1. Describe the advantages and disadvantges of the Boolean retrieval model vs. the ranked retrieval model.

Boolean Retrieval Model	Ranked Retrieval Model
Good for experts with a precise understanding of their needs and the collection. Good for applications. Not so good for most users (who can't or won't write good Boolean queries and are unwilling to wade through thousands of results). Boolean searches usually give you too few or too many results.	Good for most users. You can get relevant search results without being an expert and without a precise understanding of your needs or the collection. No "feast of famine" issue, eg. does not give you too many or too few results. You can use free text queries (one or more words in a human language), which is more user-friendly. The query search result yields an ordering over the top documents in the collection. Intuitive and easy to browse.

2. What is the idf of a term that occurs in every document? Compare this with the use of stop words.

Idf stands for "Inverse Document Frequency". It is used for attributing weights to terms in a query. The idea behind it is that rare terms are more informative than frequent terms (the, to, if, …) and should have higher weights.

idf of a term t is calculated as:

$$idf = \log(N / df_t)$$

where N = number of documents in the collection and df_t is the document frequency of the term.

In the case of a term that occurs in every document:

$$idf = \log(N / df_t) = \log(N / N) = \log(1) = 0$$

"Stop words" in Information Retrieval are very frequent terms (the, to, if, …). They will show up in every or almost every document and, hence, will have a very low idf (zero if they truly occur in every document).

3. Consider the table below of term frquencies for three documents from a collection of 806791 documents. First comput the idf values for each term and then compute the tf.idf weights for the terms for each document using the document frequency values (df_t column) in the table.

Table 1: Term frquencies and document frequencies.

term	Doc 1	Doc 2	Doc 3	df_t
car	27	4	24	18165
auto	3	33	0	6723
insurance	0	33	29	19241
best	14	0	17	25235

N = 806791

term	doc1	doc2	doc3	dtf	idf
car	27	4	24	18165	1.647526
auto	3	33	0	6723	2.079198
insurance	0	33	29	19241	1.622533
best	14	0	17	25235	1.504758

	tf.idf		
term	**doc1**	**doc2**	**doc3**
car	4.005734	2.639435	3.921459
auto	3.071227	5.236489	0
insurance	0	4.086373	3.995323
best	3.229403	0	3.356285

Formulas used: idf = log (N / dft) ; tf.idf = (1 + log $tf_{t,d}$) * log(N / dft)

4. How does the base of the logarithm in the calculation of inverse document frequency affect the tfi.idf score? How does the base of the logarithm affect the relative scores of two documents on a given query?

The smaller the base that we choose, the bigger the value of tfi.idf.

The base of the log isn't really important. We use log(N / dft) instead of (N / dft) and log(tf) instead of (tf) in order to "dampen" the effects. What matters is being consistent with the choice of the base of the logarithm so that the same system is applied to all query terms.

Two terms of a query that get their tfi.idf weights calculated using one base, and then calculated using a different base, will have the same order of importance/weight in the query regardless of which base was used.

5. Compute the Euclidean normalized document vectors for each of the three documents above, where each vector has four components, one for each of the four terms.

For normalizing the components of the document vectors we need to calculate the L_2 norm, then divide each of the vector components (the tf.idf) by that L_2 normalization factor:

	length-normalized		
term	**doc1**	**doc2**	**doc3**
car	0.668483	0.369284	0.600782
auto	0.512531	0.732639	0
insurance	0	0.571726	0.612098
best	0.538927	0	0.514195
L2 norm	5.992277	7.147439	6.527258

So the normalized vectors for each document become:
Doc1 = [0.668 , 0.512 , 0.000, 0.539]
Doc2 = [0.370, 0.732, 0.5712, 0.000]
Doc3 = [0.601, 0.00, 0.612, 0.514]

6. With the term weights as computed in the previous question, rank the three documents above by computed score for the query car insurance, for each of the following cases of term weighting in the query:

   (a) The wieght of a term is 1 if present in the query and 0 otherwise.

   (b) Euclidean normalized idf.

Query = q = (car, insurance) ; d = document

a) Score(q, doc1) = 1 + 0 = 1 ("car" is present = 1; "insurance" absent = 0)

Score(q, doc2) = 1 + 1 = 2

Score(q, doc3) = 1 + 1 = 2

In the situation where the weights would be straightforward 1 (present in document) or 0 (absent from document), we would want the highest score possible. Documents 2 and 3 would tie for being the most relevant, followed by doc 1.

b) The normalized query vector would be: q = [0.5, 0.0, 0.5, 0.0]

As a reminder, the normalized vectors for the documents were:

Doc1 = [0.668 , 0.512 , 0.000, 0.539]

Doc2 = [0.370, 0.732, 0.5712, 0.000]

Doc3 = [0.601, 0.00, 0.612, 0.514]

We want to calculate the cosine similarity between the query vector and each document vector:

$$cos(\vec{q}, \vec{d}) = \frac{\vec{q}}{|\vec{q}|} \cdot \frac{\vec{d}}{|\vec{d}|}$$

cos(q, doc1) = 0.5*0.668 + 0 + 0 + 0 = 0.334

cos(q, doc2) = 0.5*0.370 + 0 + 0.5*0.5712 + 0 = 0.471

cos(q, doc3) = 0.5*0.601 + 0 + 0.5*0.612 + 0 = 0.606

The documents with the smallest angle (=highest cosine) are the most relevant ones.

Most relevant to least relevant: doc3, doc2, doc1

c) Tf.idf weights (***for practise***): Score(q, d) = $\sum_{t \in q \cap d} tf.idf_{t,d}$

Score (q, doc1) = 4.006 + 0 = 4.006

Score(q, doc2) = 2.639 + 4.086 = 6.726 * *all numbers rounded*

Score(q, doc3) = 3.921 + 3.995 = 7.917

In this instance, the order of most relevance to least relevant is: doc3, doc2, doc1