EDAN20
Final Examination

Pierre Nugues

October 29, 2019

The examination is worth 240 points. The distribution of points is indicated with the questions. You need 40% 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 120 points.

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

Chapter 1. Annotate each word of the sentence below with its part of speech:

US companies battle for control of 5G spectrum[1]

You will use parts of speech you learned at school: common noun, proper noun, verb, or preposition. There are five nouns, two prepositions, and one verb. In this sentence, you will consider that *battle* is a verb. 3 points

Chapter 1. Annotate each group (chunk) in the sentence:

US companies battle for control of 5G spectrum

with its type, either verb group (VP), noun group (NP), or prepositional group (PP). To annotate a group, draw a box around it and mark its type with either VP, NP, or PP. There are one verb group, three noun groups, and two prepositional groups. Each prepositional group consists of one single word: The preposition. 4 points

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

US companies battle for control of 5G spectrum

and describe what entity linking is on this example and why it can be ambiguous. Give examples of such ambiguity for the entities in the sentence. In you have no named entity in mind, invent one.

You will consider that the name of a technique is a named entity when it corresponds to a trademark. See the footnote[2]. 4 points

[1]Retrieved on October 21, 2019 from `www.ft.com`

[2]The 3rd Generation Partnership Project (3GPP) unites [Seven] telecommunications stan-

Chapter 1. Represent the sentence:

> US companies battle for control of 5G spectrum

with a predicate–argument structure. You should identify one predicate, consisting of a verb, and up to three arguments:

```
predicate(arg0, arg1, arg2)
```

where the arguments will correspond to a combatant (`arg0`), another combatant, if separate (`arg1`), and a purpose (`arg2`).

One of the arguments may be missing in the sentence. Imagine what it could be then. 4 points

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

> **Razzior mot byggbolag – facket jublar**
>
> Flera allvarliga arbetsmiljöbrister uppdagades vid en omfattande myndighetskontroll av flera byggarbetsplatser i förra veckan. En fjärdedel av företagen som utförde jobb stoppades från fortsatt arbete.
>
> ...
>
> "Äntligen lyssnar regeringen och myndigheterna på Byggnads. Vi byggnadsarbetare är förbannat trötta på den kriminalitet som tagit sig in i vår bransch", skriver byggfackets ordförande Johan Lindholm i en kommentar till TT.
>
> ...
>
> I och med ett regeringsuppdrag från 2018 har flera myndigheter trappat upp sitt arbete mot brott i arbetslivet. Förra veckans kontroller på totalt 75 byggarbetsplatser är en del av den insatsen. På vissa av arbetsplatserna var säkerheten så bristfällig att arbetet tvingades upphöra omedelbart.

Source: `svd.se`, retrieved October 1st, 2019. Author TT

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-insensitive** regular expressions:

dard development organizations (ARIB, ATIS, CCSA, ETSI, TSDSI, TTA, TTC), known as "Organizational Partners" and provides their members with a stable environment to produce the Reports and Specifications that define 3GPP technologies.
...
Implementers wishing to declare conformity to the 3GPP specifications related to 5G may mark their equipment and documentation with the 5G logo which is protected by copyright and trademark for the benefit of the 3GPP Partners. On receipt of the email authorization from 3GPP, a royalty free Trademark license is granted to users to allow the use of the 3GPP 5G logo in relation with their products and services complying with the 3GPP specifications. Retrieved on October 21, 2019 from `www.3gpp.org`

1. `arbets`
2. `arbets\s`
3. `arbetsplatser(na)?`
4. `arbetsplatser(na){1,}`
5. `\p{L}+platser\p{L}+`
6. `bygg\p{L}+\.`
7. `[0-9]+`
8. `[0-9]{1,2}`
9. `[0-9]{3,}`
10. `(a)(.)\2\1`

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

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

when applied to the text above (*Flera allvarliga arbetsmiljöbrister...*). 3 points

Chapter 2. Write a simple regular expression that matches all the words of a text: lower and uppercase.

You will use a Unicode regular expression of the form `\p{}`. 2 points

Chapter 2. In this exercise, you will write a small program that matches strings consisting of letters, digits, and dashes, and containing at least one letter and one digit like `V70`, `HP-21`, `x290`, or `A320neo`. In the invented text below:

> In 2019, I flew on an A320neo to Paris and I had my old HP-21 with me.

your program should identify `A320neo` and `HP-21`.

Your program will read the file with these Python lines:

```
import regex as re

text = open('text.txt', encoding='utf8').read()
```

and then extract the strings with regexes. We will set aside the dashes to make it simpler.

You will justify the usefulness of this program and proceed as suggested below. If you think you can solve this problem in a better way, please do it:

1. Why such a program could be useful? 1 point
2. Write a regular expression `regex_1` that extracts tokens consisting of either letters, digits, or letters and digits[3]. You will use Unicode regex classes `\p{}` and gather them in a regex class `[]`; You will apply your regex to `text` using the `re.findall()` function.

   Your program will return a list of tokens called `words_numbers`: 4 points

[3]`N` is the Unicode class to denote the numbers

```
words_numbers = re.findall('regex_1', text)
```

3. Write a regex `regex_2` that matches the tokens in `words_numbers` containing at least one digit. You will use it with the `re.match()` statement: 4 points

   ```
   re.match('regex_2', token)
   ```

   The `re.match()` function returns a match object if `regex_2` matches exactly `token`, otherwise `None`;

4. Write a regex `regex_3` that matches the tokens in `words_numbers` containing at least one letter. You will use it with the `re.match()` statement: 4 points

   ```
   re.match('regex_3', token)
   ```

5. Write a small program (a loop) that traverses your `words_numbers` list and extracts the tokens containing of at least one letter and one digit. You will use a conjunction of the two previous statements. 1 point

Your final program should have this structure:

```
import regex as re

text = open('text.tex', encoding='utf8').read()
words_numbers = re.findall('regex_1', text)
for token in words_numbers:
    if YOUR CODE WITH regex_2 AND regex_3:
        print(token)
```

Chapter 3. What do the `\p{Lu}+`, `\p{Greek}+`, `\p{Hiragana}+` and `\P{Hiragana}+` Unicode regular expressions match? Note the uppercase `P` in the last expression.

Please, provide one answer for each regular expression. 4 points

Chapter 4. Describe what a supervised classifier is. What is the typical notation of the input matrix of a classifier (predictors) and of the output vector (response). Give the name of two classifiers: The one you used for the assignments and a second one. 3 points

Chapter 5. Speech recognition engines usually consist of two parts: An acoustic processing module and a language modeling module. In the this exercise and the next ones, we will suppose that the output of the acoustic module is this sequence of phonetic symbols:

['ðəb'oɪz'iːt'ðəs'ændwɪdʒɪz].

This sequence has at least eight possible interpretations, two of them being:

The boys eat the sandwiches

and

Words	Freq.	Bigrams	Freq.
The	25,000,000,000	<s> The	260,000,000
boys	55,000,000	The boys	520,000
eat	30,000,000	boys eat	unseen
the	25,000,000,000	eat the	900,000
sandwiches	3,000,000	the sandwiches	unseen
The	25,000,000,000	<s> The	260,000,000
boys	55,000,000	The boys	520,000
eat	30,000,000	boys eat	unseen
the	25,000,000,000	eat the	900,000
sand	15,000,000	the sand	1,700,000
which	800,000,000	sand which	unseen
is	5,000,000,000	which is	90,000,000

Table 1: Unigram and bigram frequencies. Counts derived and rounded from the *Google Web Trillion Word Corpus* and used Norvig's *How To Do Things With Words, The Count counts.* Unigrams are from the 1/3 million most frequent words; bigrams are from the 1/4 million most frequent bigrams. The total number of tokens in the unigram file is 600,000,000,000

The boys eat the sand which is

We will use language models to disambiguate a sequence of phonemes and transform it in a sequence of words. In this exercise, we will use the words themselves. In a more realistic case, we would use a phonetic dictionary instead.

1. List another possible sequence of words matching this phonetic sequence. 2 points

Chapter 5. We will first use a unigram language model to rank the two sentences:

The boys eat the sandwiches

and

The boys eat the sand which is

1. Give the probability formula of the two sentence candidates using a unigram language model. You will use the notation $P(\text{word})$ to denote the probability of a word and you will ignore possible start-of-sentence tokens (`<s>`). You have two formulas to write; 2 points
2. Compute these probabilities from the frequencies in Table 1. Use fractions to represent the terms in the product, *i.e.* you will write $\frac{1}{3}$, for instance. The total number of words is 600,000,000,000.

   Use Tables 7 and 8 in the *Appendix* to fill in the values; 5 points
3. Compute the product with a calculator; 3 points
4. Which candidate has the highest probability? 1 point
5. Discuss this result. 2 points

Chapter 5. You will now use bigrams:

1. Give the probability formula of the two sentence candidates using a bigram language model. You will use the notation $P(...)$ and you assume that your sentence starts with a start-of-sentence symbol: `<s>`; 2 points
2. You will handle the unseen bigrams with a backoff strategy as in the second assignment. Explain what it is; 2 points
3. You will suppose that the sentences have an average of 25 words and that the total number of tokens is 600,000,000,000. Give a rough estimate of $P(<\text{s}>)$ and of the number of `<s>` tokens, $Count(<\text{s}>)$.

   Keep it very simple: Do not normalize the probabilities i.e. do not add the count of start-of-sentence symbols to the total number of tokens. You will suppose that the corpus size is 600,000,000,000 with or without start-of-sentence symbols; 3 points
4. Compute these probabilities from the frequencies in Table 1. You will use fractions to represent the terms in the product, *i.e.* you will write $\frac{1}{3}$.

   Use Tables 9 and 10 in the *Appendix* to fill in the values. 5 points
5. Compute the product with a calculator; 3 points
6. Which candidate has the highest probability? 1 point
7. Comparing the sentence probabilities respectively with unigram and bigram language models is paradoxical. Can you explain why? Think of the size of the corpus. 2 points

Chapter 10. In this exercise, you will annotate a sentence with its syntactic groups (chunks):

US companies battle for control of 5G spectrum

You will use the BIO tagset (or IOB2 tagset), where B stands for the beginning of a chunk, I for inside, and O for outside. You will also use a suffix to denote the type of the chunk. This is the same annotation as in the 3rd assignment of the course.

To help you, you will follow the model of the annotation of this sentence in Table 2:

A record date has not been set

where the word *not* is a negation adverb. In the sentence, there are only two chunks: A noun chunk followed by a verb chunk. The verb chunk consists of the main verb and all its auxiliaries. The word *not* is part of this verb chunk.

Fill in Table 3 similarly with the parts of speech and the chunks with the BIO tagset and syntactic suffixes, such as NP, VP.

1. As part of speech you will use either: Noun, Verb, or Preposition. 1 point
2. As chunk, you will use either: B-NP, I-NP, B-PP, B-VP. 3 points

Words	Parts of speech	Chunks
A	Determiner (Article)	B-NP
record	Noun	I-NP
date	Noun	I-NP
has	Verb	B-VP
not	Adverb	I-VP
been	Verb	I-VP
set	Verb	I-VP

Table 2: The sentence *A record date has not been set* with a CoNLL-like annotation

Words	Parts of speech	Chunks
US		
companies		
battle		
for		
control		
of		
5G		
spectrum		

Table 3: The sentence *US companies battle for control of 5G spectrum* with a CoNLL-like annotation

Chapter 15. The CoNLL 2012 dataset contains sentences with their semantic roles. We will use a simplified annotation of it in this examination. Table 4 shows an example of it for the sentence:

> I worry more about things becoming so unraveled on the other side that they might become unable to negotiate.

This format uses columns to describe the words, parts of speech, and the semantic predicates and arguments of a sentence.

The two first columns are the words and their parts of speech. The rest of the columns corresponds to the predicates and their arguments:

1. The third column in Table 4 indicates the predicates and their respective sense. How many predicates are there in this sentence? List them. 2 points
2. The columns to the right of the predicates represent their respective arguments in their order in the sentence. The arguments are bracketed with their type to the right of the first bracket. How many arguments does the first predicate has (4th column)? The (V*) code is to designate the predicate. 2 points
3. The Propbank database gives the meaning of the arguments. The verb *worry.02* has two core arguments:
   (a) Arg0-PPT: *worrier* (vnrole: 31.3-2-experiencer)

Word	POS	Pred	Args1	Args2	Args3	Args4
I	PRP	–	(ARG0*)	*	*	*
worry	VBP	worry.02	(V*)	*	*	*
more	RBR	–	(ARGM-ADV*)	*	*	*
about	IN	–	(ARG1*	*	*	*
things	NNS	–	*	(ARG1*)	*	*
becoming	VBG	become.01	*	(V*)	*	*
so	RB	–	*	(ARG2*	*	*
unraveled	JJ	–	*	*)	*	*
on	IN	–	*	(ARGM-LOC*	*	*
the	DT	–	*	*	*	*
other	JJ	–	*	*	*	*
side	NN	–	*	*)	*	*
that	IN	–	*	(C-ARG2*	*	*
they	PRP	–	*	*	(ARG1*)	(ARG0*)
might	MD	–	*	*	(ARGM-MOD*)	*
become	VB	become .01	*	*	(V*)	*
unable	JJ	–	*	*	(ARG2*	*
to	TO	–	*	*	*	*
negotiate	VB	negotiate.01	*)	*)	*)	(V*)
.	.	–	*	*	*	*

Table 4: A simplified excerpt from CoNLL 2012

(b) Arg1-PAG: *topic of worry* (vnrole: 31.3-2-stimulus)

Give the values of Arg0 and Arg1 in the sentence, i.e. the words that form these arguments in the sentence. 3 points

4. The ARGM arguments are not specific to a verb and can apply to any of them. M is for modifier. ARGM-ADV means that it is an adverb that qualifies the predicate. What is its value (the words that compose it)? 2 points

5. The 5th column corresponds to the second predicate. In Propbank, *become.01* has two core arguments:

   (a) Arg1-PPT: *entity changing*

   (b) Arg2-PRD: *new state*

   When an argument is split into two segments, the second segment starts with a C- meaning discontinuous. Give the value of the two complete core arguments of this column. 2 points

6. ARGM-LOC corresponds to a location. Give its value in this column; 1 point

7. The 6th column corresponds to the third predicate, the second *become.01* of the sentence. Give the value of all the arguments. MOD means modal verb; 3 points

8. Finally, *negotiate.01* has three arguments:

   (a) Arg0-PAG: *negotiator* (vnrole: 36.1-Agent)

   (b) Arg1-COM: *explicit other party* (vnrole: 36.1-co-agent)

   (c) Arg2-PPT: *agreement* (vnrole: 36.1-Theme)

   Give their values in the 7th column. 1 point

9. In this column, two arguments do not show. Name them. 1 point

10. How could you find the two missing arguments of *negotiate.01*? 2 points

11. Represent the predicate–argument structures for the four predicates in the form of:

```
predicate(arg0: value, arg1: value, arg2: value, ...)
```

You have four lines to write. 2 points

2 Problem

In this part, documents are allowed. It is worth 120 points.

Semantic role labeling is the process of recognizing predicate–argument structures in a sentence. This is also called semantic parsing. In this part of the examination, your will analyze and implement a simplified version of a semantic role labeler created by Zhou and Xu (2015). Although their system is now behind the state of the art, the techniques they introduced are still dominant in this field. For the most recent figures, see He et al. (2018).

Before their paper, the most common way to implement a semantic role labeler was to apply a part-of-speech tagger and a dependency parser to a sentence, extract features from the words, parts of speech, and dependency graph, and then train a classifier from these features. This classifier was applied to an unseen sentence to predict the predicates and the arguments.

In this part, you will program a system that will directly predict the arguments from the words.

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

2.1 Analyzing the Paper

1. Read the *Abstract* and *Introduction* in Zhou and Xu (2015, pp. 1127-1128).
   (a) From the *Abstract* and *Introduction* sections, describe the key properties of Zhou and Xu (2015)'s method and what they claim is novel; 5 points
2. Read the three introduction paragraphs up to Section 3.1 of the *Approaches* section (Zhou and Xu, 2015, pp. 1128-1129).
   (a) Describe what are the x and y variables[4]; 4 points
   (b) Comment Eq. 1 and identify the course assignment, where you used a similar technique. Describe very shortly what you did and what were $\mathbf{x}$ and y; 2 points
   (c) Eq. 1 uses an activation function σ. Knowing that you used logistic regression in the laboratories, what is the name of the σ function you used? 1 point
   (d) Give the name of the machine learning engine Zhou and Xu (2015) use instead. 2 points
3. Read the last paragraph of the *Long Short-Term Memory* subsection (Zhou and Xu, 2015, Sect. 3.1, pp. 1129-1130), starting with *In this work, we utilize...* and ending with *... good performance.*
   (a) Knowing that the input corresponds to the sentence's words, represent graphically how these words are processed by the LSTM and give a short description of your sketch. You can think of a LSTM as an elaborate logistic regression classifier. You will represent it as a box in your sketch. 6 points

[4] x should probably be better denoted $\mathbf{x}$

2.2 Understanding the Corpus

The core idea of Zhou and Xu (2015)'s method is to consider the semantic arguments of a predicate as chunks (syntactic groups) and process them as you did in the course's third assignment on chunking.

1. In Table 1 of Zhou and Xu (2015, Section 3.2)'s paper, the arguments of the *set.02* predicate are in the label column (last column)[5]. The Propbank database lists the following arguments for *set.02:*

   (a) Arg0-PAG: *agent, setter*

   (b) Arg1-PPT: *thing set*

   (c) Arg2-PRD: *attribute of arg1*

   Give the name and value (the words in the sentence) of the two arguments of *set* in this table. 2 points

2. Following the annotation in Table 4, recreate the bracketed annotation of the arguments of the sentence *A record date hasn't been set.* You will fill in the Args1 column in Table 5.

   Use the sheet in the *Appendix.* 5 points

Words	Pred	Args1
A	–	
record	–	
date	–	
has	–	
not	–	
been	–	
set	set.02	(V*)
.	–	*

Table 5: Fill in the arguments with a bracketed notation

3. You will now replace the bracketed annotation in Table 4 with the BIO tagset and suffixes. Using the example of Table 1, in Zhou and Xu (2015, Section 3.2)'s paper, fill in Table 6.

   Use the sheet in the *Appendix.* 10 points

   You will consider that ARG2 and C-ARG2 are different tags. The first argument of *worry.02* and the rest of the first row are given to help you start.

[5]In their paper, Zhou and Xu (2015) do not give the sense number of *set*. In the original corpus, it is sense number 02

Word	Pred	Args1	Args2	Args3	Args4
I	–	B-ARG0	O	O	O
worry	worry.02	B-V			
more	–				
about	–				
things	–				
becoming	become.01		B-V		
so	–				
unraveled	–				
on	–				
the	–				
other	–				
side	–				
that	–				
they	–				
might	–				
become	become .01			B-V	
unable	–				
to	–				
negotiate	negotiate.01				B-V
.	–	O			

Table 6: Fill in the arguments with a BIO notation

2.3 Argument Detection

In this section, you will convert the corpus annotation from brackets to chunks, extract the features, vectorize them, and train a model. You will apply this model to a test set.

The annotation you will analyze is slightly simplified from that of the real original OntoNotes corpus. You will assume that Table 4 contains all the cases to analyze. The number of predicates and thus argument columns may vary however, as well as the number and names of arguments in the columns.

2.3.1 From Brackets to Chunks

Given a CoNLL corpus using the format in Table 4, you will now write a converter that transforms the bracketed arguments into BIO tags like those in Table 6.

You will assume that the whole training set is stored in the `corpus_train` variable, which consists of a list of sentences, where each sentence is a list of Python dictionaries. Each dictionary corresponds to one row i.e. one word, where the keys are the column heads and the values, the values in the row.

The excerpt below shows the data encoding of three first rows from Table 4 as well as the two last rows:

```
corpus_train = [
...
[{'word': 'I', 'pos': 'PRP', 'pred': '-', 'args1': '(ARG0*)',
    'args2': '*', 'args3': '*', 'args4': '*'},
{'word': 'worry', 'pos': 'VBP','pred': 'worry.02', 'args1': '(V*)',
    'args2': '*', 'args3': '*', 'args4': '*'},
{'word': 'more','pos': 'RBR','pred': '-', 'args1': '(ARGM-ADV*)',
    'args2': '*', 'args3': '*', 'args4': '*'},
...
{'word': 'negotiate', 'pos': 'VB', 'pred': 'negotiate.01', 'args1': '*)',
    'args2': '*)', 'args3': '*)', 'args4': '(V*)'},
{ 'word': '.', 'pos': '.', 'pred': '-', 'args1': '*',
    'args2': '*', 'args3': '*', 'args4': '*'}],
...
]
```

As a suggestion, you can solve this bracket-to-tag transformation in six steps. Nonetheless, feel free to use your own solution, if you find something smarter.

1. Given a sentence, write the `index_preds(sentence)` function that returns the list of row indices of the predicates. As a convention, the first row will have index 0.

   In Table 4, your function should return `[1, 5, 15, 18]`, as the predicates are in rows 1, 5, 15, and 18, and in Table 5, `[6]` as this sentence has only one predicate in row 6; 6 points

2. From the index list, compute the number of predicates in the sentence `pred_cnt`; 1 point

3. In a sentence, the argument columns will be named `'args1'`, `'args2'`, `'args3'`, ..., `'argsN'`, and there will be as many columns as there are predicates in the sentence.

   Write the `gen_arg_names(pred_cnt)` function that generates the argument column names from the number of arguments: `pred_cnt`. For instance, if `pred_cnt` equals 4, your function should generate: 3 points

   ```
   ['args1', 'args2', 'args3', 'args4']
   ```

4. Given a sentence and an argument column name, for instance `'args1'`, write the conversion function,

   ```
   brackets2BIO_col(sentence, arg_col_name)
   ```

   for the corresponding column. For the `'args1'` column, your function

   ```
   brackets2BIO_col(sentence, 'args1')
   ```

   should return:

   ```
   corpus_train = [
   ...
   [{'word': 'I', 'pos': 'PRP', 'pred': '-', 'args1': 'B-ARG0',
       'args2': '*', 'args3': '*', 'args4': '*'},
   {'word': 'worry', 'pos': 'VBP','pred': 'worry.02', 'args1': 'B-V',
       'args2': '*', 'args3': '*', 'args4': '*'},
   {'word': 'more','pos': 'RBR','pred': '-', 'args1': 'B-ARGM-ADV',
   ```

```
    'args2': '*', 'args3': '*', 'args4': '*'},
...
{'word': 'negotiate', 'pos': 'VB', 'pred': 'negotiate.01',
    'args1': 'I-ARG1', 'args2': '*)', 'args3': '*)',
    'args4': '(V*)'},
{ 'word': '.', 'pos': '.', 'pred': '-', 'args1': 'O',
    'args2': '*', 'args3': '*', 'args4': '*'}],
...
]
```

Note that the other columns are still in the bracketed format.

To help you, a possible start for this function is: 16 points

```
def brackets2BIO_col(sentence, arg):
    in_chunk = False  # If we are in a chunk or not
    current_chunk = None  # The value of the argument,
                          # for instance ARG1
    for word in sentence:
        if word[arg][0] == '(' and word[arg][-1] == ')':
            word[arg] = 'B-' + word[arg][1:-2]
        elif...
```

5. Using the functions you wrote before, write the conversion function `brackets2BIO(sentence)` for all the arguments of a sentence. 4 points

6. Apply your function to all the sentences in `corpus_train`. 1 points

2.3.2 Extracting the Features

1. Read the two first paragraphs of the *Pipeline* section in Zhou and Xu (2015, Sect. 3.2, page 1130)'s paper and describe the feature vector they use.

2. Build manually the feature matrix **X** as well as the **y** vector for the sentence *A record date has not been set.* in Table 1 of the paper (Zhou and Xu, 2015, Sect. 3.2, page 1130). Your matrix should have 8 rows and 4 columns. Your **y** vector will have 8 dimensions. You will use symbols and not numbers. As in Zhou and Xu (2015)'s paper, you will use a context length of 3. This exercise is simply the identification of **X** and **y** in Table 1, page 1130, of their paper. 2 points

   Use the sheets in the *Appendix*;

3. Similarly to the previous exercise, build manually the feature matrix **X** as well as the **y** vector for the three first words of the sentence

   > I worry more about things becoming so unraveled on the other side that they might become unable to negotiate.

   in Table 6. You will arrange your matrix and vector by order of predicate and you will separate the corresponding groups of features with ellipses (...);

Your matrix will have 12 rows and 4 columns; your vector will have 12 dimensions, not counting the ellipses. The matrix below shows you how to present your features. Just replace the dashes (‘–’) with values. 6 points

Use the sheets in the *Appendix*.

$$\mathbf{X} = \begin{bmatrix} \text{I} & \text{worry.02} & - & - \\ - & - & - & - \\ - & - & - & - \\ \ldots & \ldots & \ldots & \ldots \\ \text{I} & \text{become.01} & - & - \\ - & - & - & - \\ - & - & - & - \\ \ldots & \ldots & \ldots & \ldots \\ \text{I} & \text{become.01} & - & - \\ - & - & - & - \\ - & - & - & - \\ \ldots & \ldots & \ldots & \ldots \\ \text{I} & \text{negotiate.01} & - & - \\ - & - & - & - \\ - & - & - & - \end{bmatrix} ; \mathbf{y} = \begin{bmatrix} \text{B-ARG0} \\ - \\ - \\ \ldots \\ - \\ - \\ - \\ \ldots \\ - \\ - \\ - \\ \ldots \\ - \\ - \\ - \end{bmatrix}$$

4. You will now extract all the features of all the sentences of your corpus. You will ignore a possible padding of the context. For a given sentence, you will have to run the extraction as many times as there are predicates. You will store the result in a dictionary with the keys `'argu'`, `'pred'`, `'ctx-p'`, and `mr` as in Zhou and Xu (2015).

   (a) Write the function

   ```
   X, y = extract_features_col(sentence, arg, arg_inx)
   ```

   that extracts the feature **X** matrix and the **y** vector for one predicate in one sentence. `arg` is the column name, for instance `args2` and `arg_inx` is the index of the predicate in the sentence, 5, for instance: 12 points

   ```
   extract_features_col(sentence, 'args2', 5)
   ```

   (b) Write the function `X, y = extract_features(sentence)` that extracts the feature **X** matrix and the **y** vector for all the predicates in one sentence; 6 points

   (c) Write the loop that extracts all the features for all the sentences in `corpus_train`. 1 point

2.3.3 Training a Model

You will now train a model with a simple logistic regression classifier.

1. Write the code to convert your **X** matrix of symbols into a one-hot encoded matrix. It is just two lines and you will use `DictVectorizer()`; 2 points

2. Write the code to train a model. It is just two lines and you will use the `LogisticRegression()` class; 2 points

2.3.4 Predicting the Tags

You will now predict the arguments. We will suppose first that the predicates are given in the test set. As for the training set, you will assume that the whole test set is stored in the `corpus_test` variable, which consists of a list of sentences, where each sentence is a list of Python dictionaries. The only difference is that `corpus_test` will not have the `argsn` keys.

1. Describe how you would modify

   `extract_features_col(sentence, arg, arg_inx)`

   and

   `extract_features(sentence)`

   to extract the features from the test set. You do not need to write a program. 5 points

2. Write a function to convert your $\mathbf{X}_{\text{test}}$ matrix of symbols into a one-hot encoded matrix. You will use `DictVectorizer()`; 1 point

3. Write the code to predict the arguments. 3 points

2.4 Predicate Detection

In their paper, Zhou and Xu (2015) assumed that the predicates were given in the test set. This is not the case in a real application. We will now imagine a technique to determine if a word is a predicate or not, and if yes, its sense number.

1. Knowing that the predicates are verbs and that all the predicates are in the Propbank dictionary, describe a baseline technique to tell if a word is a predicate or not; 4 points

2. Each predicate has a finite number of senses. For example, *worry* has two senses in Propbank:

   (a) worry.01: Use this sense when the worrier is the direct object of an active construction, or the subject of a passive construction.

      i. Arg0-PAG: *cause of worrying, troublesome topic* (vnrole: 31.1-stimulus)
      ii. Arg1-PPT: *worrier* (vnrole: 31.1-experiencer)
      iii. Arg2-MNR: *instrument, when separate from agent*

      Example: Mr. Rowe also noted that [*political concerns*]$_{\text{Arg0}}$ also **worried** [*New England Electric*]$_{\text{Arg1}}$.

   (b) worry.02: Use this sense when the worrier is the subject of an active construction.

      i. Arg0-PPT: *worrier* (vnrole: 31.3-2-experiencer)
      ii. Arg1-PAG: *topic of worry* (vnrole: 31.3-2-stimulus)

Example: Similarly, when Chen Shui-bian negotiates with Beijing, [*no one*]$_{\text{Arg0}}$ will **worry** [*about him selling Taiwan down the river*]$_{\text{Arg1}}$.

Predicates have at least one denoted 01.

Describe shortly a simple classifier to determine the sense of *worry* using a context of five words surrounding this word. 4 points

3. Knowing that there are about 5500 verbs in Propbank, how would you then proceed to determine the sense of any given predicate? 4 points

References

He, L., Lee, K., Levy, O., and Zettlemoyer, L. (2018). Jointly predicting predicates and arguments in neural semantic role labeling. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics*, Melbourne.

Zhou, J. and Xu, W. (2015). End-to-end learning of semantic role labeling using recurrent neural networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics*, Beijing.

3 Appendix

You can use the tables below in your exam papers. Just detach them and fill them in. Insert them with your sheets. Do not forget to write your exam code on the sheets.

Words	**Freq.**	**Probabilities**
The	25,000,000,000	
boys	55,000,000	
eat	30,000,000	
the	25,000,000,000	
sandwiches	3,000,000	
Sentence probability		

Table 7: Fill in the probabilities with a unigram language model. The total number of tokens is 600,000,000,000. You will try to simplify your fractions, for instance from 2/6 to 1/3

Words	**Freq.**	**Probabilities**
The	25,000,000,000	
boys	55,000,000	
eat	30,000,000	
the	25,000,000,000	
sand	15,000,000	
which	800,000,000	
is	5,000,000,000	
Sentence probability		

Table 8: Fill in the probabilities with a unigram language model. The total number of tokens is 600,000,000,000. You will try to simplify your fractions, for instance from 2/6 to 1/3

$Count(\text{<s>}) =$

Words	**Freq.**	**Conditional probabilities**
<s> The	260,000,000	
The boys	520,000	
boys eat	unseen	
eat the	900,000	
the sandwiches	unseen	
Sentence probability		

Table 9: Fill in the probabilities with a bigram language model. You will try to simplify your fractions, for instance from 2/6 to 1/3

Words	**Freq.**	**Conditional probabilities**
<s> The	260,000,000	
The boys	520,000	
boys eat	unseen	
eat the	900,000	
the sand	1,700,000	
sand which	unseen	
which is	90,000,000	
Sentence probability		

Table 10: Fill in the probabilities with a bigram language model. You will try to simplify your fractions, for instance from 2/6 to 1/3

Words	Parts of speech	Chunks
US		
companies		
battle		
for		
control		
of		
5G		
spectrum		

Table 11: A sentence with a CoNLL-like annotation

Words	Pred	Args1
A	–	
record	–	
date	–	
has	–	
not	–	
been	–	
set	set.02	(V*)
.	–	*

Table 12: Fill in the arguments with a bracketed notation: Third column

Word	Pred	Args1	Args2	Args3	Args4
I	–	B-ARG0	O	O	O
worry	worry.02	B-V			
more	–				
about	–				
things	–				
becoming	become.01		B-V		
so	–				
unraveled	–				
on	–				
the	–				
other	–				
side	–				
that	–				
they	–				
might	–				
become	become .01			B-V	
unable	–				
to	–				
negotiate	negotiate.01				B-V
.	–	O			

Table 13: Fill in the arguments with a BIO notation: columns 3 to 6

$$\mathbf{X} = \begin{bmatrix} \text{A} & \text{set} & \text{been set .} & 0 \\ - & - & - & - \\ - & - & - & - \\ - & - & - & - \\ - & - & - & - \\ - & - & - & - \\ - & - & - & - \\ . & \text{set} & \text{been set .} & 1 \end{bmatrix}; \mathbf{y} = \begin{bmatrix} \text{B-A1} \\ - \\ - \\ - \\ - \\ - \\ - \\ \text{O} \end{bmatrix}$$

$$\mathbf{X} = \begin{bmatrix} \text{I} & \text{worry.02} & - & - \\ - & - & - & - \\ - & - & - & - \\ ... & ... & ... & ... \\ \text{I} & \text{become.01} & - & - \\ - & - & - & - \\ - & - & - & - \\ ... & ... & ... & ... \\ \text{I} & \text{become.01} & - & - \\ - & - & - & - \\ - & - & - & - \\ ... & ... & ... & ... \\ \text{I} & \text{negotiate.01} & - & - \\ - & - & - & - \\ - & - & - & - \end{bmatrix}; \mathbf{y} = \begin{bmatrix} \text{B-ARG0} \\ - \\ - \\ ... \\ - \\ - \\ - \\ ... \\ - \\ - \\ - \\ ... \\ - \\ - \\ - \end{bmatrix}$$