Group Name	**Expert Finder**
Group members	**Radhika Simant** - rrs150130 **Kanchan Waikar** - kpw150030

Problem Description

Extract the "skill set" and skill level of the different users who have posted answers to questions on given community question-answering forum. This information will then be used in order to do analysis of the question and identification of the expert who has skillset that matches question's requirement.

Full implementation Details

1. Baseline system :

System accepts question, does full text search on downloaded raw dataset and returns first user who has answered similar question.

2. Improvement strategy :

Build an Ontology of the domain at runtime and use the same along with other statistical features available, assign weight to each skill, aggregate skills along with their weight at user level and use the same in order to extract the expertise level in each of the skill.

Features Used/Implemented

Lexical Features

- Stemming used in searching Skills. Lucene internally stems the text field and the same This leads to a better multi-term entity match. We used Lucene "Text" Field type used while indexing content.

Syntactic features:

1. Used Stanford POSTagger in order to extract Part of speech tags of the Question as well as Answer
2. Feature Implemented - Extract single term entities except the ones from StopWords List
3. Feature Implemented - Extraction of multi term entities. Entities must not contain any stop word as subpart of the same

Semantic Features:

1. Implemented Automated Extraction of Ontology From semistructured text

2. Overlap finding of Ontology extracted with Question ontology
3. Query Boosting on ParentOntology field.

Training

- Identify multi-term noun entities (ontology level 2) from questions, answers given, and associate the same with the answer authors as well as the users who have tried to give answer to the question asked.
- Extract the Tags (Ontology level 1) that are already available with Question, and set them as parent in the ontology.
- Run the "Expertise Extractor" on the data downloaded. Identify the skill that Question talks about using part of speech tagger and aggregate the knowledge at user level.

Assumptions:

1. **If a User has answered a question and his answer was given any upvotes, it means that he has some knowledge about the topic**
2. **If User answers multiple questions on a topic, it proves his knowledge about those topics only if he gets upvotes on the same.**
3. **Users don't get any credit for asking a question.**

   Note: If the question asked is on an advanced topic, it could be inferred that user has has good knowledge of the basics and is thus asking an advanced question. We can use Inferential analysis based on the question asked but that would need a much stronger and handcrafted ontology that is very specific for the domain. This can be considered as possible enhancement

Description

The aim is to aggregate the skillset of User by going through all the questions that he/she has answered and assigning the skill weight based on the expertise he/she has displayed in answering the question.

Algorithm

1. **For Every Question, Extract ontology and statistics about the same.**
   a. **Do Part of speech analysis of every answer, extract the domain dictionary. Extract the ontology from every sentence written by the user. Store skill and frequency with the user. Associate the ontology from the question Text with the user who has given answer to the question and other users have approved user's answer by giving him/her either upvotes or downvotes.**
   b. **Calculate thread's max upvote and downvote count. This statistic can be used for comparing the user's skill against the others who answered.**

2. Aggregate Skillset Information for each user, apply following syntactical rules while aggregating the skill level exhibited by the user.
    a. If User's answer was accepted by other users, Assign tag as "TopSkill" for the user. This would mean that he has some level of expertise in the given arena
    b. Assign the granular expertise extracted to the user with weight that is configurable under key "BEST_ANSWERTAG_WEIGHT"
    c. If a user has answered a question, and he has some upvotes, calculate the skill frequency for the user using formula

NORMALIZED_UPVOTE_WEIGHT*((double)entry.getNoOfUpvotes()/entry.getThreadUpvotesMaxCount()) * answeredOntology.getFrequency()

    d. Consider user's lack of expertise into consideration and adjust skill level.

NORMALIZED_DOWNVOTE_WEIGHT*((double)entry.getNoOfDownvotes()/entry.getThreadmaxDownvoteCount()) *answeredOntology.getFrequency()

- Write Rules based on number of Ontology level 1, ontology level 2, Upvotes for the answer, downVotes for the answer, whether answer was nominated as the "best answer", Skill frequency extracted for the given question and approval received by the answer from other users. Negative approval demonstrates lack of knowledge and thus should negate the positive frequency for the skill.
- Aggregate the information

Prediction

Based on Question entered, identify the skillset, match it with users and extract the one with highest skillset match. Give higher preference to the user who has knowledge of all skills.

Examples :

Following are the test cases executed on the available dataset :

1) A minimal dataset of a couple of questions was made. The model was trained over these questions and then a new question was given as a test input. The results observed were are follows :

Ontology found in the question : {[app[1], Camel Mule[1], purpose[1], kind[1], Spring[1], REST[1], task[1], ESB[1], Java web service architecture[1], HTTP endpoint[1], Spring Integration[1], case[1], apps[1]]}

The results obtained from lucene is as below :

Results: (Hint: Double-click on results to display all fields)

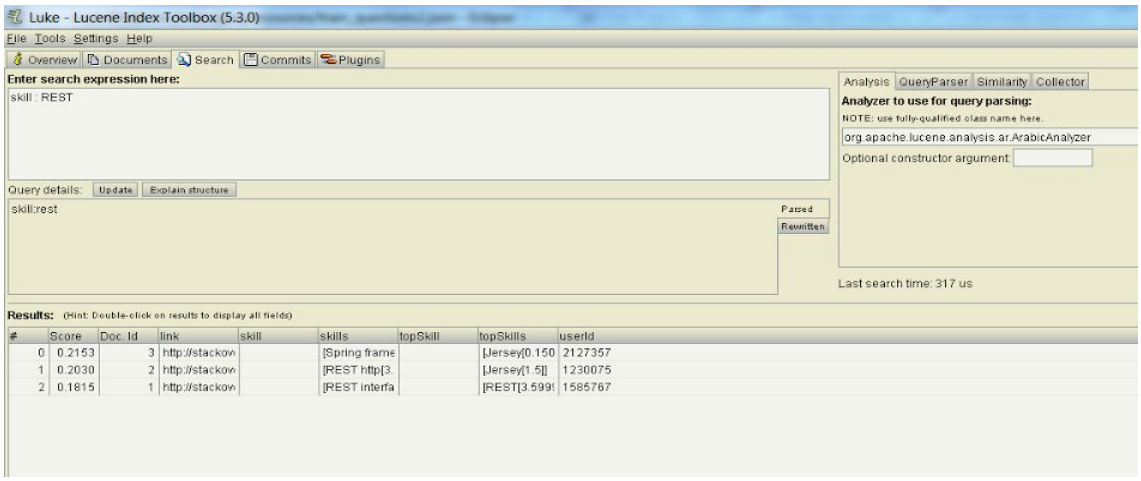
#	Score	Doc. Id	link	skill	skills	topSkill	topSkills	userId
0	0.6327	0	http://stackov		[Spring Integration[0.75], Tomc		[Spring[0.75], REST[1.5]]	1585767
1	0.0466	2	http://stackov		[Tomcat not[0.75], cache[0.75],		[REST[1.5]]	2597143
2	0.0416	4	http://stackov		[MVC pattern[1.5], I dont[1.5], o		[REST[3.0]]	16549
3	0.0001	3	http://stackov		[error[11.5], Java HotSpotTM 6		[java[11.5]]	1400755
4	0.0001	1	http://stackov		[this[0.25], [0.25], Change[0.25		[java[0.25], Java[0.75]]	1248436

The users in the order of expertise are returned by lucene, giving user with id 1585767 as the most probable user having expertise matching the most with the domain of the question.

2) A minimal dataset was made where there are no best answered posts. But a user who has answered several questions in the same domain and have received more than 1 upvote but never rated as the best answerer. A test question was given as an input having ontology matching with the user having answered several questions in a domain.

Ontology found in the question {[Spark Java[1], video file[1], cake[1], REST http[1], Stream[1], video[1], sections[1]]}

The results obtained from lucene is as below :



The user having ontology match along with the upvotes received on the answer is returned as the best answerer for the newly posted question. If we now want a user expert in “REST”, then user 2127357 is the most probable answerer. But if we consider the entire ontology of the question then the same user is not returned and the one with more match is returned which is user with ID 1230075.

Ontology Extraction:

Question:http://stackoverflow.com/questions/1794547/how-can-i-make-an-are-you-sure-prompt-in-a-dos-batchfile

Question:

Title: “How can I make an “are you sure” prompt in a DOS batchfile?“

Body: I have a batch file that automates copying a bunch of files from one place to the other and back for me. Only thing is as much as it helps me I keep accidentally selecting that command off my command buffer and mass overwriting uncommitted changes. What code would I need for my .bat file to make it say "are you sure", and make me type "y" before it ran the rest of the file? If anything but "y" is typed it should exit execution on that line. **Edit** Nov 27 Ok I marked this unanswered again because I still can't figure it out. When I call "exit;" it closes cmd.exe which is not what I want. All this because Windows implemented command buffer wrong [differently than what I am used to at least]

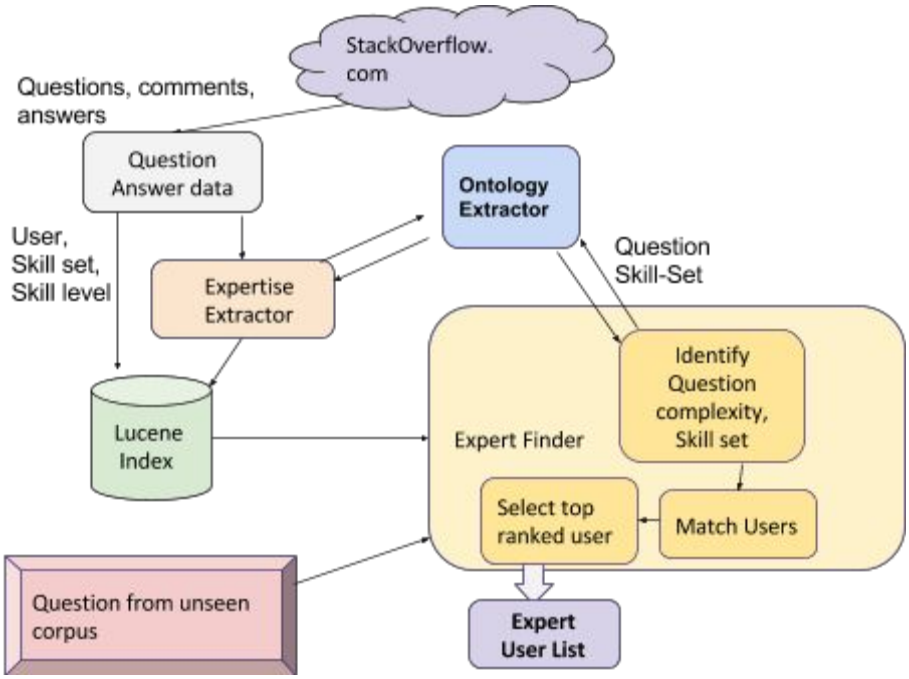
Ontology extracted:

[command buffer[2], rest[1], batch file[1], mass[1], Windows[1], command[1], exit[1], bat file[1], file[1], exe[1], files[1], DOS batchfile[1], cmd[1]]

Looking at the brief description, we can clearly see that the user is facing some problem with respect to batch file creation and needs help of somebody who is good at windows commands that could be written. Ontology extractor correctly extracts the skillset.

- **Programming tools (including third party software tools to be used) :**
  1. Java - JDK 1.8
  2. Stanford part of speech tagger
  3. List of stop Words - https://code.google.com/p/stop-words/
  4. Apache Lucene Index - For Combining and executing a Search on User Skillset Extracted. (stemming and lemmas)
  5. Luke - for viewing Lucene Index (Link - https://github.com/DmitryKey/luke/releases)

Architectural diagram :



Results

In order verify the accuracy of our expertise extractor, we executed the trained model on the input corpus and found that we were getting accuracy of 27.23%. After implementing the multi-term entity extraction, and tuning the same we were able to increase the accuracy to almost 46.98%. We further tuned the Rules that we had written for ontology aggregation and were able to increase the same to 54.61%. Further we modified the weights that we had assigned to different parameters used in aggregation and accuracy further increased to **60.60** %. In these 60.60% of the cases, the user who actually answered the question was returned among top 5.

A further analysis of some of the 40% entries for which we got different set of users than the original "Best Answerer", gave us deeper insight and our findings were as discussed below.

Examples:

- http://stackoverflow.com/questions/33183220/why-is-the-compiler-only-compiling-the-first-two-sections-of-my-code

words Extracted:

{[compiler wont[1], computer program game[1], compiler[1], user[1], sections[1]]}

User SkillSet

Top skills: [java[1.5], input[3.0]]

Granular Skills : [ORs[1.5], Change[1.5], variable[1.5], met[1.5], conditions[1.5], program[1.5], user[1.5], answer[1.5], circuit[1.5], knowledge[1.5], im[1.5], ANDs[1.5], contrast[1.5]]

We can clearly see that in this case it matched the word "input" Which was wrongly identified by StackoverFlow.com as an Important Tag. Since we are boosting weight of tags that stackoverflow chose, it resulted in identification of a user who has worked on lots of "input" based questions.

A summary of the problems encountered during the project and how these issues were resolved

Problems Encountered: Ontology Extractor Module

1. **Data Cleaning: Data contained lots of Tags, Source codes and Visual Delimiters which had to be removed.**
2. **Unstructured: This lead to Stanford POSTagger returning invalid Words as Nouns**
3. **Technical Ontology is "multi-termed".**

Approach Taken:

The main problem while extracting ontology was to keep it flexible. Also, Since the data is coming from stackoverflow.com, not all the sentences follow grammar rules. The Ontology not only needs to work for a specific domain, but should also work fairly for the dataset that was not included in the Training data set. Thus, Ontology was extracted completely at runtime based on Part of speech tagging rules. We started with extracting basic set of noun entities which belonged to the set { "NNP","NNPS", "NN", "NNS"}. Extraction of the same lead to entities which were fairly related to technical domain to which dataset belonged. We also noticed that there were quite a few words which were not technical but still were extracted as part of the vocabulary. It also contained some words, which could never be used as Noun. This was mainly because Sentences did not have perfect grammer.

In order to fix the same, we decided to add stopwords filtration on top of the entity set that we got from the training set. We used list from https://code.google.com/p/stop-words/ as our starting point After adding the same, it was evident that the dictionary we got was much more precise. We executed the dictionary extraction code and added quite a few new stop words and tuned the stop words list further. The dictionary we extracted looked good, but it still did not include any multiterm entities. We wrote a small algorithm that considered previous noun entity and extracted

multi-term entities too. The logic was further modified to ensure that multi-term entity did not contain any of the stop words too.

The Parent for the Ontology extracted was directly derived from the Tags that Stackoverflow itself maintains. These are very high level tags (E.g. Java is a Tag but AWT/Swing are not) and can be directly used as Parents for the extracted ontology. Also, there are several concepts that overlap across multiple technologies. E.g. Exception Management is associated with Android, Java as well as Scala.

Please find primary Function that performs Ontology Extraction in Appendix 1.

Lucene Index : Ontology overlap detection

Problem faced:

1. In order to do ontology matching, we need to check every skill with every skill of each user, this means that if a user has **Ni** users, and there are **Mi** skills for each user, and if there are **K** ontology entries found in the question, then we need to do Σ **Mi*Ni*K** for each user Ni. The scalability of the system will be highly restricted because of the above the number of comparisons involved as they have complexity of $\mathbf{O(n^3)}$. This resulted in a very slow program that was not able to scale beyond few hundred questions.
2. Lucene Index has its own Scoring Mechanism which results in highly relevant documents being ranked lower.

Solution Approach taken:

In order to avoid so many computations and keeping system scalable, we decided to use Lucene Index which internally uses inverted indexing mechanism and is very efficient in executing queries using which we can give higher weightage to a certain column. It also has In built similarity identification mechanism that can be leveraged for checking whether user has the identified skill.

- TopSkill - contains the Tag/Parent Skill “frequency” number of times (High Level Expertise)
- Skill - contains the Skill “frequency” number of times (Granular - more Focussed Expertise)
- UserInfo

In above index, we indexed every skill as well as top skill “frequency” number of times and wanted to use the frequency of the skill as the factor that would be used when we have to select an Expert user among two.

After executing search query on lucene index, we realized that it was returning the list of users, but it was giving more priority to the users who have less skills. This happens because Lucene’s scoring is implicitly based on following rules.

- Documents containing *all* the search terms are good

- Matches on rare words are better than for common words
- Long documents are not as good as short ones
- Documents which mention the search terms many times are good

Out of these rules, We were interested in the first and last but not middle two. In order to fix this problem, we have overridden the Default Similarity model and have made it Term_Frequency based only. Please find the document Similarity model we used for doing the overlap detection in Appendix 3.

Pending issues

None.

Potential improvements

- Automation of Ontology Hierarchy Extraction - Currently the tool only extracts one level of ontology from the dataset. This can be further improved to extract the multi-level ontology from the dataset.
- We can also further tune the stop words list and make it more exhaustive
- We can use different NLP tools to extract more information from comments given by users and use the same to do more accurate Skill Weightage assignment
- Use Sentiment detection in order to understand the sentiment behind questions

Appendix 1 : Ontology Extraction Logic (Level 2)

```
58      public List<String> extractNounEntities(String sentences) {
59          List<String> entities = new ArrayList<String>();
60          String[] sentencesArr = sentences.split("[.?!]");
61          for (String sentence : sentencesArr) {
62              List<HasWord> sent = Sentence.toWordList(sentence.split(" "));
63              List<TaggedWord> taggedSent = tagger.tagSentence(sent);
64              String previousNoun = null;
65              for (TaggedWord tw : taggedSent) {
66                  if (StringUtils.isNotBlank(tw.word())) {
67                      if (tw.tag().startsWith("NN")) {
68                          if (previousNoun != null) {
69                              entities.remove(previousNoun);
70                              previousNoun = previousNoun + " " + tw.word();
71                              if (!stopWords.contains(previousNoun.trim().toLowerCase().replaceAll("[:,//?]", ""))
72                                      && (!stopWords.contains(tw.word().trim().toLowerCase()))) {
73                                  entities.add(previousNoun);
74                              }
75
76                          } else {
77                              if (!stopWords.contains(tw.word().toLowerCase())) {
78                                  entities.add(tw.word());
79                                  previousNoun = tw.word();
80                              } else {
81                                  previousNoun = "";
82                              }
83                          }
84                      } else {
85                          previousNoun = null;
86                      }
87                  }
88              }
89          }
90          return entities;
```

Appendix 2:

```
Similarity sim = new DefaultSimilarity() {
    @Override
    public float idf(long docFreq, long numDocs) {
        //common terms are less important than uncommon ones
        return 1;
    }

    @Override
    public float tf(float freq) {
        //Return freq instead of sqrt(freq)
        return freq;
    }

    @Override
    public float queryNorm(float sumOfSquaredWeights) {
        // a term matched in fields with less terms have a higher score
        return 1;
    }

    @Override
    public float coord(int overlap, int maxOverlap) {
        // of the terms in the query, a document that contains more terms will have a higher score
        return super.coord(overlap, maxOverlap) * 100;
    }
};
```