

COMP 474 UU,COMP 6741 UU 2204

[Home](#) / [My courses](#) / [COMP-474-2204-UU](#) / 28 March - 3 April / [Lab Session #10](#)

Lab Session #10

Introduction

Welcome to Lab #10. This lab, we'll cover the foundations for developing *Text Mining* systems, in particular how to build NLP pipelines using the [spaCy](#) library for Python.

Follow-up Lab #9

Solution Task #2 (Search based chatbots)

Here's a [sample program](#) for this task from last week that finds similar questions on the Amazon QnA dataset.

This is a [sample script](#) for the spaCy exercise.

Task #1: Question Classification

We've so far looked into classification tasks using tf-idf and cosine similarity to identify the classes the query vector falls into. Since you've learnt about using spaCy to extract linguistic features let's try to utilise linguistic features to create feature vector for our data and perform classification using a decision tree classifier in scikit-learn library.

Let's first import the necessary libraries and initialize some variables required for this task.

```
import numpy as np
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.preprocessing import LabelEncoder
from collections import Counter
import spacy

nlp = spacy.load("en_core_web_sm")
```

Now let's create a function which accepts a text as an input and return a list resembling a feature vector. A feature vector can contain any combination of features of your desire. For this task following are the features we are interested to experiment with:

1. Lemma of the ROOT text
2. Total number of occurrence of the [POS tag](#) "WP -
3. Total number of occurrence of the POS tag "WRB"
4. Total number of PERSON entity tags.
5. Total number of GRE entity tags.
6. Total number of ORG entity tags.
7. Length of the text.

Use the following snippet and fill in the code to extract the features listed above:

```
def create_features(text):
    doc = nlp(text)
    pos_list = [] # Create a list with all the POS tags in the text
    pos_count_dict = Counter(pos_list) # Count the number of POS tags within the POS list
    entity_list = [] # Create a list with all the entities within the text
    ent_count_dict = Counter(entity_list) #Count the number of entity types within the entity list
    root_lemma = # Lemma of the root token in the text
    sentence_length = # Length of the text (number of characters within the text)

    return [root_lemma, pos_count_dict['WP'], pos_count_dict["WRB"], ent_count_dict["PERSON"], ent_count_dict["GRE"]]
```

Let's define our corpus:

```
corpus = [
    'Who is Bill Gates',
    'Where is Concordia located',
    'What is AI',
    'What city is McGill located in',
    'Who is McGill'
]
```

Iterate through each document within the corpus and create a feature matrix.

```
features = []
for text in corpus:
```

```
features.append(create_features(text))
```

```
feature_mtrx = np.array(features) # Convert the feature matrix to a numpy array.
```

First column in our data contains a string value which is not a suitable datastructure for the DecisionTreeClassifier we are intending to use. Let's assign integer labels to these string values using Sklearn's LabelEncoder.

```
le = LabelEncoder()  
feature_mtrx[:, 0] = le.fit_transform(feature_mtrx[:, 0])
```

Create a label array containing the class of each document within the corpus.

```
# 0=Who, 1= Where, 2=What  
y = np.array([0, 1, 2, 2, 0])
```

Train a DecisionTreeClassifier using our feature matrix created above.

```
clf = DecisionTreeClassifier(random_state=0)  
clf.fit(feature_mtrx, y)
```

Now test your system by trying to predict the class for the following query text:

```
q = "Who is Obama"
```

You can also visualize features the decision tree considered for the prediction task.

```
import matplotlib.pyplot as plt
```

```
plt.figure()  
plot_tree(clf, filled=True, feature_names=["root", "wp_count", "wrb_count", "person", "gre", "org", "length"])  
plt.show()
```

Task #2: Text Mining

The goal for this week is to build a simple text mining system. Generally, text mining means extracting interesting, structured information from unstructured (natural language) documents. The extracted information can then be exported in various formats for further analysis, e.g., using CSV, database export, or automated knowledge graph construction. One important text mining task is Information Extraction, used for applications such as business intelligence, scientific research, news/media monitoring, social media mining, and healthcare/medical record management.

NLP Pipelines

As mentioned in the lecture, spaCy's models come with a default pipeline that you can then modify and extend. The first step is to go through the [spaCy Pipelines documentation](#) to understand the components run in a default pipeline. You can view the default pipeline with the use of following snippet of code.

```
print(nlp.pipe_names)
```

Not let's try [adding a simple custom component](#) to the pipeline to count the number of title cased tokens within a text and print out in the console.

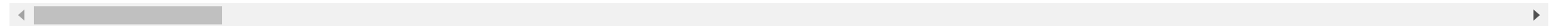
```
import spacy  
from spacy.language import Language
```

```
@Language.component("custom_title_counter")
def custom_title_counter(doc):
    title_count = sum([1 for token in doc if token.is_title])
    print(f"This document has {title_count} title cased tokens!!!")
    return doc
```

Once we have our pipeline module to perform the desired function, add it to your nlp pipeline as the last module.

Print the pipeline modules in your console to make sure your custom module is added to the nlp pipeline. Process the following paragraph with your new pipeline and see the output of your custom module.

```
text = """Concordia University (French: Université Concordia; commonly referred to as Concordia) is a public compreh
```



Information Extraction

The next step is to add an information extraction component to the pipeline. In particular, we'll start by extracting named entities (NEs), just like the entities you've extracted using the default model above. As always, the two general options for NE extraction are using either (1) machine learning or (2) a rule-based approach. While machine learning can generally provide more accurate results, rule-based systems can still be very useful in first experiments, developing demos/prototyping, and bootstrapping the manual annotations (gold standard) required for machine learning. Some entities can also be easily and accurately identified with rules (e.g., URLs, IP addresses, phone numbers) so it wouldn't make sense to apply the ML overhead.

For our first experiments here, we'll develop a rule-based IE system. As discussed in the lecture, this does not mean just using regular expressions for matching the literal text (so-called surface form). Rather, you want to combine features as extracted by your components above to obtain more general rules, e.g., by incorporating lemmas and POS tags.

To start, go through the spaCy documentation for [rule-based matching](#). Your goal today is to develop rules for identifying organizational units in a university, like faculties or departments, as you would find them on websites or scientific research publications. Rather than hard-coding string matching rules for all possible combinations like *"Department of Mathematics and Computer Science"* or *"Faculty for Extraterrestrial Space Travel"*, you should develop rules that combine unit names ("Faculty", "Department", "School", ...) with POS-tags, in particular nouns (check the [POS tags documentation](#) to find the right tag names).

Use the following text given above and try to come up with some rules and test them using the interactive [Rule-based Matcher Explorer](#) shown in the lecture.

Now let's try to do it with python code.

```
import spacy
from spacy.matcher import Matcher
nlp = spacy.load("en_core_web_sm")
matcher = Matcher(nlp.vocab)
```

Copy the pattern you used from the Rule-based Matcher Explorer as your pattern within the code.

```
pattern = #Pattern from Rule-based Matcher Explorer
```

Now add the pattern with a label to the "matcher" object created above.

```
matcher.add("uni_identifier", [pattern])
```

You can then process the text given above with your nlp pipeline. Now let's try to extract the matches matched by your pattern and print them to the console.

```
matches = matcher(doc)
```

```
for match_id, start, end in matches:
```

```
    string_id = nlp.vocab.strings[match_id] # Get string representation
```

```
    span = doc[start:end] # The matched span
```

```
    print(match_id, string_id, start, end, span.text)
```

Compare the results you obtained from the python code with the Explorer. Try experimenting with multiple patterns to extract entities.

That's all for this lab!

Last modified: Sunday, 4 April 2021, 4:17 PM

[◀ Worksheet #10](#)

Jump to...

[Lecture Slides #12 ▶](#)

You are logged in as [Yaohua Zhang](#) ([Log out](#))

[COMP-474-2204-UU](#)