

11-682/15-492:
Introduction to IR, NLP, MT and Speech
Midterm Exam
Solutions

October 21, 2003

- Before you go on, **write your name at the space provided on the bottom of this page.**
- There are 9 pages in this exam (including this page). Verify that you have a complete copy.
- Write up your answers following each question in the exam. Adequate space has been provided.
- The exam is open book and open notes and is worth a total of 100 points.
- It should require 70 minutes to complete, though you will be given 80 minutes. Budget your time accordingly.
- Keep answers short and to-the-point. Concise, direct answers will receive more credit than longer, essay-like answers.
- If you find a question ambiguous, state your assumptions precisely, and proceed.

Good Luck !!

Question	Points	Score
1	40	
2	10	
3	50	
Total	100	

Student Name: _____

Problem 1 - Text Analysis (28 minutes)

Suppose you work for a government agency that is about to change Clean Air regulations. By law the agency must allow the public to comment on the proposed changes. This is a high profile issue, so the agency expects to receive 300,000 comments, via the Web and email, and by law it must “consider” each one. The President has promised a decision 60 days after the comment period ends, so the agency must process the comments very quickly.

1. The agency has a set of twenty topics that it expects the public to comment on, for example, automobile emissions, power-plant emissions, effects of pollution on lung diseases, cost of regulations, etc. How can it quickly sort the comments by the topics they discuss so that experts in each area can evaluate them? What resources would be required? How can it identify comments that mention unexpected topics?
(15 points)

This is a text categorization problem. It could be addressed with any text categorization algorithm. In class we discussed k-nearest neighbor, which would be a reasonable choice, but there are many appropriate text categorization algorithms. Key elements of a k-nn solution are lexical processing to turn a text document into a feature vector for use with k-nearest neighbor, a similarity metric (e.g., cosine correlation), and a voting method for resolving disagreements among the k neighbors. Few people described their solution in this much detail, which was disappointing, but I was lenient here. (7 points)

The necessary resources are a set of manually-labeled documents (i.e., identifying which categories each document goes into), and some estimate of the human accuracy on this task (i.e., how often to humans agree on category assignment) so that we can measure how well the algorithm is doing as compared with humans. I insisted on human accuracy, and deducted a point for not having it. (5 points)

Unexpected categories are identified by weakly-classified documents. For k-nn this would probably be a similarity/distance threshold. It would be set using training data. Documents with weak matches to all categories are clustered to reveal possible new topics. (3 points)

Some people viewed this as a clustering problem. Clustering would group the documents, but not necessarily according to the 20 topics identified by the agency, so it’s a somewhat unresponsive answer. Clustering has less of a problem with unexpected topics (because clustering has no expectations), and doesn’t need any training resources. A well-written clustering answer was worth 8 points, total.

2. The solution you proposed for question 1 works well for some topics, but it is not very accurate on other topics. What are possible causes of the low accuracy? What changes could you make to improve accuracy?
(10 points)

Possible causes include i) poor/inappropriate lexical processing (e.g., stopword lists or stemming), ii) categories that are “too similar”, iii) not enough training data for some categories, iv) k-NN parameters (e.g., k, thresholds) not well-tuned, and v) comments that mention several topics. At least two causes were necessary for full credit. (5 points)

Arguing that the training data might be unrepresentative or bad is a weak answer, and received only partial credit.

Simple changes include tuning the lexical processing (e.g., fix stopword list or stemming, use phrases), and ii) increasing the amount of training data for rare categories. Confusion between “close” topics, such as different types of emissions, might be best handled using hierarchical text categorization. If accuracy is still low it would be appropriate to consider revising the categories. It would also be good to measure human accuracy on the task, so that one has a better sense of what “reasonable” performance is. At least two changes were necessary for full credit. (5 points)

3. Most of the documents that the agency receives will be “form letters”, i.e., copies of letters written by special interest groups. Unedited form letters (exact duplicates) are easy to detect. Edited form letters (“near” duplicates; form letters with added or deleted text), are harder to detect. Design a solution for detecting edited form letters, keeping in mind that your approach must be scalable to 300,000 documents. Your solution should be capable of identifying i) the “reference” (original unedited) form letter, ii) which documents are near duplicates, and iii) which passages were changed (it is not useful to detect word-level changes in this task).
(15 points)

This is a clustering problem. Dense clusters contain exact and near duplicates. (5 points)

Identify the reference (unaltered) form letter based on frequency (there are many copies of it) and distance (it’s the one that has the most really similar neighbors). A simple vector-based distance comparison (e.g., cosine correlation) should do this easily. (3 points)

The near duplicates are the really similar neighbors to the reference copies identified above. Anything within some distance threshold can be considered a near duplicate. Set the threshold empirically. (3 points)

Identify changed passages by dividing the reference copy and the near duplicate into passages (e.g., paragraphs or sentences). Build vectors for each passage. Do passage-level comparisons between near-duplicates and the reference copy to identify passages that were added, deleted, and changed. (4 points)

I didn’t accept “do a diff between the reference and near-duplicate” because that generates word-level differences, and the problem statement is to identify changed passages. Rule-writers would refuse to use diff output – they want to see sentences or paragraphs. Using a diff and then expanding to passages was acceptable. So was string matching over passages.

Problem 2 - Web Search (7 minutes)

Maximum Marginal Relevance (MMR) was discussed as a method of improving text summarization quality, but the basic principle can be applied to other tasks. How can MMR be used to improve user satisfaction in a Web search engine? Be specific, and justify your answer.

(10 points)

Adjust the document ranking to reflect a balance of diversity/novelty and relevance, i.e., reduce redundancy in the set of documents that is returned. (10 points)

Partial credit was given for arguing that MMR could be used to create diverse multi-document summaries of a result set. It is indeed possible to use MMR in this way, although it's more work than the solution described above, and the results would be of lower quality (because you would be mixing sentences from very different kinds of documents).

Problem 3 - Natural Language Processing (35 minutes)

Let us consider the following simple context-free grammar and lexicon:

Grammar:

- (1) S ----> NP VP
- (2) NP ----> art n
- (3) NP ----> n
- (4) NP ----> NP PP
- (5) VP ----> aux VP
- (6) VP ----> v NP
- (7) VP ----> v NP PP
- (8) PP ----> p NP

Lexicon:

- the : (art 1.0)
- can: (n 0.3) (v 0.1) (aux 0.6)
- water: (n 0.8) (v 0.2)
- flowers: (n 0.9) (v 0.1)
- in: (p 1.0)
- garden: (n 0.8) (v 0.2)

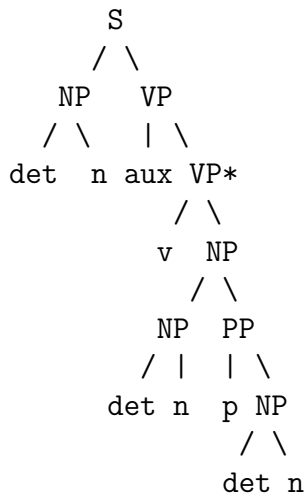
Terminals in the above grammar are the parts-of-speech (art, n, aux, v). Each lexicon entry contains a word with a list of its possible parts-of-speech. Each POS has an associated unigram probability that was derived from a large corpus of data.

We will now consider the analysis of the sentence:

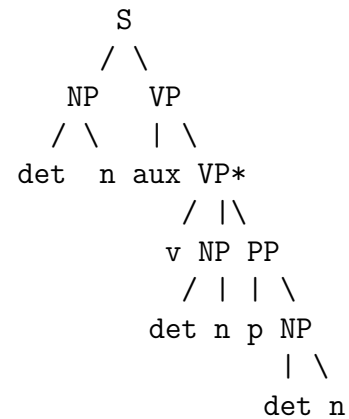
the can can water flowers in the garden.

- Show **two** possible parse trees for the above sentence according to the given grammar. Indicate clearly what POS assignment for each input word is chosen in each of the two parse trees. Select and clearly indicate which of the two trees you believe corresponds to the **correct** meaning of the sentence and briefly justify your choice.

(6 points)



Parse Tree 1



Parse Tree 2

CORRECT MEANING

- POS assignment is the same for both parse trees:

```

the  can  can  water  flowers  in  the  garden
det  n    aux  v      n        p   det  n

```

- Parse tree 1 corresponds to the correct meaning: the PP “in the garden” modifies “flowers” (it’s where the flowers are located) and not the verb “water” (where the watering occurs).

2. Are there any *local ambiguities* in the parses of the above sentence? If so, clearly identify them, and explain why they are local ambiguities.
(6 points)

The VP nodes marked with a * in the parse trees correspond to a local ambiguity: they cover the same portion of the input (“water the flowers in the garden”), analyzed as a VP in two different ways.

If we do not pack this VP, then the VP and S nodes above it will also be local ambiguities.

Technically, the ambiguity at the POS level can also be viewed as a local ambiguity, although this ambiguity is not reflected at the level of the full parse trees.

3. Assume we use a simple POS tagger that assigns the most likely tag to each word based on the unigram probabilities. What is the POS tag sequence that would be selected by the tagger? Is this the correct POS tag sequence for this input sentence? If not, clearly identify which words are tagged with an incorrect POS. Is the tag sequence chosen by the tagger parsable according to the given grammar?
(6 points)

The POS tag sequence assigned by the unigram probability tagger is:

```

the  can  can  water  flowers  in  the  garden
det  aux* aux  n*     n        p   det  n

```

It is not the correct POS sequence for the input sentence. The POS marked with * above are incorrect (the first “can” and the word “water”). The POS sequence chosen by the tagger is not parsable according to the given grammar.

4. Suggest a tag transformation rule that Transformation-based Learning (TBL) would likely learn for correcting **one** of the tagging errors made by the simple POS tagger. Recall that a TBL rule should have the form: “Replace tag X with tag Y if CONTEXT”, where CONTEXT may refer to occurrence of specific tags or words in positions prior or following the current input word. Briefly explain why you would expect TBL to learn the this rule, and how the rule would correct the tagging error.
(12 points)

There are two possible transformation rules that TBL would likely learn for correcting the above tagging errors:

- (1) For the incorrectly tagged word “can”:

Replace tag ‘AUX’ with tag ‘N’ if tag of prev word is ‘DET’

- (2) For the incorrectly tagged word “water”:

Replace tag ‘N’ with tag ‘V’ if tag of prev word is ‘AUX’ and tag of word before that is ‘N’

In both cases, we would expect TBL to learn such a rule because “N” follows “DET” and “V” follows “N AUX” in an overwhelming number of cases in the training corpus, and thus these transformations will likely significantly reduce the number of tagging errors in the tagged version of the corpus.

In the case of rule (2), it’s important to require that the preceding TWO tags be “N AUX” (rather than just “AUX”), because the sequence “AUX N” does occur correctly in many cases in the corpus (interogative questions where the sentence starts with an “AUX”). Thus, replacing the “N” with “V” wherever it’s preceded by an “AUX” may not result in an overall reduction in the number of tagging errors.

Once either of these rules is learned during training and is on the list of transformations, it will be applied at run time if and when it is the highest applicable rule on the list, and will cause the mis-tagged word to which it applies to be revised into the correct tag.

5. The version of the Earley Parsing algorithm that we studied in class could handle only a **single** possible POS for each input word, and required that the POS of each word be disambiguated prior to parsing. We would like to extend the parsing algorithm so that it may consider multiple POS for each word during parsing (finding all parses that are consistent with the grammar). Identify below which of the three main operators of the algorithm must be modified, and indicate in words how the operator(s) should be modified. Briefly explain how the extended algorithm would work.

(15 points)

Earley's Three Operators:

- **Predictor:** If state $[A \rightarrow X_1 \dots \bullet C \dots X_m, j] \in S_i$ then for every rule of the form $C \rightarrow Y_1 \dots Y_k$, add to S_i the state $[C \rightarrow \bullet Y_1 \dots Y_k, i]$

Modified Version:

NO CHANGES NECESSARY

- **Completer:** If state $[A \rightarrow X_1 \dots X_m \bullet, j] \in S_i$ then for every state in S_j of form $[B \rightarrow X_1 \dots \bullet A \dots X_k, l]$, add to S_i the state $[B \rightarrow X_1 \dots A \bullet \dots X_k, l]$

Modified Version:

NO CHANGES NECESSARY

- **Scanner:** If state $[A \rightarrow X_1 \dots \bullet a \dots X_m, j] \in S_i$ and the next input word is $x_{i+1} = a$, then add to S_{i+1} the state $[A \rightarrow X_1 \dots a \bullet \dots X_m, j]$

Modified Version:

If state $[A \rightarrow X_1 \dots \bullet a \dots X_m, j] \in S_i$ and a is a possible POS for the next input word is x_{i+1} , then add to S_{i+1} the state $[A \rightarrow X_1 \dots a \bullet \dots X_m, j]$

The modified version of scanner will now “scan” the next input word by considering ANY of the possible POS for that word, for which there is an existing item in S_i which “expects” that possible POS. The new items are added to S_{i+1} and are further considered during parsing.

6. Do the modifications you proposed have any consequences on the $O(n^3)$ worst-case time complexity of the algorithm? Briefly explain why, or why not.
(5 points)

Assuming that we perform local ambiguity packing, the modification should NOT effect the $O(n^3)$ time complexity of the algorithm. Since the number of possible POS is a constant, the consideration of multiple POS can multiply the number of items added to S_{i+1} by at most a constant factor. The consequences of this on the complexity of the other two operators is also a constant factor, thus the overall complexity should not be affected.