7: Keywords for each topic and their respective weightage(importance) in our LDA model

Following this, we found the topic distributions across all questions, and answers, to construct a vector of length 8 (number of topics per document) where each value represents the probability of presence of a topic in that document. Once the topic vectors are ready, we use them to find a similarity score between a question and its answers.

c. Similarity Measure

The output of LDA is a multinomial distribution over topics for each document in the question database and a multinomial distribution over words for each topic. The algorithm allows for the inference of topic distributions for unseen documents. In our experimental setup, answers are the unseen documents for which a topic distribution is determined using LDA, according to the topic model learnt using the database of questions. A similarity measure is therefore used to infer similarity between topic distribution for each answer and its question.
There are various similarity measures available for achieving this, but the one which we have used is **cosine score**, as it is an efficient and easy to comprehend approach.

The cosine similarity measures the cosine of the angle between two vectors. *As the measure tends towards 1, two vectors are more similar*. The cosine similarity between two vectors a and b is measured as follows:

In topic modelling, the distribution over topics can be represented as a vector. The first coordinate of that vector is the probability of the first topic in the document, the second coordinate is the probability of the second topic in the same document and so on. Since the distribution over topics for each document is known, **we measure the similarity of two documents** using the aforementioned formula**. A higher cosine score signifies a better answer.**

$$cos(a,b) = \frac{\vec{a}.\vec{b}}{|\vec{a}|.|\vec{b}|}$$

d. Linear Regression

Once we have the cosine scores of all answers for a question, we apply linear regression. To train our model, we took cosine score (topic similarity score) between the question and each answer and presence of links as our heuristics since our dataset answers were devoid of pictures, we ignored the heuristic which gave importance to presence of pictures in answers. Since we had two predictors, we used **multiple linear regression** with rank of answers as our response variable.

$$y = b0 + b1 * x1 + b2 * x2$$

The equation is as above, where y is the response variable, x1 and x2 are predictors and b0, b1 and b2 are coefficients. We split the dataset in the ratio 20:80, where 20x is our testing data and 80x is our training data. For our model, the values of b1 and b2 came out to be negative which implies that if number of links are increased, then the rank decreases and similarly, if the cosine value increases, the rank decreases and we have taken the notation of the rank such that lower rank implies higher priority and vice versa, which satisfies our heuristic that more similarity between the answer and the question, higher is the rank (lower is the value of rank). The values b1 and b2 obtained are the weights of the cosine score and presence of links, which gives the weightage each of these predictors in predicting the relevance of the answer to the question. Using this, we obtain an **ordering of the answers** of every question and compare it with the ordering done by us manually to measure the correctness of our model. **Using the equation of the regression line obtained, we then predict the relevance of the answers in the testing set, to obtain the correctness of our model on an unseen dataset**.

e. Experimental Data

Experiments are conducted using question-answer pairs taken from a publicly available real-world online collaborative question answering platforms: **Yahoo!Answers**. This portal enables users to collaborate in the form of asking question and proposing answers. They use a collaborative voting mechanism, which allows members of the community to upvote or downvote the questions and answers that they think are relevant or not. Furthermore, a person who asks a question can mark one of the answers as the best answer. Each question has multiple answers associated with it. We choose the best answer (marked by the person who asked the question) as the rank 1 answer, resulting in one answer for each question (a question-answer pair). We use the best answer selected by the asker of the question and compare it with the rank 1 answer obtained through our approach to obtain the correctness of our model.

(i) Data Gathering

Like any machine learning project, we began our project with the task of gathering data. This step is very important because the quality and quantity of data that you gather will directly determine how good your predictive model can be. In this case, we decided to use a dataset containing a set of questions and corresponding answers from Yahoo! Answers, version 1.0.
Yahoo! Answers is a website where people post questions and answers, all of which are public to any web user willing to browse or download them. The data we have collected is the Yahoo! Answers corpus as of 10/25/2007. It includes all the questions and their corresponding answers. The corpus contains 4483032 questions and their answers. In addition to question and answer text, the corpus contains a small amount of metadata, i.e., which answer was selected as the best answer, and the category and subcategory that was assigned to this question. Along with this, the data also contains other fields such as user ID, date, last answer's date, etc., which are not relevant for our purpose.

(ii) Data Preparation

Next, the dataset obtained was in XML format, so we wrote a Python code to convert it into CSV format so that it becomes suitable for various manipulations needed to be performed on the text. This task is crucial as we need our dataset in a particular format. The XML file which we had obtained had several nested tags, and so this process was time-consuming, but still only has to be done once. Among the nested tags, there were several fields that we did not require such as user ID, Date, timestamp, etc. These fields were ignored while writing to the CSV file.
The following fields were finally included in our initial database:

Subject
Content
Best Answer
Ans0-Ans9

(ii) Pre-processing

The dataset we obtained from an online source primarily contains texts so the steps followed for text pre-processing are tokenizing the dataset by breaking down the texts (questions and answers primarily) into small sentences and finally into single words known as tokens, converting all letters to either upper or lowercase, removing stop words like punctuation, accent marks, removing the occurence of all non-alphanumeric characters like special symbols, etc., common occurring words like is, a, the, to, etc. and other diacritic, removing white spaces. We then Lemmatize the text data by removing the inflections and mapping the words to its root form. We perform all these steps so that we can focus and perform our machine learning model on important words instead. We then **normalize** the data by converting it to a canonical/standard form. Our original text data also contained some html and picture tags which we used as heuristics in our model. Presence was denoted as 1 for yes and 0 for no. Since our database had very less number of pictures, it did not play a huge role in the outcome.

f. Reference Model

As a counter example to check how good our model is, we chose to apply the **WordCount** approach which calculates the **TF-IDF score** and uses it as a similarity measure to check how similar an answer is to a question. For this model, we fed it pre-processed questions and answers, calculated TF-IDF values as a vector for each question and answer and then ranked the answer on the basis of descending order of their cosine similarity to the question associated with them.

$$w_{i,j} = tf_{i,j} \times \log\left(\frac{N}{df_i}\right)$$

$tf_{i,j}$ = number of occurrences of i in j
df_i = number of documents containing i
N = total number of documents

IV. COMPARISON

WordCount Model	**Topic Modelling LDA Model**
Lexical similarity is checked, it just checks the presence of words.	Similarity is checked on the basis of topics created by the LDA model.
No semantic similarity checked.	Similarity checked through semantic similarity
Accuracy is around 31%	Accuracy is around 61%.

V. RESULT AND DISCUSSION

Two models have been implemented in this research paper, **Topic Modelling with Linear Regression and WordCount model**.

Correctness has been calculated based on the number of times the acclaimed ‘best answer’ is ranked first. We observe that by **assigning weights** from regression to the two factors mainly considered, cosine values found after topic modelling, and the **presence of heuristics**, we make a model which is more accurate.

WordCount model, however simple it may look, only takes into account the exact matching of words between question and answers, so it completely neglects the fact that an answer may not have any of the words that are present in the question.

Our results reflect this fact.

An accuracy of **61%** has been achieved by our model. This means that in 700+ threads, our best answer ranked first according to our model.

In comparison, WordCount had an accuracy of **31%**.

All of the results clearly state that our approach for ranking of answers **performed better** than the reference model.

VI. CONCLUSION AND LIMITATIONS

In the world of Community Question Answering, we may never be sure of an answer being correct as it is not always deemed or verified by experts, but an upvoting algorithm is used to rank them.

An answer’s **lexical** structure has proven to be very important for ranking correctly. This is what our model has taken into consideration and topics have been derived, rather than comparing answers word to word to their respective questions as in WordCount.

Experimentation has been conducted on Yahoo!Answers dataset and a model with accuracy **61%** has been produced. It was shown to outperform the reference model, which is WordCount in our case.

Some advantages, among many, of correct ranking of answers is the amount of time it would save for people surfing through and scrolling through plenty of answers only to find one which is technically correct and closely answers the question, along with providing helpful links and images.

For further improving our model’s accuracy, we suggest adding more heuristics such as a **user’s reputation**, which is well calculated from the number of answers he has answered correctly. If this heuristic can be incorporated in our regression model, our model’s correctness shall greatly increase.

VII. REFERENCES AND FOOTNOTES

[1]Gottipati, Swapna et al. “Finding relevant answers in software forums.” *2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011)* (2011): 323-332.
[2]Han, Sangdo et al. “Answer ranking based on named entity types for question answering.” *IMCOM '17* (2017).
[3]Barrón-Cedeño, Alberto et al. “Thread-Level Information for Comment Classification in Community Question Answering.” *ACL* (2015).
[4]Hou, Yongshuai et al. “HITSZ-ICRC: Exploiting Classification Approach for Answer Selection in Community Question Answering.” *SemEval@NAACL-HLT* (2015).
[5]Zhou, Zhi-Min et al. “Exploiting user profile information for answer ranking in cQA.” *WWW* (2012).
[6]David M. Blei, Andrew Y. Ng, and Michael I. Jordan. 2003. Latent dirichlet allocation. J. Mach. Learn. Res. 3, null (3/1/2003), 993–1022.

A.