EDAN20
Final Examination

Pierre Nugues

October 28, 2017

The examination is worth 288 points. The distribution of points is indicated with the questions. You need 50% to have a mark of 4 and 65% to have a 5.

1 Closed Book Part: Questions

In this part, no document is allowed. It is worth 130 points.

Chapter 1. Cite three applications that use natural language processing to some extent. 3 points

Chapter 1. Annotate each word of the sentence:

Elon Musk unveils some new twists on his plan for Mars[1]

with its part of speech. You will use parts of speech you learned at school: common noun, proper noun, verb, adjective, preposition. Use the determiner part of speech to replace the more traditional terms: indefinite adjectives and possessive adjectives (sometimes called possessive pronouns) 3 points

Chapter 1. Draw the graph of the sentence *Elon Musk unveils some new twists* using dependency relations. You will notably identify the subject and the object. 4 points

Chapter 1. Identify the proper nouns (or named entities) in the sentence:

Elon Musk unveils some new twists on his plan for Mars

and describe what entity linking is on this example and why it can be difficult. 8 points

Chapter 1. Represent the sentence *Elon Musk unveils some new twists on his plan for Mars* with a predicate–argument structure. 4 points

Chapter 2. Describe what a concordance is and give all the **case-sensitive** concordances of the string *kaolin* with one word before and one word after in the text below: 3 points

[1] Retrieved on September 29, 2017 from `www.economist.com`

Dom i Kaolinmålet: Ingen brytning av den vita leran i Billinge

Svenska Kaolin får nej på sin ansökan som berör Eslöv, Svalöv och Klippan. Enligt Mark- och miljödomstolen är kaolinbrytningen i Billinge inte förenlig med miljöbalken.

I somras kompletterade Svenska Kaolin sin ansökan om kaolinbrytning med en omarbetad miljökonsekvensbeskrivning. Mark- och miljödomstolens godkänner den uppdaterade versionen och anser att den duger som grundval för att pröva ansökan. Men den samlade bedömningen är fortfarande att kaolinbrytningen inte är förenlig med miljöbalkens krav. Därför blir det inget miljötillstånd.

Kaolinfyndigheten vid Billinge fälad i norra delen av Eslövs kommun upptäcktes på 1980-talet. En första ansökan om att bryta kaolin här lämnades in 1994. Svenska Kaolin AB har hållit fast vid planerna trots tidigare avslag och ansökte på nytt 2014. Ansökan gäller brytning av högst 20 miljoner ton mineralråvara under 20 år.

Sydsvenskan.se, Retrieved October 24, 2017. Author Birgitta Johansson

Please note that the hyphens, as in *kaolinbryt-ning*, were added by the Latex text processor and should not be counted as real characters.

Chapter 2. Identify what the regular expressions in the list below match in the text above (identify all the matches and just write one word before and after, or write no match if there is no match): 15 points

List of case-sensitive regular expressions:

1. `Kaolin`
2. `Kaolin\s`
3. `kaolin\p{L}*`
4. `kaolin\p{L}?`
5. `kaolin\p{L}+`
6. `kaolin\p{L}{10}`
7. `kaolin\P{L}`
8. `[Kk]aolin\s`
9. `^\p{L}` with the multiline option
10. `[A-Z](.)(.)\2\1`

Chapter 2. What will be the output of the command: 3 points

```
tr -cs '0-9' '\n' <text
```

when applied to the text above (*Dom i Kaolinmålet: Ingen brytning av den vita leran i Billinge*).

Chapter 2. Write a simple regular expression that matches all the words of a text consisting of Latin unaccented characters only: lower and uppercase. 1 point

You will either list all the letters of the Latin alphabet explicitly or use a range.

Chapter 2. Write a regular expression that matches all the words in a text that include at least one of the Swedish letters: åäö. You will assume that the text is in lowercase. 8 points

You will proceed in two steps:

- First, write a regular expression, where you match all the words that include one Swedish letter. In the phrase:

  svenska kaolin får nej på sin ansökan,

  you will match the words: *får*, *på*, and *ansökan*. In this first step, your regular expression will probably also match word fragments in words with more than one Swedish letter.
- In a second step, modify the regular expression to repeat the match so that you detect the words with multiple Swedish letters properly. In the sentence:

  därför blir det inget miljötillstånd,

  this will correspond to *därför* and *miljötillstånd*.

With the program:

```
regex = ?? # your regex
string1 = 'Svenska Kaolin får nej på sin ansökan'.lower()
string2 = 'Därför blir det inget miljötillstånd'.lower()

# select string1 or string2
matches = [match.group() for match in re.finditer(regex, string)]
print(matches)
```

your `regex` should produce :

```
['får', 'på', 'ansökan']
['därför', 'miljötillstånd']
```

Chapter 2. You will write a manual spell checker for Swedish with the word *koalin* (notice the typo). Spell checkers typically use four edit operations: deletions, transpositions, substitutions, and insertion. 8 points

1. Generate all the words resulting from a character deletion for this word;
2. Generate all the words resulting from a transposition;
3. Generate all the words resulting from a substitution; You will use an alphabet limited to one single symbol, λ, to carry out the substitutions;

4. Generate all the words resulting from an insertion; You will an alphabet limited to one single symbol, λ, to carry out the insertions.

Chapter 2. For the word above, *koalin*, how many candidates do you obtain, supposing your alphabet has 26 unaccented letters and three Swedish letters? 4 points

How would you select and rank the candidates?

Chapter 3. What do the `\p{Greek}+`, `\p{Latin}+`, and `\p{Hiragana}+` Unicode regular expressions match? Please, provide one answer for each regular expression. 3 points

Chapter 4. Describe what a supervised classifier is. What is the typical notation of input matrix of a classifier (predictors) and output vector (response). Give the name of two classifiers: The one you used for the assignments and a second one. 4 points

Chapter 5. Give the probabilistic language model of the sentence:

Därför blir det inget miljötillstånd

using no n-gram approximation. You will ignore possible start and end of sentence symbols. 2 points

Chapter 5. Using a unigram approximation, give the probabilistic language model of the sentence:

Därför blir det inget miljötillstånd

You will ignore possible start and end of sentence symbols. 2 points

Chapter 5. Using a bigram approximation, give the probabilistic model of the sentence:

Därför blir det inget miljötillstånd

You should have exactly the same number of terms as in the previous question. You can include a start of sentence symbol or not. 2 points

Chapter 5. Using the counts in Table 1, you will compute the probability of the sentence:

Därför blir det inget miljötillstånd

using a unigram and a bigram approximations. You will use fractions to represent the terms in the product and you will not try to reduce them, *i.e.* you will write $\frac{1}{3}$ and not 0.33. 8 points

You may need the total number of words in the `.se` web domain. You will interpolate it from the count of words in the Selma corpus that totals about 1 million words with 22,000 occurrences of *det*. You will justify your interpolation with regard to Table 1.

Chapter 5. Describe the Laplace method to estimate a bigram probability and apply it to the sentence: 2 points

Table 1: Word and bigram counts retrieved from Google.com on October 25, 2017 with the search limited to the Swedish domain (`site:se`).

Words	**Unigram counts**	**Bigrams**	**Bigram counts**
därför	6 270 000		
blir	6 370 000	därför blir	629 000
det	247 000 000	blir det	3 900 000
inget	5 670 000	det inget	1 740 000
miljötillstånd	54 700	inget miljötillstånd	151
miljötilstånd	23	inget miljötilstånd	0

Därför blir det inget miljötilstånd

For the exchange students: Please notice the typo in *miljötilstånd*. The correct spelling is: *miljötillstånd*

Chapter 5. Describe the simple back off method to estimate a bigram probability and apply it to the sentence:

Därför blir det inget miljötilstånd

You will use the same back off as in the labs. 2 points

Please notice the typo in *miljötilstånd*.

Chapter 5. Supposing the word *miljötilstånd* (please note the typo) is not in your dictionary, how can you use unigram and then bigram probabilities to rank the candidates produced by edit operations. 2 points

Justify in which cases bigram probabilities can produce better results than unigrams. Think of the example I gave with the word *acress* during the lectures.

Chapter 8. In this exercise, you will describe how to build a simple part-of-speech tagger using a linear classifier. You will use the example in Table 2.

1. Describe what the training step is and what you would use as data to carry it out (how would you obtain it?). Describe how you would build the input matrix (predictors) and output (response); 3 points
2. Build this input matrix using features consisting of the current word, the previous word, and the next word. You will write the five first lines of the matrix and output vector. You will only use symbols: words and parts of speech for this; 3 points
3. Describe what the test or evaluation step is and what you would use as data; 3 points
4. Name the two scikit-learn methods you would use to carry out these two steps (training and prediction); 2 points
5. The training step only accepts numerical matrices. Describe what a vectorization (or one-hot encoding) is and how you would transform a matrix of symbols (strings) into a matrix of numbers. Name the scikit-learn methods for it. 4 points

6. Name the scikit-learn class for this. 1 point

Table 2: An excerpt from the Penn Treebank with the part-of-speech annotation of the sentence: *The luxury auto maker last year sold 1,214 cars in the U.S.*

Index	Words	Parts of speech
1	The	DT
2	luxury	NN
3	auto	NN
4	maker	NN
5	last	JJ
6	year	NN
7	sold	VBD
8	1,214	CD
9	cars	NNS
10	in	IN
11	the	DT
12	U.S.	NNP

Chapter 13. Represent graphically the dependency graphs of the two following sentences:

Jag tror på det.

and

Vad beror detta på?

in Tables 3 and 4. 5 points

What is the main difference between the two graphs?

Table 3: Dependency analysis of *Jag tror på det.*

Index	Words	Head
1	Jag	2
2	tror	0
3	på	2
4	det	3
5	.	2

Chapter 13. Define the four actions – shift, reduce, left-arc, and right-arc – used in Nivre's parser to parse a dependency graph and create arcs. 8 points

Chapter 13. Using gold-standard parsing and Nivre's parser, parse the sentence:

Jag tror på det.

Table 4: Dependency analysis of *Vad beror detta på?*

Index	**Words**	**Head**
1	Vad	4
2	beror	0
3	detta	2
4	på	2
5	?	2

and give the sequence of actions to complete the parse. You will represent graphically the stack and the queue with the words they contain. 5 points

Chapter 13. Using gold-standard parsing and Nivre's parser, try to parse the sentence

Vad beror detta på?

and give the sequence of actions until you fail. Explain why the parser fails. 5 points

2 Problem

In this part, documents are allowed. It is worth 158 points.

Transition-based parsing is a technique to parse dependencies. It uses a queue of input words, a stack, and a set of actions to apply to these data structures. As course assignments, you have implemented such a dependency parser.

In this part of the examination, you will design and program a constituent parser using shift and reduce actions and the same data structure. You will follow the paper by Sagae and Lavie (2005) for the implementation. This parser can only create binary trees: Trees where nodes have at most two children. In the last exercise, you will write a program to binarize any kind of tree.

The paper by Sagae and Lavie (2005) has started a line of similar systems that improved the parsing performance over the years. Liu and Zhang (2017) is a recent example that defines the current state of the art. This second paper is only provided for information and will not be used for programming.

As programming language, you can use Python (strongly preferred), Java, Perl, or Prolog. You will focus on the program structure and not on the syntactic details. You can ignore the Python modules, Java packages or imports for instance.

2.1 Shift-Reduce Parsing

2.1.1 Analyzing the Papers

1. Read the *Abstract* and *Introduction* sections of Sagae and Lavie (2005) and the *Abstract* of Liu and Zhang (2017) and describe what they claim are the advantages of shift-reduce parsing over other techniques[2]. 8 points

2. Find in both texts the parser actions and name them. You just need to name these actions, you will analyze them later. 4 points

3. The parser by Liu and Zhang (2017) has two more actions than that of Sagae and Lavie (2005). Name these actions. 2 points

2.1.2 Manual Parsing

In this section, you will parse manually the sentences *The waiter brought the meal* and *The waiter slept*. Figures 1 and 2 show the parse trees you should obtain.

To understand the algorithm proposed by Sagae and Lavie (2005), read first the *Algorithm Outline* section.

1. During the lectures and in the textbook (Nugues, 2014), we have studied shift-reduce parsing. What are the two main differences between the algorithm in Sagae and Lavie (2005, Sect. Algorithm Outline) and the shift-reduce parsing sequence from the textbook and the course slides shown in Table 5. 4 points

   Think of the data in the input sequence and the types of reduce actions.

[2]The phrase *transition-based parsing* is a synonym of shift-reduce parsing.

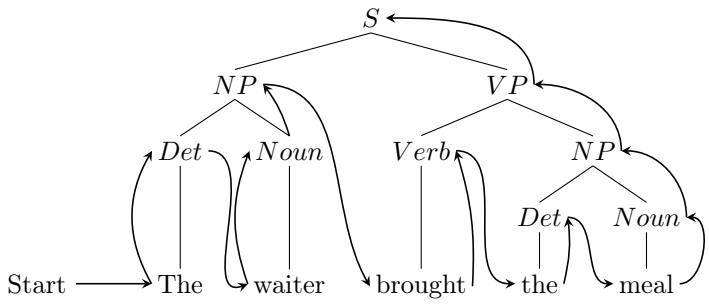

Figure 1: Bottom-up parsing. The parser starts with the words and builds the syntactic structure up to the top node

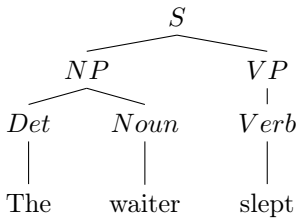

Figure 2: Parse tree of *The waiter slept*

2. Sagae and Lavie (2005) used two types of reduce actions, `reduce_unary(X)` (abridged in `ru(X)`) and `reduce_binary(X)` (abridged `rb(X)`). Here, you will ignore the left and right directions of the head. 8 points

   Replace the reduce actions in Table 5 by `ru(X)` or `rb(X)` with the correct value of `X`.

   To help you start, at iteration 2, Reduce should be replaced with `ru(det)` and at iteration 5, Reduce should be `rb(np)`.

3. Using the parse tree in Fig. 2, determine the action sequence needed to parse the sentence *The waiter slept*. You will present your results as in Table 5 with the actions `ru(X)` and `rb(X)`. 6 points

2.1.3 Building a Parse Tree From an Action Sequence

You will now write a parser that takes a sentence and an action sequence as input and produces a parse tree as output.

- You will store the queue, `queue`, and the stack, `stack`, as lists, for instance:

  ```
  ['the', 'waiter', 'brought', 'the', 'meal']
  ```

- You will store the action sequences, `transitions`, as lists, for instance:

  ```
  [['sh'], ['ru', 'det'], ['sh'], ['ru', 'noun'], ['rb', 'np']]
  ```

Table 5: Steps of the shift–reduce algorithm to parse *the waiter brought the meal*. The Python quotes have been omitted to save space. After Nugues (2014)

It.	Stack	S/R	Word list
0			[the, waiter, brought, the, meal]
1	[the]	Shift	[waiter, brought, the, meal]
2	[det]	Reduce	[waiter, brought, the, meal]
3	[det, waiter]	Shift	[brought, the, meal]
4	[det, noun]	Reduce	[brought, the, meal]
5	[np]	Reduce	[brought, the, meal]
6	[np, brought]	Shift	[the, meal]
7	[np, v]	Reduce	[the, meal]
8	[np, v, the]	Shift	[meal]
9	[np, v, det]	Reduce	[meal]
10	[np, v, det, meal]	Shift	[]
11	[np, v, det, n]	Reduce	[]
12	[np, v, np]	Reduce	[]
13	[np, vp]	Reduce	[]
14	[s]	Reduce	[]

to represent the sequence: `[sh, ru(det), sh, ru(noun), rb(np)]`.

- Your parser will output a tree that you will represent as Python recursive lists, lists within lists, such as

```
parse_tree = \
    ['s',
     ['np', ['det', 'the'], ['noun', 'waiter']],
     ['vp',
      ['verb', 'brought'],
      ['np', ['det', 'the'], ['noun', 'meal']]]]
```

for the parse tree in Fig. 1. The first item of a list is the node label, `node_label` and the rest, the children, `children`, all this recursive.

We have:

```
print(parse_tree[0]) # the node label
s
```

and

```
print(parse_tree[1:]) # the tree
[['np', ['det', 'the'], ['noun', 'waiter']],
 ['vp',
   ['verb', 'brought'],
   ['np', ['det', 'the'], ['noun', 'meal']]]]
```

1. Given the phrase *The waiter* and the action sequence:

```
[['sh'], ['ru', 'det'], ['sh'], ['ru', 'noun'], ['rb', 'np']]
```

draw the stack and the queue for each action (five in total), as well as the resulting parse trees on the top of the stack after each reduce action (three parse trees). 8 points

You will need to complement the data structure storing the items of the stack so that each reduce builds a partial tree. To help you, after the first reduce, the top of the stack should be `[['det', 'the']]` and at the end of the parse, the top of the stack must be:

```
[['np', ['det', 'the'], ['noun', 'waiter']]]
```

2. Program the `shift(stack, queue)` function. 10 points
3. Program the `reduce_unary(stack, X)` function. 12 points
4. Program the `reduce_binary(stack, X)` function. 16 points
5. Program a `'__main__'` block that processes the input and produces the output. 10 points

2.1.4 Gold-Standard Parsing

Normally, when you parse a sentence, you do not know the action sequence. In the assignments, sa well as in in Sagae and Lavie (2005), you determined the actions using a classifier. To train such a classifier, you associated a dependency graph with an action sequence. This is called gold-standard parsing.

You will now program this function, `oracle()`, so that given a constituent parse tree as input, the oracle outputs an action sequence, `transitions`.

1. Using the tree in Fig. 2, annotate each node and leave with the action corresponding to its creation or move in the stack, either `rb(X)`, `ru(X)`, or `sh`. (You will redraw the tree on your exam paper and mark each node with `rb(X)`, `ru(X)`, or `sh`). 8 points
2. Compare this annotated tree with the sequence you obtained in Sect. 2.1.2. How can you obtain the action sequence from a tree traversal? The possibilities are preorder, inorder, or postorder. 6 points
3. Write a gold-standard parser that takes a parse tree in the form of list as input and outputs an action sequence. 18 points

   You can use the code skeleton below and complete the `FILL IN` parts:

```
transitions = []
def oracle(parse_tree):
    """
    The oracle predicts the transition sequence from a parse tree
    """
    if len(parse_tree) == 0:
        return
    if type(parse_tree[0]) == str and type(parse_tree[1]) == str:
        # We have reached the words
```

```
        transitions.append('sh')
        transitions.append(['ru', parse_tree[0]])
        return
    if len(parse_tree[1:]) == 1:
        FILL IN

    if len(parse_tree[1:]) == 2:
        FILL IN

    if len(parse_tree[1:]) > 2:
        print('Not a binary tree')
        return
```

Given the input parse tree:

```
tree = ['S',
             ['NP', ['DT', 'the'], ['NN', 'waiter']],
             ['VP',
              ['VBD', 'brought'],
              ['NP', ['DT', 'the'], ['NN', 'meal']]]]
```

your oracle should output the following transition sequence:

```
['sh', ['ru', ['DT', 'the']], 'sh', ['ru', ['NN', 'waiter']],
['rb', ['NP', ['DT', 'NN']]], 'sh', ['ru', ['VBD', 'brought']],
'sh', ['ru', ['DT', 'the']], 'sh', ['ru', ['NN', 'meal']],
['rb', ['NP', ['DT', 'NN']]], ['rb', ['VP', ['VBD', 'NP']]],
['rb', ['S', ['NP', 'VP']]]]
```

2.1.5 Training a Classifier

Now that you have the programs, describe the method to train a classifier and build a parser? What could be the features? 10 points

2.2 Binarizing Trees

Sagae and Lavie (2005)'s parser only accepts binary trees. In most treebanks, nodes can have more than two children. This means that we have to binarize the trees before applying a gold-standard parsing. In addition, as the parser only produces binary trees, this also means that we will have to "unbinarize" the parsed sentences.

Examine Figure 1 in Sagae and Lavie (2005), where you will ignore the phrase heads. The tree is said to be right-binarized because when a node has more than two children, they created recursively subnodes with the children to the right, for example, `(a b c)` is transformed into `(a (b c))` and `(a b c e)` into `(a (b (c e)))`.

In this exercise, you will write a right binarization program, where the input will be a parse tree: `parse_tree` consisting of recursive lists, where we will have:

```
node_label = parse_tree[0]
children = parse_tree[1:]
```

and the output will be a binarized tree with the same structure and `len(children) < 2` for all the nodes in the tree.

To make it easier to find a solution, we can decompose the program in the four steps below. Should you want not to follow these suggested steps, please feel free to do so.

1. Write a function, `binarize()`, that traverses a parse tree, possibly non-binary. The function will be recursive and do nothing else than traverse the tree; 6 points

2. Modify `binarize()` so that it prints the nodes that have more than two children. You will need a condition of the type: 6 points

   ```
   if children and len(children) > 2:
     FILL IN
   ```

3. In the conditional block where a node has more than two children, create a new node, `new_node` that brackets the nodes to the right and replace the right children after the first one with this new node. 6 points

   For instance, in the tree

   ```
   ['NP', ['DT', 'the'], ['JJ', 'diligent'], ['NN', 'waiter']]
   ```

   you would have

   ```
   new_node = ['NP', ['JJ', 'diligent'], ['NN', 'waiter']]
   ```

   where we reuse the node label, here `NP`, and we woud binarize the tree as:

   ```
   ['NP',
     ['DT', 'the'],
     ['NP',
       ['JJ', 'diligent'], ['NN', 'waiter']]]
   ```

4. Finally, modify the recursive call to `binarize()` so that you build complete binary trees and your program is complete. 10 points

References

Liu, J. and Zhang, Y. (2017). Shift-reduce constituent parsing with neural lookahead features. *Transactions of the Association for Computational Linguistics*, 5:45–48.

Nugues, P. M. (2014). *Language Processing with Perl and Prolog. Theories, Implementation, and Application.* Springer Verlag, Berlin Heidelberg New York, second edition.

Sagae, K. and Lavie, A. (2005). A classifier-based parser with linear run-time complexity. In *Proceedings of the Ninth International Workshop on Parsing Technology*, Parsing '05, pages 125–132, Stroudsburg, PA, USA. Association for Computational Linguistics.