CS160 - Homework 3 solutions

1. (2.5 points) 6.22 (1-2 sentences)

   If a query term is not in M, our dictionary, then it will not have an impact on cosine similarity with any of the documents, since no document vectors contain this term. The easiest way to deal with this is when creating the vector for the query, you ignore terms not in the dictionary. The only drawback to this is that if we do length normalization on the query after dropping terms, it will give us a different answer.

   Another common answer was to just leave them in. While this would work, it can cause complications since those terms don't actually exist in our index and when doing the dot product, we have to account for that. In IR, this may cause less of an issue depending on how we actual traverse the index.

2. (2.5 points) 7.1 (1-2 sentences)

   Ordering them in decreasing order makes doing champion list generating trivial, since it is the first r documents in the posting list.

3. (20 points) Calculating tf-idf

   Feel free to use matlab, python or excel to calculate these answers.

   (a) 6.10

   To get the TF-IDF entries, we multiply each document term frequency by the IDF for that term. So, all term frequencies of "car" are multiplied by 1.65, all "auto" by 2.08, etc.

	Doc1	Doc2	Doc3
car	44.55	6.6	39.6
auto	6.24	68.64	0
insurance	0	53.46	46.98
best	21	0	25.5

(b) 6.15

When we normalize by length, we're normalizing the document vectors. For each document, we calculate it's length as the square root of the sum of the square of it's entries. Then, we divide each entry for that document by the document length. Length normalization should be the LAST normalization we do on the data (i.e. after term normalization and after term weighting).

Document lengths: 49.7, 69.0, 66.5

	Doc1	Doc2	Doc3
car	.90	.076	.60
auto	.13	.79	0
insurance	0	.61	0.71
best	.42	0	.38

(c) Compute the document similarities with the document vectors above for the query "insurance car insurance for new autos" using boolean term frequncies for the query.

Since "auto" and "autos" are different terms, we do not have an entry for it. As mentioned above, for terms that don't occur in the index, we can just omit them. For IR, it is not uncommon to leave the query vector unnormalized, i.e. don't apply length normalization. In this case, the ranking doesn't change if we normalize or not and since it wasn't clear, I accepted both answers.

Query vector = [1, 0, 1, 0]
Doc1: .9 = .9
Doc2: .076+.61 = 0.69
Doc3: .60 + .71 = 1.31

(d) Compute the document similarities with the document vectors above for the query "insurance car insurance for new autos" using "**lnc**" query weighting. Use your proposal from above to handle out of vocabulary terms in the query. Did your ranking change from the previous question? What type of query would you expect to see a change in answer?

Query:
original - car: 1, insurance 2
term norm - car: 1, insurance 1.3
length norm (length=1.64) - car: 0.61, insurance, 0.79

Doc1: .61*.9 = .55
Doc2: .61*.0760 + .79*.61 = .53

Doc3: .61*.6 + .79*.71 = .93

The ordering does not change. Changing the frequencies of the terms could cause the ordering to change. For example, if we increased the frequency of "car" in our query then eventually, Doc1 would become better when using term frequencies, while the boolean normalization would not change.

4. (10 points) Let $K = 2$ and $r = 3$. Give an index and a query such that the list of candidate documents generated using the champion list approach does NOT contain the best K documents using a nnn.nnn weighting model (i.e. only tf weighted).

   Take the query $w_1 w_2$. If w_2 does not occur in any of the r documents where w_1 occurs most frequenty and vice versa, then the intersection of the two champion lists will not find a match.

5. (5 points) Cluster pruning - If we let $b_1 = 2$ and $b_2 = 1$, that is, assign followers to the two nearest leaders and then search documents associated with the closest leader to the query, can we still have the case that the closest document to the query is not found? If yes, provide an explaination, if no, a counterexample.

6. (5 points) Play with Google's page query system (found at www.google.com/advance_search, under "Page-specific tools" with the heading "Find pages similar to the page:"). How well does it work? How do you think it works?