Hyperiondev

TASK

Semantic Similarity (NLP)

Visit our website

Introduction

WELCOME TO THE SEMANTIC SIMILARITY TASK!

Texts can be "similar" in lots of 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 task will use spaCy to detect similarity within texts.

Get in touch

Connect for support

Remember that with our courses, you're not alone! You can contact an expert code reviewer to get support on any aspect of your course.

The best way to get help is to login to Discord at **https://discord.com/invite/hyperdev** where our specialist team is ready to support you.

Our team is happy to offer you support that is tailored to your individual career or education needs. Do not hesitate to ask a question or for additional support!

SIMILARITY WITH SPACY

We can find similarities between words, 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.

SpaCy 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, where 0 indicates 'most dissimilar' and the strength of the similarity increases all the way up to 1.

For the next code example you will need a more advanced language model **'en_core_web_md'** which is able to 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 spaCy commands to determine similarity:

```
import spacy
nlp = spacy.load('en_core_web_md')

word1 = nlp("cat")
word2 = nlp("monkey")
word3 = nlp("banana")

print(word1.similarity(word2))
print(word3.similarity(word2))
print(word3.similarity(word1))
```

The syntax for getting the similarity between the words is by using the keyword **'similarity'** 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 our coding mentor

Jared

How Netflix's Recommendation 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.

*(Source: **https://help.netflix.com/en/node/100639**)*

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.

```
tokens = nlp('cat apple monkey banana ')

for token1 in tokens:
```

```
    for token2 in tokens:
        print(token1.text, token2.text, token1.similarity(token2))
```

The result here is as seen below:

```
cat cat 1.0
cat apple 0.28213844
cat monkey 0.5351813
cat banana 0.28154364
apple cat 0.28213844
apple apple 1.0
apple monkey 0.2929498
apple banana 0.5831845
monkey cat 0.5351813
monkey apple 0.2929498
monkey monkey 1.0
monkey banana 0.45207784
banana cat 0.28154364
banana apple 0.5831845
banana monkey 0.45207784
banana banana 1.0
```

Did you notice interesting inferences about the similarity in the output above? To point out a couple of things:

- 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 although monkey does. So, the model does not explicitly seem to recognise transitive relationships in its calculation.

Try and play around with the code above, adding or replacing words that can show interesting relationships!

WORKING WITH SENTENCES

Many NLP applications need to compute the similarity in meaning between short texts.

An obvious use, as we read earlier regarding Netflix's recommendation system, is suggesting new articles for a user to read or new videos to view on YouTube. Similarly, search engines need to model the relevance of a document to a query, beyond the overlap in words between the two. Question-and-answer sites such as

Quora and StackOverflow 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 + " - ", similarity)
```

Instructions

Make sure to run the **example.py** file in this Task folder for a walk-through of an example comparing longer texts.

Compulsory Task 1

Follow these steps:

- Create a file called **semantic.py** and run all the code extracts above.
- Write a note about what you found interesting about the similarities between cat, monkey and banana and think of an example of your own.
- Run the example file with the simpler language model 'en_core_web_sm' and write a note on what you notice is different from the model 'en_core_web_md'.
- Host your solution on a Git host such as GitLab or GitHub with a Dockerfile and instructions to run included.

    - If it doesn't already, please ensure that your repo includes a file named **requirements.txt** to automate the installation of the project's requirements.
    - Remember to exclude any venv or virtualenv files from your repo.
- Add the link for your remote Git repo to a text file named **semantic_similarity.txt**

Compulsory Task 2

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

- Create a file called **watch_next.py**
- Read in the **movies.txt** file. Each separate line is a description of a different movie.
- Your task is to create a function to return which movies a user would watch next if they have watched *Planet Hulk* with 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 land on the planet Sakaar where he is sold into slavery and trained as a gladiator."
- The function should take in the description as a parameter and return the title of the most similar movie.
- Host your solution on a Git host such as GitLab or GitHub with a Dockerfile and instructions to run included.
    - If it doesn't already, please ensure that your repo includes a file named **requirements.txt** to automate the installation of the project's requirements.
    - Remember to exclude any venv or virtualenv files from your repo.
- Add the link for your remote Git repo to your **semantic_similarity.txt** file.

Things to look out for:

Make sure that you have installed and setup **Python and your chosen editor** correctly. If you are having any difficulties, please feel free to contact our specialist team **on Discord** for support.

Completed the task(s)?

Ask an expert to review your work!

Review work

Rate us

Share your thoughts

HyperionDev strives to provide internationally-excellent course content that helps you achieve your learning outcomes.

Think that the content of this task, or this course as a whole, can be improved, or think we've done a good job?

Click here to share your thoughts anonymously.