Hyperiondev

TASK

NLP – Semantic Similarity

Visit our website

Introduction

WELCOME TO THE NLP – SEMANTIC SIMILARITY TASK!

Texts can be "similar" in many different ways – they can have a similar structure, discuss similar topics, or express similar ideas.

Predicting similarity is useful for building recommendation systems or for detecting plagiarism. For example, when you search for a recipe using Google Search you can get suggestions for different recipes that an algorithm may deem similar to what you searched for to begin with. This lesson will use spaCy to detect similarity within texts.

SIMILARITY WITH SPACY

We can find similarities between words, and even sentences and short passages, using natural language processing (NLP). To do this, we start with a target word (or sentence) and compare it with a list of other words (or sentences) to find semantic similarity. To start, we will just be comparing words, and once we're more confident with the process we can move on to comparing sentences and passages of text.

We will do this by using spaCy, which is able to compare two objects and make a prediction of how similar they are. Let's begin by using simple examples to understand how spaCy categorises similar and dissimilar objects. The similarity is shown as a floating decimal from 0 to 1, with 0 indicating "most dissimilar" and the strength of the similarity increasing all the way up to 1.

For the next code example, you will need a more advanced language model, `en_core_web_md`, which can find similarities and differences better than the original language model we used in the first task, `en_core_web_sm`. If you do not have this model yet please type the line below in your command prompt (terminal):

```
python -m spacy download en_core_web_md
```

Now that you have that installed, type in the following to practise determining similarity with spaCy commands:

```
# Load the medium-sized English model
nlp = spacy.load('en_core_web_md')

# Process the words with the NLP model
word1 = nlp("cat")
word2 = nlp("monkey")
word3 = nlp("banana")

# Print the similarity scores
print(word1.similarity(word2))
print(word3.similarity(word2))
print(word3.similarity(word1))
```

We use the keyword **`similarity`** in the syntax to get the similarity between the words, as seen in the last three lines of code above. When you run the above lines, write a note about what is interesting about the result you get.

A note from the
HyperionDev Team

How Netflix's recommendations system works

When you access the Netflix service, their **recommendations system** helps you find a show or movie to enjoy with minimal effort. Netflix estimates the likelihood that you will watch a particular title in their catalogue based on a number of factors, including:

- your interactions with Netflix (such as your viewing history and how you rated other titles),
- interactions by other Netflix members with similar tastes and preferences as yours,
- information about the titles, such as their genre, categories, actors, release year, etc.

In addition to knowing what you watched previously, Netflix also looks at things like:

- the time of day you watch something,
- the devices you are watching Netflix on, and
- how long you watch a movie or show.

All of these pieces of data are used as inputs that Netflix then processes in their algorithms and uses to make recommendations.

WORKING WITH VECTORS

In the case where you have a series of words and want to compare them all with one another, you can use the format outlined in this section.

We will use two **for** loops to allow us to undertake a comparison of the words. We will first compare one word (**token1**) to all the other "tokens" in the string, and then do the same for the next word (**token2**) and repeat the cycle.

```
# Process the sentence to tokenise the words
tokens = nlp('cat apple monkey banana ')

# Iterate over each pair of tokens
for token1 in tokens:
      for token2 in tokens:
            # Print the text and similarity score
            print(token1.text, token2.text, token1.similarity(token2))
```

The result is seen below:

```
cat cat 1.0
cat apple 0.2036806046962738
cat monkey 0.5929930210113525
cat banana 0.2235882580280304
apple cat 0.2036806046962738
apple apple 1.0
apple monkey 0.2342509925365448
apple banana 0.6646699905395508
monkey cat 0.5929930210113525
monkey apple 0.2342509925365448
monkey monkey 1.0
monkey banana 0.4041501581668854
banana cat 0.2235882580280304
banana apple 0.6646699905395508
banana monkey 0.4041501581668854
banana banana 1.0
```

Did you notice interesting inferences about the similarity in the output above? Let's look at a couple of observations we can make:

- "Cat" and "monkey" seem to be similar because they are both animals.
- Similarly, "banana" and "apple" are similar because they are both fruits.
- Interestingly, "monkey" and "banana" have a higher similarity than "monkey" and "apple". So we can assume that the model already puts together that monkeys eat bananas and that is why there is a significant similarity.
- Another interesting fact is that "cat" does not have any significant similarity with any of the fruits but "monkey" does. So, the model does not explicitly seem to recognise **transitive relationships** in its calculation.

Play around with the code above, adding or replacing words that could show interesting relationships.

WORKING WITH SENTENCES

Many NLP applications need to compute the similarity in meaning between short texts. An obvious use case, as we read earlier regarding Netflix's recommendations system, is suggesting articles, videos, or products to users. Similarly, search engines need to model the relevance of a document to a query, beyond the overlap in words between the two. Also, question-and-answer sites such as Quora and Stack Overflow need to determine whether a question has already been asked before.

This type of text similarity is often computed by first embedding the two short texts and then calculating the **cosine similarity** between them.

We can work on ascertaining similarity between longer sentences using the syntax below:

```
sentence_to_compare = "Why is my cat on the car"

sentences = ["where did my dog go",
"Hello, there is my car",
"I\'ve lost my car in my car",
"I\'d like my boat back",
"I will name my dog Diana"]

model_sentence = nlp(sentence_to_compare)

for sentence in sentences:
      similarity = nlp(sentence).similarity(model_sentence)
      print(sentence + "-" + str(similarity))
```

Instructions

Make sure to run the example code file in this task folder for a walkthrough of an example that compares longer texts.

Practical task 1

Follow these steps:

1. Create a code file called **semantic**. You may use a .py or .ipynb file.
2. Run all the code extracts above.
3. Write a note on what you noticed about the similarities between “cat”, “monkey”, and “banana”, and think of an example of your own.
4. Run the example file with the simpler language model `en_core_web_sm` and write a note on what you notice may be different between it and the `en_core_web_md` model.

Important: Be sure to upload all files required for the task submission inside your task folder and then click "Request review" on your dashboard.

Practical task 2

Let’s build a system that will tell you what to watch next based on the similarity of the description of movies.

1. Create a code file called **movies_task**. You may use a .py or .ipynb file.
2. Read in the **movies.txt** file from the code files for this task. Each separate line is a description of a different movie.
3. Your task is to create a function to return which movies a user would watch next if they have watched *Planet Hulk*. The film has the description: “Will he save their world or destroy it? When the Hulk becomes too dangerous for the Earth, the Illuminati trick Hulk into a shuttle and launch him into space to a planet where the Hulk can live in peace. Unfortunately, Hulk lands on the planet Sakaar where he is sold into slavery and trained as a gladiator.”

4. The function should take in the description as a parameter and return the title of the most similar movie.

Important: Be sure to upload all files required for the task submission inside your task folder and then click "Request review" on your dashboard.

Share your thoughts

Please take some time to complete this short feedback **form** to help us ensure we provide you with the best possible learning experience.