MODEL SOLUTIONS

SETTER: Lucia Specia / Mark Hepple

Data Provided: None

DEPARTMENT OF COMPUTER SCIENCE Autumn Semester 2014-2015

TEXT PROCESSING 2 hours and 30 minutes

Answer the question in Section A, and THREE questions from Section B.

All questions carry equal weight. Figures in square brackets indicate the percentage of available marks allocated to each part of a question.

SECTION A

1. a) Two Information Retrieval systems, System 1 and System 2, each return a ranked list of 10 documents they believe to be relevant for a particular query. It is known that this collection has 12 relevant documents. The following table shows whether each document returned by each system is actually relevant (✓) or not (×) to the query.

Document	System 1	System 2
d1	✓	×
d2	×	✓
d3	✓	×
d4	✓	✓
d5	✓	✓
d6	×	×
d7	×	✓
d8	✓	✓
d9	×	✓
d10	×	✓

Compute the overall precision of each System, showing the equations as part of your answer. Then, compute the precision at two cutoff points: top 3 and top 5. Finally, discuss the differences between overall precision and precision at different cutoff points when comparing two or more Information Retrieval systems. Use your solution to exemplify your answer. [30%]

ANSWER:

[P1] Overall precision System 1 = TP/TP+FP = 5/10; Overall precision $System2 = 7/10$.

[P2] Precision at cutoff point 3, System 1 = 2/3; Precision at cutoff point 3, System 2 = 1/3. Precision at cutoff point 5, System 1 = 4/5; Precision at cutoff point 5, System 2 = 3/5.

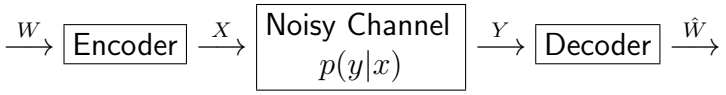
[P3] The difference between overall precision and precision at cutoff points is that the former considers the entire list of documents judged relevant by the IR system, as long as this ranked list can be. The latter only considers the top x documents, and therefore is better at evaluating whether the actual proposed ranking of the documents judged relevant is correct. In this particular case, System 2 is considered better according to overall precision, but if one looks at the top-ranked answers (both top 3 and top 5), one sees that System 1 is actually better at ranking relevant documents correctly.

b) What is the noisy channel model? Give a diagram of the model as part of your answer. Suggest a text processing context where the noisy channel model has been used. [30%]

ANSWER:

[P1] Using information theory, Shannon modelled the goal of communicating down a telephone line – or in general across any channel – in the following way. Imagine communication along a low-quality telephone line where the sender creates a message which is being transmitted over a channel with limited capacity; the receiver has to guess what the original message was. It is assumed that the output of the channel depends probabilistically on the input. The goal is to encode the message in such a way that it occupies minimal space while still containing enough redundancy to be able to detect and correct errors. On receipt, the message is decoded to give what was most likely the original message.

[P2]



[P3] A version of the noisy channel model has been used to model the machine translation process. As an example, suppose we want to translate a text from English to French. The channel model for translation assumes that the true text is French, but that unfortunately when it was transmitted to us, it went through a noisy communication channel and came out as English. All we need to do in order to translate is to recover the original French, i.e., to decode the English to get the French.

c) Explain two metrics to evaluate the quality of binary (negative/positive) sentiment analysis systems. Show their formulae as part of your answer. [20%]

ANSWER:

Based on an a gold-standard dataset (aka test corpora): i.e., text segments that have been classified by humans into positive vs negative cases, we can compare the system output to human classifications and compute metrics like (any 2 answers, 10% each):

Accuracy = number of correctly classified segments / number of segments, where correctly classified segments are those where the system agrees with the human decision (i.e., both positive or both negative)
Precision Positive = number of segments correctly classified as positive / number of segments classified as positive (idem for Precision Negative)
Recall Positive = number of segments correctly classified as positive / number of positive segments (idem for Recall Negative)
F-measure Positive = (2 * Precision Positive * Recall Positive) / Precision Positive + Recall Positive (idem for F-measure Negative)

d) LZ77 is a popular compression method, used in common compression utilities such as *gzip*. The following shows some possible LZ77 encoder output (assuming the encoding representation presented in the lectures of the Text Processing module):

$$\langle 0,0,b\rangle\langle 0,0,a\rangle\langle 0,0,d\rangle\langle 3,3,b\rangle\langle 1,3,a\rangle\langle 1,3,d\rangle\langle 1,3,a\rangle\langle 11,2,a\rangle$$

Sketch how LZ77 works. State the output that would be produced by decoding the above representation, showing how your answer is derived. [20%]

ANSWER:

The **key idea** underlying the LZ77 adaptive dictionary compression method is to replace substrings with pointers to previous occurrences of the same substrings in same text. During decoding, the text is recreated incrementally, with the substring for each pointer being found by looking back into the material decoded so far.

The encoder output is a series of triples where:

- the first component indicates how far back in the decoded output to look
- the second indicates the length of the substring to be copied from there
- the third is a character to be added after the copied substring

Decoding of the example representation would proceed as follows:

1. $\langle 0,0,b\rangle$ Go back 0 copy for length 0 and end with b: b
2. $\langle 0,0,a\rangle$ Go back 0 copy for length 0 and end with a: ba
3. $\langle 0,0,d\rangle$ Go back 0 copy for length 0 and end with d: bad
4. $\langle 3,3,b\rangle$ Go back 3 copy for length 3 and end with b: $badbadb$
5. $\langle 1,3,a\rangle$ Go back 1 copy for length 3 and end with a: $badbadbbbba$
6. $\langle 1,3,d\rangle$ Go back 1 copy for length 3 and end with d: $badbadbbbbaaaad$
7. $\langle 1,3,a\rangle$ Go back 1 copy for length 3 and end with a: $badbadbbbbaaaaddddda$
8. $\langle 11,2,a\rangle$ Go back 11 copy for length 2 and end with a: $badbadbbbbaaaaddddabba$

Thus, the decoded string is:

`badbadbbbbaaaaddddabba`

SECTION B

2. In the context of Information Retrieval, given the following documents:

Document 1: Your dataset is corrupt. Corrupted data does not hash!!!

Document 2: Your data system will transfer corrupted data files to trash.

Document 3: Most politicians are corrupt in many developing countries.

and the query:

Query 1: hashing corrupted data

a) Apply the following term manipulations on document terms: *stoplist removal*, *capitalisation* and *lemmatisation*, showing the transformed documents. Explain each of these manipulations. Provide the stoplist used, making sure it includes punctuation. [20%]

ANSWER:

[P1](5%)

Document 1: dataset corrupt corrupt data hash

Document 2: data system transfer corrupt data file trash

Document 3: politician corrupt developing country

[P2](10%) A *stoplist* is a list of words ('stop-words') that are ignored when documents are indexed. These are words that are so widespread in the document collection that they are of very little use for discriminating between documents that are/are not relevant to a query. *Capitalisation* refers to the process of normalising the case of words so that a single case is used, for example, all words are lowercased (e.g. Corrupted = corrupted) disregarded. *Lemmatisation* is the process of reducing words that are morphological variants to their common form, e.g. so that variants *corruption, corrupted, corrupt*, etc. are reduced to a canonical form *corrupt*.

[P3](5%) Stoplist: {`your, is, are, in, will, to, most, many, not, !, .`}

b) Explain what is meant by an *inverted index* and why such indices are important in the context of Information Retrieval. Show how Document 1, Document 2 and Document 3 would be represented using an inverted index which includes term frequency information. This inverted index should not have more than 10 words.

[20%]

ANSWER:

[P1] An inverted index is a data structure that lists for each word in a document

collection all documents that contain it and the frequency of occurrence in each document. An inverted index makes it easy to search for 'hits' of a query word, i.e. one just goes to the part of the inverted index that corresponds to the query word to get a list of document identifiers for documents containing the word. A more sophisticated version of an inverted index might also record position information for occurrences, to allow for efficient retrieval of phrases.
[P2]

term-id	word	docs
1	corrupt	d1:2, d2:1, d3:1
2	country	d3: 1
3	data	d1:1, d2:2
4	dataset	d1:1
5	developing	d3:1
6	file	d2:1
7	hash	d1:1
8	politician	d3:1
9	system	d2:1
10	transfer	d2:1
11	trash	d2:1

c) Using *term frequency* (TF) to weight terms, represent the documents and query as vectors. Produce rankings of Document 1, Document 2 and Document 3 according to their relevance to Query 1 using two metrics: Cosine Similarity and Euclidean Distance. Show which document is ranked first according to each of these metrics.
[30%]

ANSWER:

[P1] Using the term order taken from the inverted index above, we can represent the three documents and the query as vectors as follows:

d1: $\langle 2, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0 \rangle$
d2: $\langle 1, 0, 2, 0, 0, 1, 0, 0, 1, 1, 1 \rangle$
d3: $\langle 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0 \rangle$
q: $\langle 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0 \rangle$

[P2] **Ranking using cosine similarity**: The cosine between two vectors $\vec{d}$ and $\vec{q}$ is computed as:

$$cos(\vec{d}, \vec{q}) = \frac{\vec{d} \cdot \vec{q}}{|\vec{d}||\vec{q}|} = \frac{\sum_{i=1}^{n} d_i q_i}{\sqrt{\sum_{i=1}^{n} d_i^2} \sqrt{\sum_{i=1}^{n} q_i^2}}$$

The vector magnitudes and cosine values are then:

$$|\mathrm{d1}| = \sqrt{2^2 + 1^2 + 1^2 + 1^2} = \sqrt{7} = 2.65$$

$$|\mathrm{d2}| = \sqrt{1^2 + 2^2 + 1^2 + 1^2 + 1^2 + 1^2} = \sqrt{9} = 3$$

$$|\mathrm{d3}| = \sqrt{1^2 + 1^2 + 1^2 + 1^2} = \sqrt{4} = 2$$

$$|\mathrm{q}| = \sqrt{1^2 + 1^2 + 1^2} = \sqrt{3} = 1.73$$

$$cos(\mathrm{d1}, \mathrm{q}) = \frac{2 + 1 + 1}{\sqrt{(7)} * \sqrt{(3)}} = 4/(2.65 * 1.73) = 0.87$$

$$cos(\mathrm{d2}, \mathrm{q}) = \frac{1 + 2}{\sqrt{(9)} * \sqrt{(3)}} = 3/(3 * 1.73) = 0.58$$

$$cos(\mathrm{d3}, \mathrm{q}) = \frac{1}{2 * \sqrt{(3)}} = 1/(2 * 1.73) = 0.29$$

The cosine values computed for the three documents indicate that document 1 is more relevant to the query.

[P3] **Ranking using euclidean distance**: Euclidean distance is computed as $\sqrt{\sum_{i=1}^{n}(q_i - d_i)^2}$. In this case:

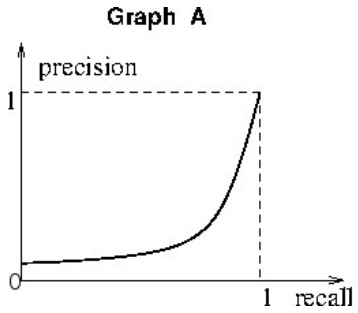
$$dis(\mathrm{d1}, \mathrm{q}) = \sqrt{(2-1)^2 + (1-1)^2 + (1-0)^2 + (1-1)^2} = \sqrt{2} = 1.41$$

$$dis(\mathrm{d2}, \mathrm{q}) = \sqrt{(1-1)^2 + (2-1)^2 + (1-0)^2 + (0-1)^2 + (1-0)^2 + (1-0)^2 + (1-0)^2} = \sqrt{6} = 2.45$$

$$dis(\mathrm{d3}, \mathrm{q}) = \sqrt{(1-1)^2 + (1-0)^2 + (0-1)^2 + (1-0)^2 + (0-1)^2 + (1-0)^2} = \sqrt{5} = 2.24$$

Documents 1 is ranked as more relevant (this is a distance metric - the smaller the score, the more relevant the document).

d) Define the precision and recall measures in Information Retrieval. Is Graph A a possible precision/recall graph? Is a curve of this shape likely when evaluating the results of a realistic Information Retrieval system such as Google? Explain your answers. [20%]



ANSWER:

[P1](10%) Assuming that: RET is the set of all documents the system has retrieved for a specific query; REL is the set of relevant documents for a specific query; $RETREL$ is the set of the retrieved relevant documents, i.e., $RETREL = RET \cap REL$. Precision is defined as $|RETREL|/|RET|$ and recall is defined as $|RETREL|/|REL|$.

[P2](5%) Graph A is not possible. The fact that the curve touches the point (1,1) indicates both precision and recall are one, meaning that all documents retrieved are relevant. However, this contradicts the fact that precision is higher than zero when recall is zero, which is not possible given that the nominator for both equations are the same.

[P3](5%) A curve like this is not likely in a realistic IR scenario. Often as recall goes up, precision goes down, since more retrieved documents often results in some of them not being relevant.

e) Discuss the advantages and disadvantages of $boolean$ versus $ranked$ approaches to Information Retrieval. [10%]

ANSWER:

Boolean approaches model the retrieval as a binary decision: either a document is relevant or it isn't. This is often achieved by using complex queries through the combination of basic terms or keywords (which may be drawn from a specialised indexing vocabulary) using logical operators (such as AND, OR, NOT, etc). The advantage is that the approach allows the formulation of very precise queries, which may also benefit from specialised manual indexation of documents. However, expertise is often required for formulating good/precise queries. In addition, the lack of ranking can present a problem if many documents are returned for a query.

Ranked retrieval approaches compute a similarity, under some measure, between queries and documents, returning a ranked list of candidate relevant documents. Not all search terms are required to appear in documents, and term frequency/weighting may be exploited in ranking. The approach does not allow such precise formulation of queries as boolean retrieval, but query formulation does not require particular expertise, and the ranking of results is helpful where many are likely to be returned, e.g. in web search.

3. a) When applied to translating from French to English, the IBM approach to Statistical Machine Translation can be expressed by the following equation:

$$E^* = \underset{E}{\text{argmax}}\ P(E) \cdot P(F|E)$$

Explain what this equation means, and indicate the role played by the components $P(E)$ and $P(F|E)$ in the process of translation. [20%]

ANSWER:

The equation tells us that the optimal English translation E^* of a French sentence F is found by identifying the English sentence E that maximises the right hand side of the equation, which has two components, that serve different purposes, as follows. The (monolingual) Language Model (LM) $P(E)$ favours candidate Es that are good strings of English, i.e. this component provides for the *fluency* of translations. The Translation Model (TM) $P(F|E)$ tests if candidate Es are likely translations of F, i.e. it provides for *faithfulness* in translation.

b) Show how the equation given in 3(a) is derived using Bayes Rule. What is the benefit of this approach as compared to one attempting to use the probability $P(E|F)$ directly? [30%]

ANSWER:

The starting point is the idea that we want to find the English sentence E that is most probable *given* the F we want to translate, which is expressed by the following equation:

$$E^* = \underset{E}{\text{argmax}}\ P(E|F)$$

Using Bayes Rule, we can restate this as:

$$E^* = \underset{E}{\text{argmax}}\ \frac{P(E) \cdot P(F|E)}{P(F)}$$

Here, $P(F)$ is constant across different E, so it doesn't affect the maximisation. hence, it can be dropped to give:

$$E^* = \underset{E}{\text{argmax}}\ P(E) \cdot P(F|E)$$

It would be very difficult to model $P(E|F)$ directly, due to data sparseness problems, i.e. it would difficult to get good probability estimates from even a v.large amount of parallel text. The decomposition to $P(E) \cdot P(F|E)$ allows us to break the problem into smaller problems, with probabilities that are easier to acquire from data. The

$P(F|E)$ can be used to provide a collection of English $words$ that are good candidates for translating F, which is much easier than determining complete ordered English sentences for translating F. The $P(E)$ then deals with finding an ordering of candidate words that constitutes good English. These two models can be trained independently.

c) Consider the following text processing techniques studied in the context of Information Retrieval: capitalisation, stop-word removal and stemming. Discuss whether or not each of these techniques could be useful in the context of Phrase-based Statistical Machine Translation and why. Would each of these techniques be equally applicable to the source and target language data? At which stage of the process would they be applied? Give examples of words to support your answer. [25%]

ANSWER:

Capitalisation would be very useful and it is often a common technique as it normalises different words like Cat and cat as the same word, making the statistics more reliable. It could be used for both source and target languages, and for the language model component (target language). The case in the target language could be restored using recasing models. Stop-word removal would harm the performance of the translation system, as important (although content-less) words would make the translations ungrammatical (e.g. 'the cat sat on the mat' could become 'cat sat mat'), and therefore should be used neither in the source nor in the target languages. Stemming could be used on the source language only, to put together words like 'cat' and 'cats', but would harm performance if used in the target. These techniques would be used to pre-process the parallel corpus (or its source/target side only) before the process of learning a translation system starts (before word alignment).

d) Ensuring that output is grammatical and fluent is one of the main goals in machine translation. Explain how this problem is addressed in Phrase-based Statistical Machine Translation approaches. Your explanation should specify what type of data is necessary to ensure fluency in the building of a Phrase-based Statistical Machine Translation system. Discuss how you would collect such data for a given language, say Spanish. Would you pre-process the data in any way? Cite and explain two pre-processing techniques. [25%]

ANSWER:

[P1](10%) Fluency is handled specifically by the language model component in SMT.

This component is defined as the P(E), where E is the target language. In other words, the probability that the sequence of words in a candidate translation is common in the target language.

[P2](5%) The language model is built from a monolingual corpus of the target language, often a very large one. The model itself consists of relative frequencies of sequences of words (e.g. 1-5), and its application to a candidate translation consists of the product of these relative frequencies.

[P3](10%) Monolingual corpora are often easy to acquire. A large Spanish corpus could be collected by simply crawling websites with textual content from Spanish speaking countries. Language identification would be important to make sure the texts are actually in Spanish. Cleaning the content to remove url and other markup would be one pre-processing step. Capitalisation could be another pre-processing step.

4. a) Text compression techniques are important because growth in volume of text continually threatens to outstrip increases in storage, bandwidth and processing capacity. Briefly explain the differences between:

(i) **symbolwise** and **dictionary** text compression methods; [10%]

ANSWER:

- **Symbolwise methods** work by estimating the probabilities of symbols (characters or words/non-words) and coding one symbol at a time using shorter codewords for the more likely symbols
- **Dictionary methods** work by replacing word/text fragments with an index to an entry in a dictionary

(ii) **modelling** versus **coding** steps; [10%]

ANSWER:

Symbolwise methods rely on a modeling step and a coding step

- **Modeling** is the estimation of probabilities for the symbols in the text – the better the probability estimates, the higher the compression that can be achieved
- **Coding** is the conversion of the probabilities obtained from a model into a bitstream

(iii) **static**, **semi-static** and **adaptive** techniques for text compression. [10%]

ANSWER:

Compression techniques can also be distinguished by whether they are

- **Static** – use a fixed model or fixed dictionary derived in advance of any text to be compressed
- **Semi-static** – use current text to build a model or dictionary during one pass, then apply it in second pass
- **Adaptive** – build model or dictionary adaptively during one pass

b) Sketch the algorithm for Huffman coding, i.e. for generating variable-length codes for a set of symbols, such as the letters of an alphabet. What does it mean to say that the codes produced are *prefix-free*, and why do they have this property? [20%]

ANSWER:

- To begin, we need the relative probabilities of the symbols in the symbol set.
- We start the algorithm by creating a leaf node for each symbol, which is labelled with its associated probability.
- Then, iterate over this collection of nodes, until only one node remains, as follows:
  - with each iteration, select and remove two nodes, which are joined under a new parent node
  - crucially, the nodes selected must have the lowest associated probabilities (or strictly, there must be no other nodes that have lower probabilities)
  - the new parent node is labelled with a probability value that is the sum of probabilities of the two child nodes
  - the new node is added back into the collection of nodes
- The single node that remains represents a binary tree whose root should be labelled with probability 1.0, and whose leaf nodes store the full set of symbols.
- Finally, a code-tree is produced by labelling the left and right side of each branch within the tree with 0 or 1, respectively. Then, the path from the root to the symbol at any leaf node gives a unique sequence of 0s and 1s, which is the code for this symbol.

That codes are *prefix-free* means that no symbol has a code which is a prefix of any other code, e.g. as would hold if there were codes `110` and `1101`. This property follows from the fact that symbols only appear on the leaf nodes of code trees, as a consequence of the algorithm. There could only be a code that was the prefix of another if its symbol was labelled on a non-leaf node, i.e. so that the path from the root to this symbol's node then *continued on* to the node for the second symbol.

c) We want to compress a large corpus of text of the (fictitious) language *Fontele*. The writing script of Fontele employs only the six letters found in the language name (`f,o,n,t,e,l`) and the symbol ␣, used as a 'space' between words. Corpus analysis shows that the probabilities of these seven characters are as follows:

Symbol	Probability
`e`	0.3
`f`	0.04
`l`	0.26
`n`	0.2
`t`	0.04
`o`	0.1
␣	0.06

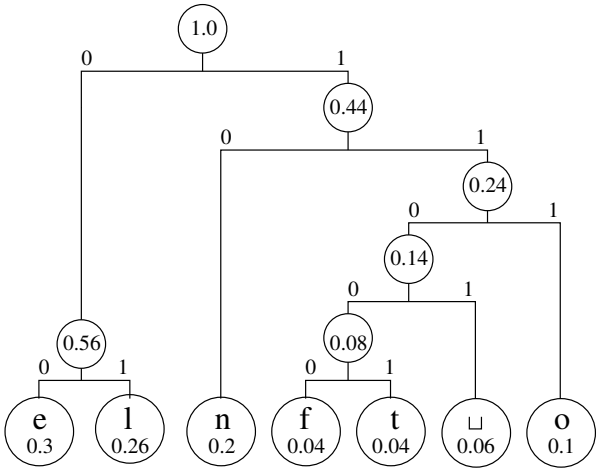
(i) Show how to construct a Huffman code tree for Fontele, given the above probabilities for its characters. Use your code tree to assign a binary code for each character. [20%]

ANSWER:

Start off by creating a leaf node for each character, with associated probability (a). Then join two nodes with smallest probabilities under a single parent node, whose probability is their sum, and repeat until only one node left. Finally, 0's and 1's are assigned to each binary split, giving the result shown in (b).

(a)

(b)



The code for each symbol corresponds to the sequence of 0s and 1s encountered on path from the root of the tree down to the leaf for that symbol, which gives the following codes:

Symbol	Probability	code
e	0.3	00
f	0.04	11000
l	0.26	01
n	0.2	10
t	0.04	11001
o	0.1	111
␣	0.06	1101

(ii) Given the code you have generated in 4(c)(i), what is the average bits-per-character rate that you could expect to achieve, if the code was used to compress a large corpus of Fontele text? How does this compare to a minimal fixed length binary encoding of this character set? [20%]

ANSWER:

We assume that the distribution of characters in the large corpus is the same as the one that we have been given from previous corpus analysis. Then, can calculate the average bits-per-character rate summing across symbols for the probability of that symbol multiplied by its code length:

Symbol	Probability	code	Code length	$p \times len$
`e`	0.3	00	2	0.6
`f`	0.04	11000	5	0.2
`l`	0.26	01	2	0.52
`n`	0.2	10	2	0.4
`t`	0.04	11001	5	0.2
`o`	0.1	111	3	0.3
␣	0.06	1101	4	0.24
				2.46

giving a cost of 2.46 bits-per-character. A minimal fixed length binary encoding would require 3 bits per character, i.e. this is sufficient to encode up to $2^3 = 8$ characters, whilst a 2 bit code would allow only $2^2 = 4$ characters, and we have 7 to encode. Using a 3 bit fixed-length code would result in corpus storage requiring about 20% more space than the Huffman codes.

(iii) Use your code to encode the message "`telefone␣noel`" and show the resulting binary representation. Compare the average bits-per-character rate achieved for this message to the expected rate that you computed in 4(c)(ii), and suggest an explanation for any difference observed between the two values. [10%]

ANSWER:

Encoding for "`telefone␣noel`" will be

```
t     e  l  e  f     o   n  e  |_|  n  o   e  l
11001 00 01 00 11000 111 10 00 1101 10 111 00 01
```

This representation has 36 bits, for a 13 character message, giving an average bits-per-character rate of $36/13 = 2.77$, which is clearly worse than the expected 2.46 bpc. This is due to the distribution of chars within the message diverging considerably from the expected distribution, e.g. the message contains two chars with 5-bit codes, having probabilities of 0.04, i.e. such that we might expect around 1 such char per 25 chars, and here we have two occurrences with a 13 letter message.

5. a) Consider the two sentences:

- *My new phone works well, is very pretty and much faster than the old one.*
- *My new phone has 32GB of memory and plays videos.*

What is the first step to detect the sentiment in these two sentences? Should both these sentences be addressed in the same way by Sentiment Analysis approaches? If not, explain a common approach to select only relevant sentences for Sentiment Analysis. [20%]

ANSWER:

[P1](10%) The first step is to run a subjectivity analysis. This has to do with whether the text (word/phrase, sentence, document) contains opinions, emotions, sentiment, or simply facts. Only subjective sentences should then be put forward for sentiment analysis, which has to do with the actual polarity of the text (word/phrase, sentence, document): positive, negative, or more fine-grained distinctions.

[P2](5%) In the example, only the first sentence is subjective and therefore has a sentiment that can be analysed.

[P3](5%) A simple rule-based subjectivity classifier can be built: a sentence/document is subjective if it has at least n (say 2) words that belong to an emotion words lexicon; a sentence/document is objective otherwise.

b) Given the following sentences S1 to S4 and opinion lexicon of adjectives, apply the weighted lexical-based approach to classify EACH sentence as **positive**, **negative** or **objective**. Show the final emotion score for each sentence, and also how it was generated. In addition to using the lexicon, make sure you consider any general rules that have an impact on the final decision. Explain these rules when they are applied. [15%]

Lexicon:

awesome	5
boring	-3
brilliant	2
funny	3
happy	4
horrible	-5

(S1) He is brilliant and funny.
(S2) I am not happy with this outcome.
(S3) I am feeling AWESOME today, despite the horrible comments from my supervisor.
(S4) He is extremely brilliant but boring, boring, very boring.

ANSWER:

(3% each sentence (=12%) + 3% general model) The general weighted lexical-based approach counts positive (Cpos) and negative (Cneg) words in the text and weights them using the weights in the lexicon given:
If Cpos $>$ Cneg then positive
If Cpos $<$ Cneg then negative
If Cpos $=$ Cneg $= 1$ then objective

(S1) Emotion(brilliant) = 2; Emotion(funny) = 3. Therefore Cpos = 2+3 and Cneg = 0, so Cpos $>$ Cneg = positive

(S2) Emotion(happy)= 4; "not" is detected in neighbourhood (of 5 words around). Emotional valence of term is decreased by 1 and sign is inverted. Therefore Emotion(happy)=-3, and Cneg=-3, so Cneg $>$ Cpos = negative

(S3) Emotion(horrible) = -5, Emotion(awesome) = 5, but "awesome" is intensified because it is in capital letters, and in this case it's intensified by 1 (because it's a positive word). Therefore, Cneg = -5, Cpos = 6, so Cpos $>$ Cneg = positive

(S4) Emotion(brilliant) = 2; Emotion(boring) = -3, but it happens 3x, so Emotion(boring) = -9. "extremely" is a (positive) intensifier in this case, with +2 added, so Emotion(brilliant) = 4. "very" is a (negative) intensifier in this case, with -2 subtracted, so Emotion(boring) = -11. Cpos = 4 and Cneg = -11, therefore
Cneg $>$ Cpos = negative

c) According to Bing Liu's model, an **opinion** is said to be a quintuple $(o_j, f_{jk}, so_{ijkl}, h_i, t_l)$. Explain each of these elements and exemplify them with respect to the following text. Identify the features present in the text, and for each indicate its sentiment value as either *positive* or *negative*. Discuss two language processing challenges in automating the identification of such elements. [25%]

> "I have just bought the new iPhone 5. It is a bit heavier than the iPhone 4, but it is much faster. The camera lenses are also much better, taking higher resolution pictures. The only big disadvantage is the cost: it is the most expensive phone in the market. Lucia Specia, 12/08/2014."

ANSWER:

[P1](10%)

- o_j is a target object: iPhone 5
- f_{jk} is a feature of the object o_j: weight, speed, camera/lenses/pictures, price
- so_{ijkl} is the sentiment value of the opinion: negative, positive, positive, negative
- of the opinion holder h_i (usually the author of the post): Lucia Specia
- on feature f_{jk} of object o_j at time t_l: 12/08/2014.

[P2](15%) Some of the challenges are (only two necessary):

- Need Named Entity Recognition to identify target objects, such as iPhone 5.
- Need Information Extraction to extract features of the target object (properties of objects), as well as time and holder.
- Need co-reference resolution to know that "it" = iPhone 5.
- Need synonym match when words used do not belong to lexicon of opinion words, e.g. fast versus efficient
- A lot of the opinions in this example are actually expressed in relative terms, in comparison to another object (iPhone 4), so it is hard to identify the sentiment of the object features (faster = fast?; better = good? higher resolution = high resolution?)

d) Assume a lexicon-based approach to binary Sentiment Analysis. A manually created initial lexicon is available which contains only three positive words:

- good
- nice
- excellent

and three negative words:

- bad
- terrible
- poor

This lexicon needs to be expanded in order for the approach to be effective in a realistic task. Explain two alternative methods to expand this lexicon automatically. Which of these methods should result in the larger lexicon and why? [20%]

ANSWER:

[P1](14%, 7% per method) The two most common methods to create lexica for Sentiment Analysis are 1) lookup in dictionaries or lexical databases like WordNet for synonyms in both sets of seed words to expand those sets, for antonyms in the negative set (generating positive words) and in the positive set (generating negative words). This can be repeated once the sets become larger. 2) corpus-based methods where patterns are built from the seed words, such as "good and ", "fragile but ", "very tasty and ", and searched for in corpora (such as the Web). In this case, all examples of patterns should result in more positive words or phrases (such as "good and reliable").

[P2](6%) The corpus-based method is more flexible (patterns can include phrases) and possible for many languages (as long as corpora are available), and therefore it could result in larger lexica.

e) Explain the intuition behind using a Naive Bayes classifier for Sentiment Analysis. Give the general classifier equation as part of your answer. What are the main components in this classifier? Give two types of features that could be used and provide examples for these types of features. [20%]

ANSWER:

[P1](7%) A Naive Bayes classifier, like other machine learning algorithms, is a model learnt from examples labelled for sentiment. It is a model that estimates the probability of each class c_i given a text segment T: $P(c_i|T) = \frac{P(c_i)P(T|c_i)}{P(T)}$, where T is described by a number of features $t_1, ..., t_j$. Assuming independence between features in the text segment (the "naive" assumption): $P(T|c_i) = P(t_1, t_2, ..., t_j|c_i) = \prod_{j=1}^{n} P(t_j|c_i)$.

[P2](7%) Main components:

- **Prior**: probability of segment having class c_i

$$P(c_i)$$

- **Likelihood**: product of probabilities of each feature value of segment occurring with class c_i

$$\prod_{j=1}^{n} P(t_j|c_i)$$

- **Evidence**: product of probabilities of features of segment – constant term, can be disregarded:

$$\prod_{j=1}^{n} P(t_j)$$

[P3](6%) Features could be (only two necessary): adjectives in the text segments (e.g. good, happy, terrible), verbs (e.g.: like, dislike, hate), negation words (e.g.: not, don't), or all words in the text segments.

END OF QUESTION PAPER