Date: 11/30/2016

Introduced Features and Methods:

1. (Feature) Introduced custom analyzers and search api tweaks that achieve the following: (1) apply flexibility in project title search, (2) increase precision and recall on queries that include a programming language term, (3) understand synonyms such as JS and Javascript, and (4) prevent default filter from removing special characters from programming languages such as “C++” and “C#”, which by default became “C.”
2. (Method) Collected a bag of sample queries that represent the queries Stage users may conduct.
3. (Method) Established evaluation methods for comparing and contrasting old and new E.S. search settings.

Mapping and Search API

*The corresponding Excel outlines changes suggested in details.

Sample Search Queries

Note: The queries listed are hypothetical, not based on actual user data searched in Stage search, but based on assumptions on what users might search from simple keywords to more complex expressions. Some queries are inspired from posts found on Booz Allen Yammer as listed below:

Yammer Post 1:
"Looking for examples of accredited use of Hadoop on DoD networks. Specifically, are you using Hortonworks HDP or Cloudera distributions, what version, and where might I find evidence of their approved use (DADMS?)?
We are looking to use an HDFS-based Data Lake and some other Hadoop ecosystem tools in a DoD environment, it finding the latest examples of others doing this will be very helpful in getting started. If you're not using a pre-canned distro like Hortonworks or Cloudera, what are you using? Assembled on your own -- how was the accreditation for that?”

Yammer Post 2:
"We are looking into implementing Apache Ranger to provide access control policies for a data analytics environment for a DoD client. Does anyone know of any projects that have used Apache Ranger in production to secure a Hadoop cluster? Or are there alternate tools you've used for production cluster access control management?"

Yammer Post 3:

“Our solution that we are providing leverages both Apache Hive for persistence and Apache Spark as the primary computation engine. We believe we have run into some potential performance issues with the interaction between the two and was seeking some guidance on how to potentially track those issues down. We’re in the process of assessing our internal capabilities to prepare for a potential Oracle ERP opportunity. Would you please share Oracle ERP related BAH projects that you worked on, and/or your Oracle ERP skills?”

Simple Queries

• Python • Go • Java • Javascript • JS • C# • Shell • Angular JS • Jquery • apache nifi • cloud • big data • data science • cloud broker • data ingestion platform • Analytics	• web apps • projectjellyfish • kafka • spark • Hadoop • Apache ranger • Production cluster control management • Hortonworks HDP • Cloudera distributions • HDFS-based Data Lake • Hadoop ecosystem • Storm • Accumolo • Scrum • Enterprise • Public • Oracle

Complex Search Queries

- Hortonworks HDP or Cloudera distributions
- Hadoop on DoD networks.
- HDFS-based Data Lake and some other Hadoop ecosystem tools in a DoD environment
- a pre-canned distro like Hortonworks or Cloudera
- Apache Ranger to provide access control policies for a data analytics environment for a DoD client.
- any projects that have used Apache Ranger in production to secure a Hadoop cluster
- there alternate tools you've used for production cluster access control management
- both Apache Hive for persistence and Apache Spark as the primary computation engine
- Oracle ERP opportunity.

Evaluation Methods – Precision and Recall

Two Important Measures for Evaluating Search Performance:

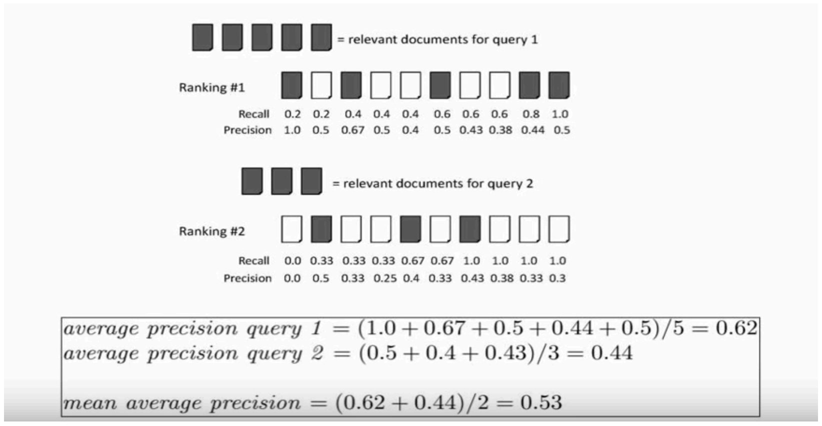
1. **Precision –** This measure represents the proportion of retrieved documents that are relevant. For instance, suppose a search engine contains a corpus of 1,000 documents. A user searches a query, and 100 documents are retrieved. Of those 100 documents, what percentage of them are relevant to the user? If 15 out of 100 documents shown in the result are relevant, then the precision for the query is 15%.
2. **Recall –** Represents the proportion of relevant documents that are retrieved. For instance, of those 1,000 documents as above, how many relevant documents are recalled in the result? Suppose a query has 500 documents that would be relevant. Of those 500, just 50 are displayed in the result in a total result size of 100. Then, the recall for that query would be 50/500 = 10%.

The reason both is needed is that sometimes a high score in one measure doesn't mean that the other measure would be high also. Using an earlier example, a search engine that retrieved 99 relevant documents out of a 100, the precision score would be 99%. But, it would score low in terms of recall, if the total relevant document size in a corpus of 1,000 was 500. It means that the recall is just at a shy of 10%. That's not so great. Ideally, you want a search engine that measures high in both precision and recall.

Manual Testing:

Definition:

Generate a list of queries that represents user search. Define a cutoff for the rank regarded for evaluations. For instance, if the cutoff is 10, then evaluation includes just the projects with the highest 10 relevancy scores. Keep the cutoff constant across queries. Operationalize what would be deemed relevant per query. Flag each document as relevant or not relevant. Calculate the mean average precision score.


= relevant documents for query 1
Ranking #1
= relevant documents for query 2
Ranking #2

Ranking #1										
Recall	0.2	0.2	0.4	0.4	0.4	0.6	0.6	0.6	0.8	1.0
Precision	1.0	0.5	0.67	0.5	0.4	0.5	0.43	0.38	0.44	0.5

Ranking #2										
Recall	0.0	0.33	0.33	0.33	0.67	0.67	1.0	1.0	1.0	1.0
Precision	0.0	0.5	0.33	0.25	0.4	0.33	0.43	0.38	0.33	0.3

$$\textit{average precision query 1} = (1.0 + 0.67 + 0.5 + 0.44 + 0.5)/5 = 0.62$$
$$\textit{average precision query 2} = (0.5 + 0.4 + 0.43)/3 = 0.44$$

$$\textit{mean average precision} = (0.62 + 0.44)/2 = 0.53$$

Pros:

- As queries get more complex, examining relevancy is easier than automated testing

Limitations:

- Procedure is time-consuming
- Usually requires multiple raters

Automated Testing:

Definition:

Auto-flag a document as relevant or not relevant. One suggestion might be using keyword matching. If a query is "Python," any document containing the word is deemed relevant; otherwise, not relevant. For each query term, define a bag of related terms. If any of the related terms is present in a document, then it is deemed relevant also.

Pros:

- Fast
- Assess recall in a bulk of queries
- Assess MVP in a bulk of queries.

Limitations:

- Procedure is simpler, but it's difficult to account for variations of words. For instance, the word Go is a programming language, but a document might contain the word "go," and that document might be falsely considered relevant.
- As queries get more complex, examining relevancy is harder.

Evaluation Report – Baseline vs New

The following tests were conducted to evaluate new settings against the baseline:

1. Conduct manual testing involving precision average on the 10 highest ranked documents
2. Conduct automated testing to calculate precision and recall on language query terms.
3. Conduct automated testing to calculate precision and recall on technology terms.
4. Conduct manual testing involving misspellings or incomplete query terms, such as "jelly" when referring to "Projectjellyfish."

**The sheet 2 of the corresponding Excel document contains on the evaluation scores displayed in the table below.

(1) Manual Testing – Mean Average Precision

Baseline average = 0.803 MAP score
New average = 0.653 MAP score

(2) Automated Testing Results – Language Terms

Language Evaluation				
	Baseline		New	
Queries	Precision	Recall	Precision	Recall
Python	0.9958	0.68	0.9839	1
Go	0.9308	0.91666	0.9053	1
Java	0.9063	0.84615	0.9369	1
Javascript	1	0.77	**1**	1
JS	1	0.097	1	0.78
C#	0.0909	0.11	0.3385	1
Shell	0.9752	0.19	0.9526	1
Averages	**0.842714286**	**0.515687143**	**0.873885714**	**0.968571429**

(3) Automated Testing Results – Technology Terms

Tech-Term Evaluation				
	Baseline		New	
Queries	Precision	Recall	Precision	Recall
spark	1	0.8	0.679166667	0.8
cloud	0.556855465	0.571428571	0.597167165	0.785714286
hortonworks hdp	0.875545635	0.2	0.875978188	0.3
accumulo	1	1	0.305555556	1
analytics	1	0.142857143	0.325357143	0.155844156
apache nifi	1	0.333333333	0.674587014	0.444444444
oracle	0.757575758	0.666666667	0.383333333	0.666666667
angular js	1	0.571428571	1	0.857142857
big data	0.48525641	0.828947368	0.186119551	0.842105263
enterprise	0.943399092	1	0.994345347	1
hdfs-based data lake	0.504840516	0.894736842	0.411068278	0.894736842
storm	0.513888889	0.6	0.275793651	0.6
projectjellyfish	1	0.666666667	0.725490196	1
jquery	1	1	0.71	1
cloudera distributions	1	1	0.208333333	1

web apps	0.544129037	0.3	0.424972161	0.327272727
public	1	0.973913043	0.838456243	0.982608696
data science	0.992282791	0.772151899	1	0.784810127
data ingestion				
platform	1	1	1	1
kafka	0.477777778	1	1	1
hadoop	1	1	0.681392093	1
Averages	**0.840550065**	**0.729625243**	**0.633195996**	**0.782067303**

(4) Manual Testing – Misspellings/Incomplete Queries

Query	Intent	Baseline	New
projectjelly	projectjellyfish	0	1
Fedswithspending	fedspendingtransparency.github.io	0	1
Angular	Angular	0	1
frontend	Front End	1	1
cyber sim cpt	cybersimcpt	0	1
jaba	projects w/ java	1	1
cms data lake	cmsdatalake	0	0
basicprotocal	BasicProtocolReverseEngineeringTalk	0	1
fedsspending	fedspendingtransparency.github.io	0	1
	Averages	**0.222222222**	**0.888888889**