koko Documentation

Release 0.0.1

BigGorilla Team

Nov 14, 2017

Contents

Koko is an information extraction tool (developed in Python 3) that allows users to query a text corpus and extract those entities that is of interest to them. A wide range of lexical, syntactic and semantic patterns can be specified using Koko's query language which can be combined to efficiently extract and score relevant entities. For a quick introduction, please start with *A Quick Guide*.

Table of Contents:

CHAPTER 1

A Quick Guide

To use Koko, there are three simple steps that you need to know about:

1. Installing Koko
2. Running Koko
3. Writing queries using Koko.

Here, we discuss each of these steps briefly.

Installation

Koko is a registered on PyPi and can be simply installed using the following command:

```
pip install pykoko
```

Note: The current version of Koko is only compatible with Python 3.

Running Koko

To run Koko, we first need to have a basic query to execute. We will discuss the syntax and semantics of Koko queries in the next section, but for now, let's focus on a simple task.

Which characters from Les Miserables introduce themselves in the book?

We can obtain a copy of the book from http://www.gutenberg.org/files/135/135-0.txt. The next step would be to write a simple query in a query file (let's call it `miserables.koko`) as follows.

```
extract "Ents" x from "135-0.txt" if
        ("My name is" x {0.1})
with threshold 0.0
```

The query above simply states that we are interested in entities (or `"Ents"`) that follow right after the words `My name is` from the specified text file. Every name that matches this pattern receives a score of `0.1`. The threshold allows us to exclude entities with a low score, but in this case we are interested to extract all the names matching our specified pattern.

Now we can run koko in a python script as follows:

```
import koko
koko.run('miserables.koko')
```

and the results would be as follows:

```
Results:

Entity name                                                Entity score
============================================================
Thénardier                                                 0.200000
Marius                                                     0.200000
Marius Pontmercy                                           0.200000
Bienvenu Myriel                                            0.100000
Jean Valjean                                               0.100000
Félix Tholomyès                                            0.100000
Madame Thénardier                                          0.100000
Champmathieu                                               0.100000
Pontmercy                                                  0.100000
Gribier                                                    0.100000
Lesgle                                                     0.100000
Éponine                                                    0.100000
Cosette                                                    0.100000
Euphrasie                                                  0.100000
Eight                                                      0.100000
```

Writing Koko Queries

Here, we discuss the expressive power of Koko queries and how Koko queries can be crafted. We start by describing the generic structure of a koko query.

The following shows the structure of a koko query which consists of four main parts.

```
extract "<type>" x from "<document>" if      # Part 1: specifying the task.
       (<pattern_1> {<score_1>}) or          # Part 2: specifying the patterns
       (<pattern_2> {<score_2>}) or          #         of interest.
       ...
       (<pattern_n> {<score_n>})
with threshold <threshold_value>             # Part 3: specifying the threshold.
excluding                                    # Part 4: specifying patterns for
       (<excluding_pattern_1>) or            #         which the matching entities
       (<excluding_pattern_2>) or            #         should be ignored.
       ...
       (<excluding_pattern_m>)
```

- **Part 1 - Specifying the task:** This is the first line of the query in which two main parameters are specified: (1) `<type>` which specifies what types of text spans that are considered for extraction. For instance, we can ask koko to extract *2-grams* or *noun phrases* or *entities*, and (2) `<document>` which specifies to the document that the query will be executed on.
- **Part 2 - Specifying the patterns:** This part is where the patterns that describe the desired entities are listed. Note that each pattern is accompanied by a score which determines how important the pattern is. More precisely, each entity will receive the specified score if it matches the pattern. Perhaps the most simple pattern is what words follow or precede an entity (similar to the pattern in the `miserables.koko` query). We will provide a detailed list of possible pattern in the next section, but here are a few examples (which are self-explanatory):

```
("My name is " x {0.1})
(x "is the number one country in" {0.5})
(str(x) contains "Cafe" {0.2})
```

- **Part 3 - Specifying the threshold:** Based on the specified patterns, each entity will receive a total score which represents how well it matches the patterns. The threshold can be used to prune the entities that have received a low score to obtain high-quality results. Any entity with a score smaller than the threshold will be eliminated from the results.
- **Part 4 - Specifying the excluding patterns:** The patterns specifies in this section are used by Koko to exclude entities from the results. Note that these pattern are *not* accompanied by a score. That's because that any entity that would match these entities (even once) would be eliminated from the results. For example, we can write a pattern to exclude entities that don't contain any spaces which essentially removes single words from the final results. Here, is an updated example of our `miserables.koko` query that uses the excluding patterns:

```
extract "Ents" x from "135-0.txt" if
        ("My name is" x {0.1})
with threshold 0.0
excluding
        (str(x) matches "^\S*$")     # matches if there are no whitespaces
```

which results in:

```
Results:

Entity name                                                Entity score
============================================================
Marius Pontmercy                                           0.200000
Bienvenu Myriel                                            0.100000
Jean Valjean                                               0.100000
Félix Tholomyès                                            0.100000
Madame Thénardier                                          0.100000
```

Patterns Supported by Koko

Koko supports a wide range of patterns which can be combined to make powerful queries. Here, we provide a description of these patterns and show a few examples from each which, as before, we run on the text from Les Miserables. Note that not all patterns are binary, and thus only a subset can be used as excluding patterns. Here, we also clarify which patterns can be used in the excluding section as well.

Entity name token containment

This pattern allows us to check if an entity `contains` a sequence of specified tokens. Note that the tokens should appear in the same exact form and order to be considered a match.

```
# Which characters have the title "Count"?
extract "Ents" x from "135-0.txt" if
        (str(x) contains "Count" {0.1})
with threshold 0.0

Results:

Entity name                                                Entity score
============================================================
```

```
Count                                                    0.300000
Count Lynch                                              0.100000
Count Anglès                                             0.100000
```

Note: This pattern can be used as an excluding pattern.

Entity name substring

This pattern allows us to check if the entity `mentions` a given substring. Note that the `mentions` pattern is different from `contains` since the specified string can appear as a substring in the entity and does not need to be an exact match. The following query would make the difference between the two patterns more clear.

```
# Which entities have "Count" as a substring?
extract "Ents" x from "135-0.txt" if
        (str(x) mentions "Count" {0.1})
with threshold 0.0

Results:

Entity name                                              Entity score
============================================================
Count                                                    0.300000
Counterfeiting                                           0.100000
Count Lynch                                              0.100000
Count Anglès                                             0.100000
```

Note: This pattern can be used as an excluding pattern.

Entity name regular expression

The `matches` pattern allows us to input a regular expression and any entity matching the regular expression would be extracted. The regular expression follow the same format as regular expression in Python (see the documenation).

```
# Which entities end in "ton"?
extract "Ents" x from "135-0.txt" if
        (str(x) matches ".*ton" {0.1})
with threshold 0.2

Results:

Entity name                                              Entity score
============================================================
Wellington                                               1.000000
Danton                                                   1.000000
Washington                                               0.600000
Picton                                                   0.400000
Milton                                                   0.300000
Breton                                                   0.300000
Clinton                                                  0.200000
Newton                                                   0.200000
Mirliton                                                 0.200000
Canton                                                   0.200000
Triton                                                   0.200000
```

Note: This pattern can be used as an excluding pattern.

Strict left context

This pattern allows us to specify what sequence of tokens precede the entities that we are interested in.

```
# Some of the cities mentioned in the book
extract "Ents" x from "135-0.txt" if
        ("the city of" x {0.1})
with threshold 0.0

Results:

Entity name                                                Entity score
============================================================
Paris                                                      0.200000
Angoulême                                                  0.100000
Toryne                                                     0.100000
Voltaire                                                   0.100000
```

Note: This pattern can be used as an excluding pattern.

Strict right context

This is simply the counterpart of the previous pattern. The pattern specifies the tokens that follow the desired entities.

```
# Who is described in the books as "beautiful"?
extract "Ents" x from "135-0.txt" if
        (x "was beautiful" {0.1})
with threshold 0.0

Results:

Entity name                                                Entity score
============================================================
Fantine                                                    0.200000
She                                                        0.200000
```

Note: This pattern can be used as an excluding pattern.

Loose left context

This pattern can be considered as a relaxed-version of the strict left context. That is, we can specify a sequence of tokens that normally precede the desired entities but maybe with a few tokens in between. The pattern would assign the complete score if the pattern appears right before the entity, but if the tokens appear within a distance of `D` tokens then a fraction of the score (i.e., `1/D`) will be assigned to the entity. As a result the score will be smaller if the pattern is found further away from the entity. Also, if the distance between the pattern and the entity (`D`) is more than 10, then it would not be considered a match.

```
# What entities follows "Italian"?
extract "Ents" x from "135-0.txt" if
        ("Italian" near x {0.1})
with threshold 0.0

Results:
```

```
Entity name                                                Entity score
============================================================
German                                                     0.100000
Greek                                                      0.075000
Latin                                                      0.066667
Levantine                                                  0.050000
Here                                                       0.050000
Romance Romance                                            0.033333
Les Misérables                                             0.033333
European                                                   0.025758
Germans                                                    0.025000
Spanish                                                    0.025000
English                                                    0.023377
Spaniard                                                   0.020000
Polish                                                     0.020000
Milan                                                      0.016667
Hebrew                                                     0.014286
Parisian                                                   0.014286
French                                                     0.012500
Mediterranean                                              0.012500
Basque                                                     0.010000
I                                                          0.009091
```

Note: This pattern can **not** be used as an excluding pattern.

Loose right context

This is the counterpart of the previous pattern. The pattern describes what sequence of strings usually follows the entities that we are interested in.

```
# Which entities precede the word "King"
extract "Ents" x from "135-0.txt" if
        (x near "the gun" {0.1})
with threshold 0.05

Results:

Entity name                                                Entity score
============================================================
The                                                        0.238182
This                                                       0.128730
Louis Philippe                                             0.105833
Huguenot                                                   0.100000
King                                                       0.073611
Tuileries                                                  0.057619
Vernon                                                     0.050000
Cotton                                                     0.050000
Agamemnon                                                  0.050000
Megaryon                                                   0.050000
```

Note: This pattern can **not** be used as an excluding pattern.

Semantic left context

This pattern can be considered as a stronger version of the "strict left context" pattern. Similar to the "strict left context" pattern, it searches for entities that follow the specified sequence of tokens, but it allows the pattern to take

different linguistic forms. For instance, imagine that we would like to extract the dishes that are described as delicious in the corpus. We can search for entities that follow the phrase "eating a delicious" to find such entities. But we should also consider entities that follow the phrase "having a tasty". The "semantic left context" pattern enables the user to search for both phrases without having to specify both phrases. This is done automatically by Koko through expanding phrases into other phrases that are semantically close. The score assigned to each entity depends on how (semantically) similar the expanded phrase is to the original phrase in the query. Koko prints out the phrases obtain by expanding the original phrase. This helps the user to better understand the semantics of the query.

```
# Where did wars happen in the story?
extract "Ents" x from "135-0.txt" if
        ("fighting at" ~ x{0.1})
with threshold 0.0

Parsed query: extract "Ents" x from "135-0.txt" if
    (['fighting', 'at'] (1.00) or
     ['fight', 'at'] (0.88) or
     ['fights', 'at'] (0.78) or
     ['fought', 'at'] (0.77) or
     ['battle', 'at'] (0.77) or
     ['battles', 'at'] (0.74) or
     ['battling', 'at'] (0.73) or
     ['combat', 'at'] (0.72) or
     ['fighters', 'at'] (0.72) or
     ['fighter', 'at'] (0.67) or
     ['fighting', '@'] (0.65) or
     ['fight', '@'] (0.57) or
     ['fights', '@'] (0.50) ~ x { 0.10 })
with threshold 0.00

Results:

Entity name                                                Entity score
============================================================
Spire                                                      0.077123
Worms                                                      0.077123
Neustadt                                                   0.077123
Turkheim                                                   0.077123
Alzey                                                      0.077123
Waterloo                                                   0.076818
```

Note: This pattern can **not** be used as an excluding pattern.

Semantic right context

This is the counterpart of the previous query which searches for entities that precede a given phrase or any of its semantically-related expansions. Let's repeat the example we used for the "strict right context" pattern and replace it with this pattern.

```
# Who is described in the books as "beautiful"?
extract "Ents" x from "135-0.txt" if
        (x ~ "was beautiful" {0.1})
with threshold 0.2

Parsed query: extract "../env_test/135-0.txt" Ents from "x" if
    (x ~ ['was', 'beautiful'] (1.00) or
         ['was', 'gorgeous'] (0.90) or
         ['was', 'lovely'] (0.87) or
```

```
         ['was', 'stunning'] (0.81) or
         ['had', 'beautiful'] (0.81) or
         ['was', 'wonderful'] (0.79) or
         ['been', 'beautiful'] (0.78) or
         ...
         ['he', 'beautifully']  (0.51) or
         ['he', 'beautifull']  (0.51) { 0.10 })
with threshold 0.20

Results:

Entity name                                              Entity score
=========================================================================
She                                                      0.454535
Cosette                                                  0.253201
Fantine                                                  0.200000
```

Note: This pattern can **not** be used as an excluding pattern.