# EDAN20
# Final Examination

Pierre Nugues

October 27, 2015

The examination is worth 221 points. The distribution of points is indicated with the questions. You need 55% to have a mark of 4 and 70% to have a 5.

## 1 Closed Book Part: Questions

**In this part, no document is allowed. It is worth 107 points.**

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

**Chapter 1.** Annotate each word of the sentence *The boy hit the ball* with its part of speech. You will use parts of speech you learned at school. 3 points

**Chapter 1.** Draw the graph of the sentence *The boy hit the ball* 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 *Aristotle wrote the Organon* and describe what entity linking is on this example. 8 points

**Chapter 1.** Peedy is a 3D animated character designed by Microsoft that responds to spoken commands to change discs. Figure 1 shows it architecture. Describe, notably in terms of input and output, the modules: whisper, names, NLP, semantics, and dialogue. 5 points

**Chapter 2.** Describe what a concordance is and give all the case-sensitive concordances of the string *Vårfru*: 3 points

> Här är planen för skolor i Centrum-Väster
> Förändringar för Vårfruskolan och Apelskolan. En ny skola och kanske förskola på Stenkrossen. Detta är kärnan i del 1 av planen för skolor och förskolor i Centrum-Väster.
>
> Denna första del av områdeslokalplanen åker nu ut på remiss. Delplanen berör förskola och skola upp till tredje klass.
>
> De förändringar som föreslås gäller Vårfruskolan, Apelskolan och nybyggnation på Stenkrossen.

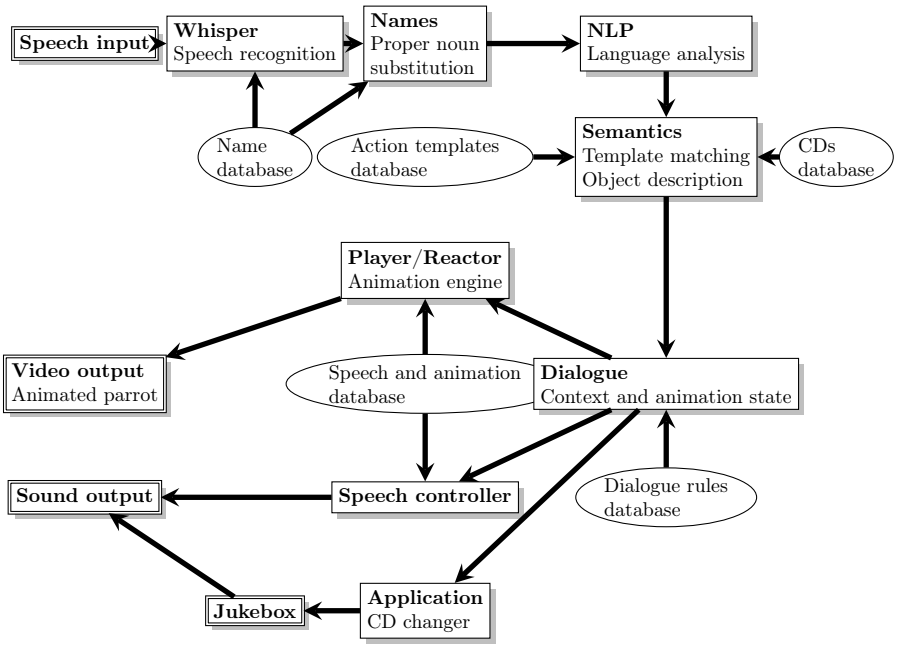


Figure 1: Architecture of the Persona conversational assistant. After Ball et al. (1997)

> Två alternativ ges i denna plan för 2016-2020. Det alternativ som barn- och skolförvaltningen förordar innebär att:
> — Montessorigrundskolan Apelskolan (som nu rymmer förskoleklass och årskurs 1-6) blir en skola för förskoleklass och årskurs 1-3.
> — En ny förskola och F-3-skola byggs på Stenkrossen.
> — F-3-klasserna på Vårfruskolan flyttas till denna nya enhet på Stenkrossen.
> — Vårfruskolan blir en skola för klasserna 4-6, i stället för att som nu vara en F-6-skola.
>
> *Sydsvenskan.se*, Retrieved October 20, 2015. Author Alexander Agrell

**Chapter 2.** Identify what the regular expressions in the list below will match in the text above (identify all the matches or write no match if there is no match): 10 points

List of regular expressions:

1. `för+skole`
2. `(för)+skole`
3. `(för)+skol(a|e)`
4. `(för)+skol[a-z]`
5. `(för)+skol.`

6. `(för)+skol[a-z]{2}`
7. `(för)+skol\w+`
8. `(för)+skol\w{2,}`
9. `\d+-\d+`
10. `([ao])[a-z]\1`

**Chapter 2.** Write a regular expression that matches all the words, where the stem *skol* occurs. Be sure to be Unicode compatible. 2 points

**Chapter 2.** Spell checkers identify words that are not in their dictionaries and suggest corrections to unknown words. They typically use four operations: deletions, insertion, substitutions, and transpositions. 12 points

Given the string *acress* not in the dictionary:

1. How many deletions can you apply to this string? List them all.
2. How many transpositions can you apply to this string? List them all.
3. How many insertions can you apply to this string? List two of them.
4. How many substitutions can you apply to this string? List two of them.

You will only consider lowercase unaccented characters in your counts.

**Chapter 3.** Describe what a Unicode block is and give an example of a block. 2 points

**Chapter 4.** Describe what a decision tree is. 2 points

**Chapter 4.** Describe very succinctly how a decision tree can be induced from a data set. Use an intuitive formulation. 2 points

**Chapter 5.** What does this Unix command do? 3 points

```
tr -cs 'A-Za-z' '\n' <file.txt
```

**Chapter 5.** What do these Unix commands do? 3 points

```
sort <file.txt | uniq -c
```

**Chapter 5.** Many interactive search systems provide a word prediction and word completion module that given the start of a phrase suggests possible following words. Figure 2 shows an example of such an interaction, where a user has typed the word *Justin* and where the search system suggests *gatlin*, *bieber*, *timberlake*, *rose*, and *williams* as possible next words.

1. Language models are a model of word prediction. Define what a language model is and how to approximate it in practice. 4 points
2. Language models use word $N$-grams. Identify all the unigrams, bigrams, and trigrams in the text below excerpted from *Nils Holgersson* by Selma Lagerlöf: 3 points


Figure 2: Term search with a popular Swedish news site. A user has typed the word *Justin* in the right-hand input box and the system shows a list of suggested next words in bold characters below it. Retrieved on October 20, 2015 from the `http://www.dn.se/` site.

> Det var en gång en pojke. Han var så där en fjorton år gammal, lång och ranglig och linhårig.

You will ignore the uppercase/lowercase difference and the punctuation.

3. Write a probabilistic model of the first sentence above using unigrams and bigrams. You will not try to compute these probabilities. 3 points
4. Describe how you would estimate a unigram probability. 2 points
5. Describe how you would estimate a bigram probability. 3 points
6. Finally, describe you would carry out the bigram ranking in Fig. 2. 3 points

**Chapter 5.** Describe what is an unseen bigram (or $N$-gram). 1 point

**Chapter 5.** Describe what is the simple back-off method to cope with unseen bigrams. 2 points

**Chapter 5.** Give the definition of the mutual information association measure. 2 points

**Chapter 7.** Using parts of speech from the set: {determiner, adjective, noun, pronoun, modal, and verb}, and the sentences: *The can rusted* and *The boy can swim*, 6 points

1. Annotate each word of the two sentences with its correct part of speech;
2. In your opinion, what is the most frequent part of speech of the word *can*?
3. Reannotate each word of the two sentences with the most frequent part of speech of each word;
4. Imagine a rule taking into account the context to disambiguate the word *can*.

| Words | POS | Groups |
|---|---|---|
| Rockwell | NNP | |
| said | VBD | |
| the | DT | |
| agreement | NN | |
| calls | VBZ | |
| for | IN | |
| it | PRP | |
| to | TO | |
| supply | VB | |
| 200 | CD | |
| additional | JJ | |
| so-called | JJ | |
| shipsets | NNS | |
| for | IN | |
| the | DT | |
| planes | NNS | |
| . | . | |

Table 1: An excerpt of the CoNLL 2000 dataset (Tjong Kim Sang and Buchholz, 2000)

**Chapter 10.** Table 1 shows an excerpt of the CoNLL 2000 dataset (Tjong Kim Sang and Buchholz, 2000). Using the IOB-2 scheme, consisting of the begin, inside, and outside tags, annotate the sentence words with the noun groups (NP) and verb groups (VP). You will annotate the remaining words with O. 4 points

**Chapter 11.** Draw the dependency graph of the sentence *Aristotle wrote the Organon* 2 points

**Chapter 11.** Extract a triple consisting of the subject, the verb, and the object from the sentence *Aristotle wrote the Organon* 1 points

**Chapter 13.** Nivre's parser uses four parsing actions: left-arc, right-arc, reduce, and shift. Define these four actions. 4 points

**Chapter 16.** Table 2 shows two sentences from the CoNLL 2011 corpus (Pradhan et al., 2011). The Chain column contains the coreference chains of entities. Give all the coreference chains appearing in these two sentences as well as their list of mentions. You will use the notation:

$$\begin{aligned} & CorefChain(0) = \{Mention1, Mention2, ...\}, \\ & ... \\ & CorefChain(x) = \{Mention1, Mention2, ...\} \\ & ... \end{aligned}$$

5 points

Table 2: Simplified annotation of two sentences in the CoNLL 2011 corpus. After Pradhan et al. (2011).

| Document | Inx | Word | POS | Parse bit | Type | Chain |
|---|---|---|---|---|---|---|
| wsj_0771 | 0 | “ | “ | (TOP(S(S* | * | - |
| wsj_0771 | 1 | Vandenberg | NNP | (NP* | (PERSON) | (8\|(0) |
| wsj_0771 | 2 | and | CC | * | * | - |
| wsj_0771 | 3 | Rayburn | NNP | *) | (PERSON) | (23)\|8) |
| wsj_0771 | 4 | are | VBP | (VP* | * | - |
| wsj_0771 | 5 | heroes | NNS | (NP(NP*) | * | - |
| wsj_0771 | 6 | of | IN | (PP* | * | - |
| wsj_0771 | 7 | mine | NN | (NP*)))) | * | (15) |
| wsj_0771 | 8 | , | , | * | * | - |
| wsj_0771 | 9 | ” | ” | *) | * | - |
| wsj_0771 | 10 | Mr. | NNP | (NP* | * | (15 |
| wsj_0771 | 11 | Boren | NNP | *) | (PERSON) | 15) |
| wsj_0771 | 12 | says | VBZ | (VP* | * | - |
| wsj_0771 | 13 | , | , | * | * | - |
| wsj_0771 | 14 | referring | VBG | (S(VP* | * | - |
| wsj_0771 | 15 | as | RB | (ADVP* | * | - |
| wsj_0771 | 16 | well | RB | *) | * | - |
| wsj_0771 | 17 | to | IN | (PP* | * | - |
| wsj_0771 | 18 | Sam | NNP | (NP(NP* | (PERSON* | (23 |
| wsj_0771 | 19 | Rayburn | NNP | *) | *) | - |
| wsj_0771 | 20 | , | , | * | * | - |
| wsj_0771 | 21 | the | DT | (NP(NP* | * | - |
| wsj_0771 | 22 | Democratic | JJ | * | (NORP) | - |
| wsj_0771 | 23 | House | NNP | * | (ORG) | - |
| wsj_0771 | 24 | speaker | NN | *) | * | - |
| wsj_0771 | 25 | who | WP | (SBAR(WHNP*) | * | - |
| wsj_0771 | 26 | cooperated | VBD | (S(VP* | * | - |
| wsj_0771 | 27 | with | IN | (PP* | * | - |
| wsj_0771 | 28 | President | NNP | (NP* | * | - |
| wsj_0771 | 29 | Eisenhower | NNP | *)))))))))) | (PERSON) | 23) |
| wsj_0771 | 30 | . | . | *)) | * | - |
| | | | | | | |
| wsj_0771 | 0 | “ | “ | (TOP(S* | * | - |
| wsj_0771 | 1 | They | PRP | (NP*) | * | (8) |
| wsj_0771 | 2 | allowed | VBD | (VP* | * | - |
| wsj_0771 | 3 | this | DT | (S(NP* | * | (6 |
| wsj_0771 | 4 | country | NN | *) | * | 6) |
| wsj_0771 | 5 | to | TO | (VP* | * | - |
| wsj_0771 | 6 | be | VB | (VP* | * | (16) |
| wsj_0771 | 7 | credible | JJ | (ADJP*))))) | * | - |
| wsj_0771 | 8 | . | . | *)) | * | - |

# 2 Problem

**In this part, documents are allowed. It is worth 114 points.**

In this part, you will program an elementary question–answering system, inspired by IBM Watson (Ferrucci, 2012). IBM Watson proved to be better than any human to answer questions such as this one from the *Jeopardy* quiz game:

> RECENT HISTORY: President under whom the U.S. gave full recognition to Communist China.
>
> (Answer: Jimmy Carter)

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

## 2.1 Understanding a Module of IBM Watson

Before you write a program, read the Sections: Introduction, Searching unstructured resources, and Generating candidates from search results (only the first paragraph) from the article *Finding needles in the haystack: Search and candidate generation* (Chu-Carroll et al., 2012). This paper describes the techniques used by IBM Watson to find candidate answers to a question. It was written by the designers of this system.

1. Summarize in about 10-15 lines how IBM Watson extracts paragraphs relevant to a question from an encyclopedia. 8 points
2. Summarize in about 5-10 lines the baseline technique IBM Watson uses to extract candidate answers to a question from the passages. 6 points

## 2.2 Passage Retrieval

The real IBM Watson is complex and we propose here a simplified architecture of a typical question–answering system (Figure 3). It consists of three main modules, where

- The first module parses the question and classifies it;
- The second module ranks documents that it extracts from a document store;
- The third module extracts the answer from a set of documents extracted by the second module.

In this section, we will focus on the second module, where the document store is a large set of documents such as all the articles in Wikipedia. You will represent the documents using the vector space model.

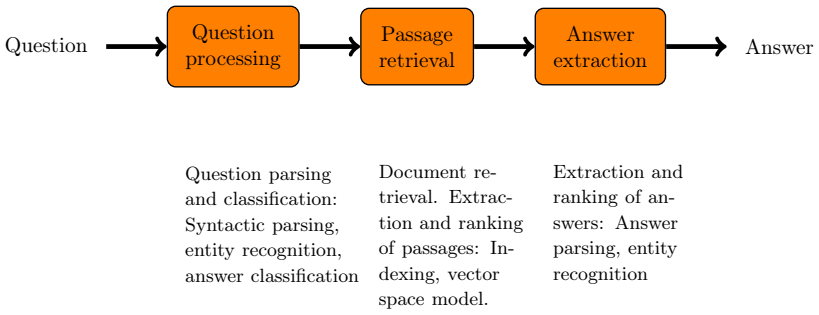


Figure 3: Overall architecture of a question–answering system

### 2.2.1 Representing the Documents

1. The vector space model represents a document by a vector, where the dimensions are the words and the coordinates are computed with the $TF \cdot IDF$ formula. $Tf_i$ is the relative frequency of term $i$ in the document and $idf_i$ is the inverted document frequency computed with this formula:

$$idf_i = \log(\frac{N}{n_i}),$$

where $N$ is the total number of documents and $n_i$ is the number of documents in the collection, where term $i$ occurs at least once.

Using the $TF \cdot IDF$ model, represent manually the two short documents below:

**D1.txt:** Chrysler plans new investments in Canada and in Latin America.

**D2.txt:** Chrysler plans major investments in Canada and in Mexico.

with two vectors.

You will set all the words in lowercase; you will sort them in alphabetical order; and you will ignore the punctuation. Compute manually the term frequency of each word in a document and compute the document frequency of each word. Compute manually $TF \cdot IDF$. You will use $\log_2$ to compute the logarithms. Finally, write the two vectors representing D1.txt and D2.txt. 8 points

2. Write a program that reads a document, sets the text in lowercase, tokenizes its content, counts the words, and outputs the words in alphabetical order with their relative frequencies in the document. You will ignore non-alphabetical symbols. The screen output for `D1.txt` should look like to:

```
america 0.1
and 0.1
canada 0.1
chrysler 0.1
in 0.2
...
```

Break down your code into `tokenize` and `count` functions. In Java, use Map structures and, in Python, dictionaries to count the words, where the keys will be the words and the values, the counts. 12 points

3. Extend your program so that it reads all the documents with a `.txt` suffix in a folder, for each document, extracts the relative frequencies, and stores them in a Java map or Python dictionary. You will call this map: `collection`. For each (key, value) pair, the key will be the document name and the value will be the map of word frequencies obtained in the previous question. In our small collection, `collection` will have two keys: `D1` and `D2`.

   If you do not know the name of the function to read a folder, invent a plausible one with a short description in terms of input output. 6 points

4. Write a function that computes the document frequency of the terms and outputs the terms in alphabetical order. The document frequency of a term is the number of documents in the collection that contain this word. Store these document frequencies in a map that you will call `df` and output it (on the screen). For our collection of two documents, the output should look like: 6 points

```
america 1
and 2
canada 2
chrysler 2
in 2
investments 2
latin 1
...
```

5. Finally, for each document, compute the $TF \cdot IDF$ of each word. You will store the results in `collection` map. 6 points

### 2.2.2 Cosine Similarity

Now that we have represented the documents using the vector space model, we can compute the cosine similarity between a question and a document. The similarity will range from 0 (totally different documents) to 1 (very similar documents). The cosine similarity of two vectors $\vec{u}$ and $\vec{v}$ is defined as:

$$\frac{\vec{u} \cdot \vec{v}}{||\vec{u}|| \cdot ||\vec{v}||}$$

1. Implement a `dot` function that computes the dot product of two vectors. The two vectors will be stored as Map structures in Java or dictionaries in Python. You will traverse all the keys of one vector, and for each of these keys, you will multiply the values obtained from the two vectors. You will return the sum. 3 points

2. Standardize the documents in `collection` with their respective norm: Divide all the parameters of each document vector by its norm. The norm of a vector $\vec{u}$ is defined by: 3 points

$$||\vec{u}|| = \sqrt{\vec{u} \cdot \vec{u}}$$

3. Write a program to represent the question as a vector. You will do exactly like for the documents: You will use a vector that you will store in a map. You will standardize the question with its norm. 4 points

4. Given a question as input, compute the cosine similarity between this question and all the documents in the collection and extract the maximum. 4 points

### 2.2.3 Building an Inverted Index

The procedure you programmed has to compare the question to all the documents. This is something not viable for collections of billions of documents. You will represent your collection with an inverted index instead. Table 3 from Nugues (2014) shows an example of such an index, where each key is a word and each value, a list of documents that contains the word. You will replace the word positions with the $TF \cdot IDF$ values and in your case, the words will be in lower case.

Table 3: An inverted index. Each word in the dictionary is linked to a posting list that gives all the documents in the collection where this word occurs and its positions in a document. Here, the position is the word index in the document. In the examples, a word occurs at most once in a document. This can be easily generalized to multiple occurrences

| **Words (Keys)** | **Posting lists (Values)** |
|---|---|
| *America* | (D1, 7) |
| *Chrysler* | (D1, 1) → (D2, 1) |
| *in* | (D1, 5) → (D2, 5) |
| *investments* | (D1, 4) → (D2, 4) |
| *Latin* | (D1, 6) |
| *major* | (D2, 3) |
| *Mexico* | (D2, 6) |
| *new* | (D1, 3) |
| *plans* | (D1, 2) → (D2, 2) |
| ... | |

1. Write a program to build an inverted index from the document collection. This index will be similar to that in Table 3. The words you collected from the corpus are the keys of a Map (or a dictionary in Python) and the values are Maps themselves, where the key is the document name and the value is the $TF \cdot IDF$ score. 12 points

2. Compute the cosine similarity between the question and the documents stored in the inverted index. 6 points

### 2.3 Candidate Extraction

In this section, we will focus on the third module in Fig. 3.

1. Given the question in the introduction of the programming part (Part II), let us suppose that the passage retrieval module retrieves this paragraph from Wikipedia (`https://en.wikipedia.org/wiki/China-United_States_relations`)

   > ... Although Brzezinksi sought to quickly establish a security relationship with Beijing to counter the Soviet Union, Carter sided with Vance in believing that such a deal would threaten existing U.S.-Soviet relations, including the SALT II negotiations. Thus, the administration decided to cautiously pursue political normalization and not military relations. Vance, Brzezinski, and Oksenberg traveled to Beijing in early 1978 to work with Leonard Woodcock, then head of the liaison office, to lay the groundwork to do so. The United States and the People's Republic of China announced on December 15, 1978 that the two governments would establish diplomatic relations on January 1, 1979.

   List all the candidates the baseline extractor of IBM Watson would extract from this passage. See *Finding needles in the haystack: Search and candidate generation* (Chu-Carroll et al., 2012) for a description of the baseline extractor. 4 points

2. Table 4 shows the words of a sentence from the Wikipedia excerpt annotated with their part of speech. Write a program that enables your system to extract the candidates from the sentence. As input you will use a list of triples `(index, word, pos)`, for instance a Java `List`, storing the words in the table. You can use the triple representation you want and you will assume that your program has read the list. Your program should extract all the candidates in this table and must be as generic as possible (i.e. work for other sentences). 8 points

3. Once extracted, how would you rank the candidates? 8 points

4. Outline a strategy to link the entity *Carter* to the word *president* in the question. To serve as inspiration, you can read the Section PRISMATIC Search in *Finding needles in the haystack: Search and candidate generation* (Chu-Carroll et al., 2012). 8 points

# References

Ball, G., Ling, D., Kurlander, D., Miller, J., Pugh, D., Skelly, T., Stankosky, A., Thiel, D., Dantzich, M. V., and Wax, T. (1997). Lifelike computer characters: the Persona project at Microsoft Research. In Bradshaw, J. M., editor, *Software Agents*, pages 191–222. AAAI Press/MIT Press, Cambridge, Massachusetts.

| Index | Word | POS | Index | Word | POS |
|---|---|---|---|---|---|
| 1 | Although | IN | 20 | with | IN |
| 2 | Brzezinksi | NNP | 21 | Vance | NNP |
| 3 | sought | VBD | 22 | in | IN |
| 4 | to | TO | 23 | believing | VBG |
| 5 | quickly | RB | 24 | that | IN |
| 6 | establish | VB | 25 | such | PDT |
| 7 | a | DT | 26 | a | DT |
| 8 | security | NN | 27 | deal | NN |
| 9 | relationship | NN | 28 | would | MD |
| 10 | with | IN | 29 | threaten | VB |
| 11 | Beijing | NNP | 30 | existing | VBG |
| 12 | to | TO | 31 | U.S.-Soviet | JJ |
| 13 | counter | VB | 32 | relations | NNS |
| 14 | the | DT | 33 | , | , |
| 15 | Soviet | NNP | 34 | including | VBG |
| 16 | Union | NNP | 35 | the | DT |
| 17 | , | , | 36 | SALT | NNP |
| 18 | Carter | NNP | 37 | II | NNP |
| 19 | sided | VBD | 38 | negotiations | NNS |

Table 4: Words annotated with their part of speech.

Chu-Carroll, J., Fan, J., Boguraev, B., Carmel, D., Sheinwald, D., and Welty, C. (2012). Finding needles in the haystack: Search and candidate generation. *IBM Journal of Research and Development*, 56(3.4):6:1–6:12.

Ferrucci, D. A. (2012). Introduction to "This is Watson". *IBM Journal of Research and Development*, 56(3.4):1:1 –1:15.

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

Pradhan, S., Ramshaw, L., Marcus, M., Palmer, M., Weischedel, R., and Xue, N. (2011). CoNLL-2011 shared task: Modeling unrestricted coreference in OntoNotes. In *Proceedings of the Fifteenth Conference on Computational Natural Language Learning: Shared Task*, pages 1–27, Portland, Oregon, USA. Association for Computational Linguistics.

Tjong Kim Sang, E. F. and Buchholz, S. (2000). Introduction to the CoNLL-2000 shared task: Chunking. In *Proceedings of CoNLL-2000 and LLL-2000*, pages 127–132, Lisbon.