Aims:

This exercise aims to get you to:

1. Practice the use of Stanford CoreNLP Server
2. Run Natural Language Processing (NLP) tasks using CoreNLP server (useful for data curation)
3. Making HTTP requests with Scala

1 Running Stanford CoreNLP Server on VLab

Stanford CoreNLP is a comprehensive API that provides access to a set of natural language processing tools (e.g., named-entity recognition, lemmatization, tokenization, etc.). CoreNLP is a very useful resource for performing data curation on unstructured, textual data. CoreNLP can be used as a library in Java and other programming languages or as a HTTP service through CoreNLP server. The latter provides a set of web APIs that are accessible through HTTP requests.

In this lab, we will use the CoreNLP server. First, you need to open a terminal (command line tool) in VLab and run the following program:

```
$ 9313
```

This program will setup the environment needed to run the CoreNLP server. You should see an output like the following:

```
$ 9313
Welcome to COMP9313!
newclass starting new subshell for class COMP9313...
```

Next, run the command below to start the CoreNLP server:

```
$ corenlp-server.sh
...
[main] INFO CoreNLP - http://stanfordnlp.github.io/CoreNLP/download.html
[main] INFO CoreNLP -     Threads: 2
[main] INFO CoreNLP - Starting server...
[main] INFO CoreNLP - StanfordCoreNLPServer listening at /0.0.0.0:9000
```

If everything goes fine, you should see an output similar to the one above. The server should be able to receive HTTP requests (using CoreNLP's web APIs) on http://localhost:9000.

Alternatively, you may get an output as follows (this means that the server is already up and running in a different session (or by a different user)) :

```
$ corenlp-server.sh

CoreNLP Server is running on http://localhost:9000...
```

You can open another terminal (and run the program as 9313 shown above) and then use the command below to check if CoreNLP is up and running:

```
$  wget  --post-data  'I  travelled  to  Sydney  last  night  and  had  dinner  with  John'
'localhost:9000/?properties={"annotators":"tokenize","outputFormat":"json"}' -O -
...
characterOffsetBegin":2,"characterOffsetEnd":11},{"index":-
1,"word":"to","originalText":"to","characterOffsetBegin":12,"characterOffsetEnd":14},{"inde
x":-
1,"word":"Sydney","originalText":"Sydney","characterOffsetBegin":15,"characterOffsetEnd":21
},{"index":-
1,"word":"last","originalText":"last","characterOffsetBegin":22,"characterOffsetEnd":26},{"
index":-
1,"word":"night","originalText":"night","characterOffsetBegin":27,"characterOffsetEnd":32},
{"index":-
1,"word":"and","originalText":"and","characterOffsetBegin":33,"characterOffsetEnd":36},{"in
dex":-
1,"word":"had","originalText":"had","characterOffsetBegin":37,"characterOffsetEnd":40},{"in
dex":-
1,"word":"dinner","originalText":"dinner","characterOffsetBegin":41,"characterOffsetEnd":47
},{"index":-
1,"word":"with","originalText":"with","characterOffsetBegin":48,"characterOffsetEnd":52},{"
index":-1,"word":"John","originalTe-                   100%[=====================>]   1.08K
--.-KB/s    in 0s
...
```

If the server is up and running properly, you should see an output similar to the one shown above. Notice that the first request you make may take a while before you get a response. This is because the first HTTP call to CoreNLP Server triggers the activation/initialization of a number of components that are required for performing NLP tasks. The following HTTP calls should return results faster.

In the example above, we use CoreNLP to the task of tokenizing the text sent to the server. There are multiple other tasks that you can perform with CoreNLP. We will see some of them next.

2 Performing NLP tasks with CoreNLP

Let's now perform a number of NLP tasks typically done with CoreNLP. The first task, consist in splitting a sequence of tokens into sentences. To do this, run the command below:

```
$ wget --post-data 'I travelled to Sydney yesterday. John was waiting for me at the airport.'
'localhost:9000/?properties={"annotators":"ssplit","outputFormat":"json"}' -O -
...
"sentences":[{"index":0,"tokens":[{"index":1,"word":"I","originalText":"I","characterOffset
Begin":0,"characterOffsetEnd":1,"before":"","after":"
"},{"index":2,"word":"travelled","originalText":"travelled","characterOffsetBegin":2,"chara
cterOffsetEnd":11,"before":"                                                    ","after":"
"},{"index":3,"word":"to","originalText":"to","characterOffsetBegin":12,"characterOffsetEnd
":14,"before":"                                                                 ","after":"
"},{"index":4,"word":"Sydney","originalText":"Sydney","characterOffsetBegin":15,"characterO
ffsetEnd":21,"before":" ","after":" "}
...
```

The result above consists in a list of `sentences`, where the first sentence starts at `"index": 0`. You also get a number of other attributes that characterizes the text you sent in the request.

The next example shows how to perform Part-Of-Speech (POS) tagging:

```
$ wget --post-data 'I travelled to Sydney yesterday.'
'localhost:9000/?properties={"annotators":"pos","outputFormat":"json"}' -O -
```

```
...
"sentences":[{"index":0,"tokens":[{"index":1,"word":"I","originalText":"I","characterOffset
Begin":0,"characterOffsetEnd":1,"pos":"PRP","before":"","after":"
"},{"index":2,"word":"travelled","originalText":"travelled","characterOffsetBegin":2,"chara
cterOffsetEnd":11,"pos":"VBD","before":" ","after":"
"},{"index":3,"word":"to","originalText":"to","characterOffsetBegin":12,"characterOffsetEnd
":14,"pos":"TO","before":" ","after":"
"},{"index":4,"word":"Sydney","originalText":"Sydney","characterOffsetBegin":15,"characterO
ffsetEnd":21,"pos":"NNP","before":" ","after":"
"},{"index":5,"word":"yesterday","originalText":"yesterday","characterOffsetBegin":22,"char
acterOffsetEnd":31,"pos":"NN","before":"
","after":""},{"index":6,"word":".","originalText":".",
...
```

The output shows that the word `travelled` was tagged with a part-of-speech `VBD`, which is a tag that used for verbs in past tense.

If you are interested in the base (or dictionary) form of a word, you can use CoreNLP's lemmatization. See the example below.

```
$ wget --post-data 'I travelled to Sydney yesterday.'
'localhost:9000/?properties={"annotators":"lemma","outputFormat":"json"}' -O -
...
"sentences":[{"index":0,"tokens":[{"index":1,"word":"I","originalText":"I","lemma":"I","cha
racterOffsetBegin":0,"characterOffsetEnd":1,"pos":"PRP","before":"","after":"
"},{"index":2,"word":"travelled","originalText":"travelled","lemma":"travel","characterOffs
etBegin":2,"characterOffsetEnd":11,"pos":"VBD","before":"                        ","after":"
"},{"index":3,"word":"to","originalText":"to","lemma":"to","characterOffsetBegin":12,"chara
cterOffsetEnd":14,"pos":"TO","before":"                                          ","after":"
"},{"index":4,"word":"Sydney","originalText":"Sydney","lemma":"Sydney","characterOffsetBegi
n":15,"characterOffsetEnd":21,"pos":"NNP","before":"                             ","after":"
"},{"index":5,"word":"yesterday","originalText":"yesterday","lemma":"yesterday","characterO
ffsetBegin":22,"characterOffsetEnd":31,"pos":"NN","before":"
","after":""},{"index":6,"word":".","originalText":".","lemma":".","characterOffsetBegin":3
1,...
```

In the example above, the lemma (base form) for the word `travelled` is `travel`.

Finally, you can also perform named-entity recognition (NER) using the `ner` annotation as shown in the example below:

```
$ wget --post-data 'I travelled to Australia yesterday.'
'localhost:9000/?properties={"annotators":"ner","outputFormat":"json"}' -O -
...
"sentences":[{"index":0,"tokens":[{"index":1,"word":"I","originalText":"I","lemma":"I","cha
racterOffsetBegin":0,"characterOffsetEnd":1,"pos":"PRP","ner":"O","before":"","after":"
"},{"index":2,"word":"travelled","originalText":"travelled","lemma":"travel","characterOffs
etBegin":2,"characterOffsetEnd":11,"pos":"VBD","ner":"O","before":"              ","after":"
"},{"index":3,"word":"to","originalText":"to","lemma":"to","characterOffsetBegin":12,"chara
cterOffsetEnd":14,"pos":"TO","ner":"O","before":"                                ","after":"
"},{"index":4,"word":"Australia","originalText":"Australia","lemma":"Australia","characterO
ffsetBegin":15,"characterOffsetEnd":24,"pos":"NNP","ner":"LOCATION","before":"    ","after":"
"},{"index":5,"word":"yesterday","originalText":"yesterday","lemma":"yesterday","characterO
ffsetBegin":25,"characterOffsetEnd":34,"pos":"NN","ner":"DATE","normalizedNER":"OFFSET   P-
1D","before":"      ","after":"","timex":{"tid":"t1","type":"DATE","altValue":"OFFSET     P-
1D"}},{"index":6,"word":".","originalText":".","lemma":".","characterOffsetBegin":34,
...
```

In the output above, we can see that the word `Australia` (a named-entity) is tagged with `LOCATION` (its entity type).

3 Making HTTP Requests with Scala

There are a number of external libraries that allows for making HTTP requests with Scala. One such libraries is scalaj-http (https://github.com/scalaj/scalaj-http). In the examples below, we show how to use this library to make GET requests from spark-shell (you can also use this library to make requests from Spark standalone applications).

First, you will need to start the 9313 program (if you haven't done so already).

```
$ 9313
Welcome to COMP9313!
newclass starting new subshell for class COMP9313...
```

Next, run spark-shell with the `--packages` option (`--packages` allows spark-shell to load external packages that you need to use in your program).

```
$ spark-shell --packages "org.scalaj:scalaj-http_2.11:2.4.2"
...
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.4.3
      /_/

Using Scala version 2.11.12 (OpenJDK 64-Bit Server VM, Java 1.8.0_222)
Type in expressions to have them evaluated.
Type :help for more information.

scala>
```

In the example above, we use `scalaj-http_2.11`. Here, `2.11` refers the Scala version installed in VLab. If you run the example above on your own computer, you may need to check the Scala version installed to add the correct version.

In order to make HTTP requests, you first need to import the scalaj-http library as follows:

```
scala> import scalaj.http._
import scalaj.http._
```

With the library above imported, you can now start making HTTP requests to HTTP servers. For example, the code below makes a `GET` HTTP request to www.google.com.

```
scala> Http("http://www.google.com").method("GET").asString
HttpResponse(<!doctype    html><html    itemscope=""    itemtype="http://schema.org/WebPage"
lang="en-AU"><head><meta content="text/html; charset=UTF-8" http-equiv="Content-Type"><meta
content="/images/branding/googleg/1x/googleg_standard_color_128dp.png"
itemprop="image"><title>Google</title><script nonce="6La3HbW5yrog3wW0DgZcSw==">(function()
...
```

Similarly to the example above, you can make other types of HTTP requests such as `POST`, `DELETE`, etc. More details of how to use this library can be found here: https://github.com/scalaj/scalaj-http