TASK

Natural Language Processing with SpaCy

Visit our website

Introduction

WELCOME TO THE NATURAL LANGUAGE PROCESSING TASK

Natural language processing (NLP) is currently one of the largest fields of research in artificial intelligence (AI). In this task, you will be categorising and researching NLP applications. Let's get started!

First, we'll get you started with some background knowledge. Then, you will learn about how to use the spaCy library to classify a piece of text and detect similarity within texts.

AN INTRODUCTION (USING IRON MAN)

Adapted from: Iron Man, Marvel Studios

The image above is from the movie Iron Man. In this movie, the fictional main character, Tony Stark, is a billionaire genius engineer. He creates a suit of armour that is so powerful and advanced that he becomes a superhero nicknamed 'Iron Man'.

Tony has a robot AI inside his mansion called Jarvis. He talks to this robot many times during the film and it helps him perform certain tasks. The picture above is from a scene where Tony has just spoken to Jarvis.

Let's think about this a bit. What is the most advanced technology or most crazy idea in the movie Iron Man?

- Is it the fact that Tony has built a suit made out of metal that he can fly around in?
- Is it the fact that the suit makes Tony so strong that he can shoot missiles from it, reflect bullets, and fly around?
- Is it the fact that Tony is a billionaire engineer that is smart enough to do this by himself?
- Or is it the seemingly small and insignificant fact that Tony can talk to the AI, Jarvis, and Jarvis can understand exactly what he says?

If you didn't know better and had no background in AI, you might think that flying around in a suit shooting missiles is more advanced than a robot understanding the things that Tony says to it, and being able to hold a conversation in the manner of a human assistant. But you'd be wrong!

The fact that Jarvis understands Tony's simple words, "Wake up, daddy's home" and can reply correctly with "Welcome home, sir" is a massive technological feat for AI. While creating a superhero suit and soaring through the skies may captivate our imagination, it pales in comparison to the vast expanse of NLP, the primary focus of AI research today. This is evident in recent developments, where virtual assistants like Siri, Alexa, and Google Assistant have revolutionised technology interaction. These intelligent, NLP-powered, voice-activated systems understand user commands, answer questions, and perform tasks. Building upon their success, the field of NLP has witnessed the rise of large language models (LLMs) like ChatGPT. These models utilise deep learning techniques to generate human-like text and perform various NLP tasks such as text generation, classification, question answering, and translation. Researchers eagerly anticipate the further evolution of LLMs, with future models expected to possess even more powerful capabilities.

Continuing the path of advancements in NLP, technologies like ChatGPT are at the forefront of text similarity applications. As explained previously, these systems aim to understand and provide accurate responses to human inquiries, bringing us closer to the AI companion Jarvis from Iron Man. However, the power of ChatGPT extends beyond human communication. From an enterprise perspective, ChatGPT is poised to revolutionise multiple business sectors. Currently, its applications include copywriting, customer service, compliance, and IT, offering persuasive content, personalised engagement, regulatory support, and technical assistance. As ChatGPT and similar technologies evolve, they will open new avenues for problem-solving and decision-making across industries.

Extra resource

Read more about **ChatGPT**, including the caveats and benefits of the technology. **Try it out** for yourself on the OpenAI website.

NATURAL LANGUAGE PROCESSING

In order to start thinking about creating Jarvis in real life (i.e. a robot that can understand what we say and act on it or even just reply correctly) we need many things.

We call programs like Jarvis, that converse with humans in natural language, conversational agents or dialogue systems. Natural languages are languages humans use to talk to each other; 'formal languages' are programming languages like Python or Java.

Jarvis must be able to recognise words from an audio signal and generate an audio signal from a sequence of words. These tasks of speech recognition and speech synthesis require knowledge about phonetics and phonology: how words are pronounced in terms of sequences of sounds and how each of these sounds is created acoustically. Pronouncing variations of words correctly (such as plurals, contractions) requires knowledge about morphology – the way words break down into component parts that carry meanings.

What if we asked Jarvis, "How many University of Cambridge students are in the Math-130 class by the end of the day?" Jarvis needs to know something about lexical semantics – the meaning of all the words (e.g. 'class' or 'students') and compositional semantics (what exactly makes a student a 'University of Cambridge student' and not another type of student?). What does 'end' mean when combined with 'the day'? Jarvis needs to know about the relationship of the words to each other – how does Jarvis know that 'by the end of the day' refers to a time and doesn't refer to something like 'the book that is written by the author Neil Gaiman'? Humans know this automatically, but how can computers learn this?

How does Jarvis know that when Tony says 'Daddy's home', Tony is actually talking about himself? Jarvis knows this because he says 'Welcome home, sir', so clearly Jarvis understood that somehow. This knowledge about the kind of actions that speakers intend by their use of sentences is pragmatic or dialogue knowledge. To summarise, Jarvis needs the following knowledge of language:

- Phonetics and Phonology – knowledge about linguistic sounds
- Morphology – knowledge of the meaningful components of words
- Syntax – knowledge of the structural relationships between words
- Semantics – knowledge of meaning
- Pragmatics – knowledge of the relationship of meaning to the goals and intentions of the speaker
- Discourse – knowledge about linguistic units larger than a single utterance

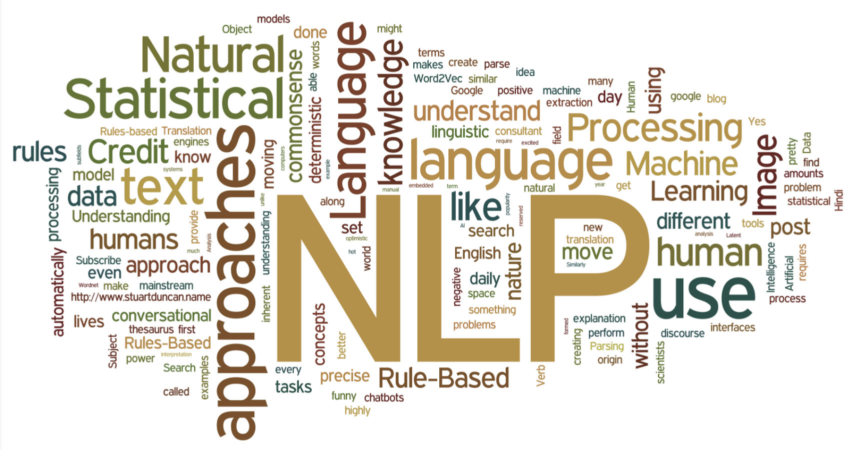


Image source: ***Deep Learning Analytics***

Ambiguity

Now, what if Tony was telling Jarvis a story about his female assistant who had annoyed him? As it happened, Tony threw a piece of paper at her and she ducked to avoid it. Describing the incident, Tony says the following sentence to Jarvis:
“I made her duck”.

This simple sentence has the following meanings. The correct meaning in Tony’s story is in bold:

- I cooked waterfowl for her.
- I cooked waterfowl belonging to her.
- I created the (plaster?) duck she owns.
- **I caused her to quickly lower her head or body (to avoid something).**
- I waved my magic wand and turned her into a common waterfowl.

But how can we make Jarvis smart enough to know this? There are many ambiguities in this sentence because the word ‘duck’ can be a verb (move your head down) or a noun (a waterfowl), and ‘her’ can mean the woman, or can refer to the fact that the duck belongs to her.

What about hearing? Say the word ‘I’ out loud. How does Jarvis know that this word isn’t actually ‘eye’? What about ‘made’? It sounds just like ‘maid’! Poor Jarvis!
We must use complicated models and algorithms as ways to resolve or disambiguate (remove) these ambiguities.

Siri and Alexa, as prime examples, utilise advanced techniques to effectively manage ambiguity in user queries. By analysing the conversation's context, including prior interactions and user preferences, they can deduce the most probable interpretation of ambiguous queries. Moreover, they leverage machine learning algorithms to continually enhance their comprehension and disambiguation skills by learning from vast datasets. Whenever ambiguity arises, Siri and Alexa may seek clarification through follow-up questions, allowing them to gather more details and deliver precise responses. This interactive approach facilitates their understanding refinement, ensuring users receive the desired information or assistance with greater accuracy.

Part of speech tagging

Deciding whether ‘duck’ is a verb or a noun is known as part of speech (POS) tagging. Verbs and nouns are different ‘parts of speech’ and we ‘tag’ a word in a sentence by assigning it one part of speech that we think is correct for the context or sentence it has been used in.

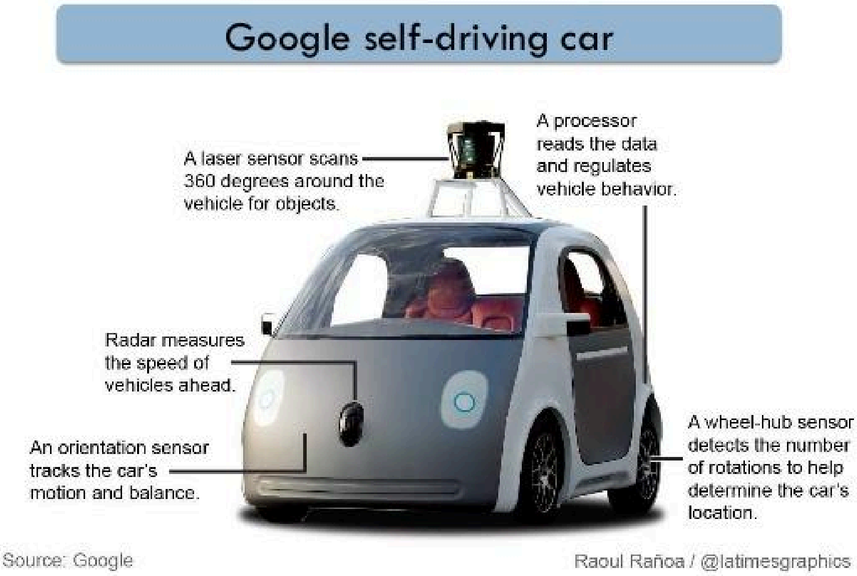
In a phrase like “The old man and the boat”, the task of tagging the correct parts of speech may involve the following:

***The*:** tag as ‘determiner’
***old*:** tag as ‘adjective’
***man*:** tag as ‘noun’ (or verb?)’
***the*:** tag as ‘determiner’
***boat*:** tag as ‘noun’

Sometimes, probabilities are used to decide this. For example, the probability is higher that the word ‘man’ above means the noun ‘an older male person’ than the verb ‘put someone there’ (e.g. “The enemy is here! Man the cannons!”).

As you can see, NLP is a huge field and extremely important to AI. NLP also includes the study of things like machine translation which is the same as the technology behind Google Translate – automatically translating between two languages. Google Search uses NLP techniques in order to understand your searches faster, and this is what makes Google Search more accurate, more reliable, and faster than any other search method on the Internet. NLP is also used in Gmail in order to identify spam mail and delete it.

NLP research is one of the main reasons Google is so successful. The probabilistic, machine learning techniques that Google applies to NLP can be applied to other AI tasks such as creating driverless cars.


Google self-driving car
A laser sensor scans 360 degrees around the vehicle for objects.
A processor reads the data and regulates vehicle behavior.
Radar measures the speed of vehicles ahead.
An orientation sensor tracks the car's motion and balance.
A wheel-hub sensor detects the number of rotations to help determine the car's location.
Source: Google
Raoul Rañoa / @latimesgraphics

We hope you can see how many fields in AI have to do with probability. This is because it is the only way we can deal with ambiguity, in order to enable robots/AI programs to make the best decisions!

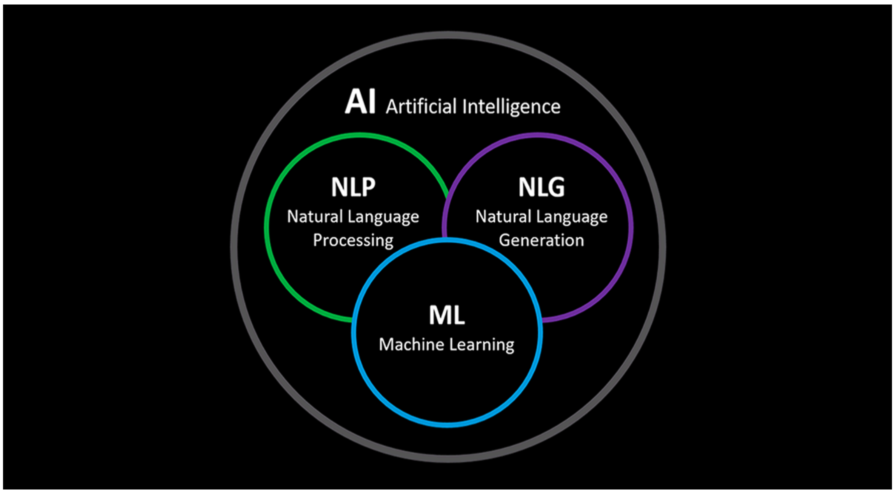

AI Artificial Intelligence
NLP
Natural Language Processing
NLG
Natural Language Generation
ML
Machine Learning

Image source: ***Conversica***

MACHINE LEARNING

NLP is a big part of AI because a large portion of NLP has to do with training computers or AI programs. These AI programs identify patterns and use probabilities to make informed decisions.

This is known as machine learning and is a massive field of research. Facebook uses machine learning to try to recommend friends to you; Amazon uses it to recommend items to buy; Google Image Search uses machine learning techniques to identify patterns in pixels to try to find similar images; and Google has designed driverless cars to act according to their environment by integrating many different aspects of machine learning. The list is endless.

Solving POS tagging

One example is to solve the problem of POS tagging as explained earlier. We can give a program a big set of tagged words (training data) and then give it a new sentence where it must try to tag words using the information learnt from the training data.

The University of Pennsylvania had the first NLP research program to ever take a large corpus (body) of words and tag each and every one by hand to create a **treebank**. A program was then activated and slowly learnt how to tag words and the probability that a certain word appeared with a certain tag in a certain context. Ever since then, AI runs on huge sets of training data. The larger the dataset, the more accurate we can train an AI program to be. Training sets utilised for ChatGPT and other LLMs generally encompass substantial volumes of text data. To illustrate, the training set for ChatGPT-3 was an extensive collection of 570 GB of text, drawn from diverse sources such as online content, books, and articles.. This vast corpus facilitates the model's acquisition of patterns, syntax, and semantic relationships inherent in human language, empowering it to generate responses that are coherent and contextually appropriate.

In POS tagging we try to tag words with the correct POS tag so that we can then parse the sentence correctly. **Parsing** is the formal term for ‘putting a sentence together in the right way’ so that it can be ‘understood’. We will talk more about this later.

Text classification

We also use machine learning to try to classify texts. For example, Gmail classifies emails as either ‘spam’ or ‘not spam’. It uses machine learning by having trained an

AI program on a set of already ‘classified’ emails (examples of non-spam and spam emails). Then when the AI program sees a new incoming email, it can use its prior knowledge or ‘training’ to classify the new email correctly.

This task will have an example of how we can use machine learning to get a program to identify positive or negative tweets – similar to the problem of identifying spam mail through machine learning – but first, we need to start with spaCy.

STARTING WITH SPACY

SpaCy is a Python NLP library specifically designed with the goal of being a useful library for implementing production-ready systems. It is particularly fast and intuitive, making it a top contender for beginners in NLP. Before doing anything, you need to have spaCy installed, as well as its English language model.

To install spaCy, first ensure that the pip, setuptools and wheel are up to date by running the following command:

```
pip install -U pip setuptools wheel
```

Thereafter, run the following command on your terminal to install spaCy and it’s dependencies:

```
pip install -U spacy
```

SpaCy is not very useful without at least one language model. The model allows you to process different languages. SpaCy currently has 23 language models which include French, Dutch, Spanish, and of course English. You can install more than one model, or even install a multi-language model all at once. SpaCy's models can be installed as Python packages. This means that they're a component of your application, just like any other module. Models can be installed from a download URL or a local directory, manually or via pip. Their data can be located anywhere on your file system.

Although spaCy and its English model are installed and set up for you, you will still need to import, load, and use spaCy. We’ll walk you through the commands to use below.

Firstly, go into your Python console and import spaCy as below:

```
python
import spacy
```

Now we will load the model and assign it to a variable.

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

The input to NLP will be a simple stream of Unicode characters (typically UTF-8). Basic processing will be required to convert this character stream into a sequence of lexical items (words, phrases, and syntactic markers) which can then be used to better understand the content.

For spaCy, you can do this by passing the string through the language model you imported at the beginning of the script. Remember we named our model `nlp` so to process our string in preparation for spaCy manipulation, we use the code below:

```
doc = nlp("this is a test sentence")
print([(w.text, w.pos_) for w in doc])
```

Now that we are able to process a string, we can do more complicated stuff such as tokenization, lemmatization, and named entity recognition. Let’s take a closer look at these.

Tokenization

Tokenization is a foundational step in many NLP tasks. Tokenizing text is the process of splitting a piece of text into words, symbols, punctuation, spaces, and other elements, thereby creating “tokens”.

Lemmatization

A related task to tokenization is lemmatization. Lemmatization is the process of reducing a word to its base form, its ‘mother word’, if you like. Different uses of a word often have the same root meaning. For example: practise, practised, and practising all essentially refer to the same thing. It is often desirable to standardise words with similar meanings to their base form.

Named entity recognition

Named entity recognition is the process of classifying named entities found in a text into predefined categories, such as persons, places, organisations, dates, etc. SpaCy uses a statistical model to classify a broad range of entities, including persons, events, works-of-art, nationalities, and religions. See the **spaCy documentation** for more information.

Examples of tokenization, lemmatization, and named entity recognition are in the **nlp_example.py** file provided, so open it up and take a look!

SIMILARITY WITH SPACY

We can find similarities between words, and even sentences and short passages, using natural language processing. 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, and
- 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.

(Netflix, n.d.)

WORKING WITH VECTORS

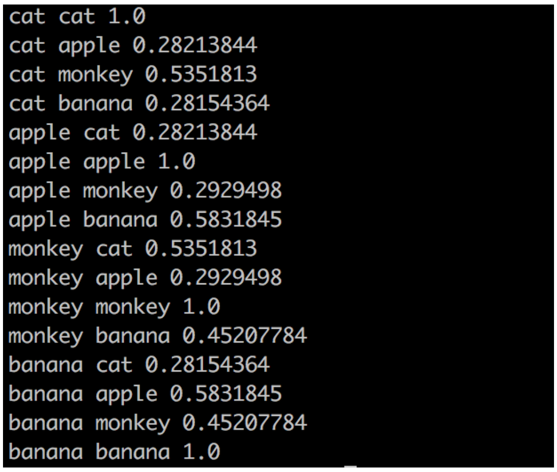
In the case where you have a series of words and want to compare them all with one another, you can use the format outlined in this section.

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.
- Banana and apple are similar because they are both fruits.
- 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)
```

NATURAL LANGUAGE PROCESSING APPLICATIONS

NLP applications can be classified into a number of categories based on what they do. Let's have a brief look at these categories.

Language translation

The volume of information being put up online is growing at an incredible rate, but due to language barriers, not everything is accessible to everyone. Language translation helps us conquer these language barriers by facilitating the translation of technical manuals, support content, or websites quickly and inexpensively. The challenge with language translation technologies is not in translating words, but in understanding the meaning of sentences to provide a true translation. If you've ever watched a movie in one language with subtitles in another language where you have been able to understand both languages, you will realise how prone to error the process of translation, even using human experts can be. Sometimes the subtitles are outright wrong, and sometimes, although the words are translated accurately, the nuance of the meaning is lost or miscommunicated. Really effective NLP language translation is a challenging area in which exciting progress is being made.

Text classification

One of the applications of NLP that we experience on a daily basis is the text classification in our email folders: by using predefined categories, we can organise our spam folders and inbox so that we can access relevant emails or messages more efficiently.

Automatic summarisation

When working with huge amounts of information (like articles, books, and websites), it can be extremely useful to be able to shorten these pieces into condensed forms that only show the pieces of information that are most useful to you. This is what automatic summarisation is about. According to Expert System (2020): "Automatic summari[s]ation is relevant not only for summari[s]ing the meaning of documents and information, but also for understanding the emotional meanings inside the information, such as in collecting data from social media."

Sentiment analysis

Similarly to how we can infer someone's meaning from their tone, sentiment analysis allows us to detect the emotion behind a piece of text using NLP. This is particularly useful for large companies who want to know what the general sentiment people hold towards their company is. By analysing articles and write-ups about their company using sentiment analysis, they can gain a fairly accurate idea of how people feel about them based on the language used when they are discussed.

Question answering

This is an application of NLP that has come a long way in a short space of time. Simply put, these systems allow a computer to answer a question posed by a human. Siri and Okay Google are well-known voice-based question answering systems, but text-based systems can now be seen on almost every banking or online shopping site in the form of a chatbox. ChatGPT is a relatively advanced implementation of a question-answering model.

Practical Task 1

In this task, you will develop a Python program that performs sentiment analysis on a dataset of product reviews.

Follow these steps:

- **Download a dataset of product reviews**: **Consumer Reviews of Amazon Products**. You can save it as a CSV file, naming it: amazon_product_reviews.csv.
- **Create a Python script**, naming it: sentiment_analysis.py. Develop a Python script for sentiment analysis. Within the script, you will perform the following tasks using the spaCy library:
  1. **Implement a sentiment analysis model using spaCy**: Load the `en_core_web_sm` spaCy model to enable natural language processing tasks. This model will help you analyse and classify the sentiment of the product reviews.
  2. **Preprocess the text data**: Remove stopwords, and perform any necessary text cleaning to prepare the reviews for analysis.
     2.1. To select the 'review.text' column from the dataset and retrieve its data, you can simply use the square brackets notation. Here is the basic syntax:

```
reviews_data = dataframe['review.text']
```

This column, 'review.text,' represents the feature variable containing the product reviews we will use for sentiment analysis.

2.2. To remove all missing values from this column, you can simply use the `dropna()` function from Pandas using the following code:

```
clean_data = dataframe.dropna(subset=['reviews.text'])
```

3. **Create a function for sentiment analysis**: Define a function that takes a product review as input and predicts its sentiment.

4. **Test your model on sample product reviews**: Test the sentiment analysis function on a few sample product reviews to verify its accuracy in predicting sentiment.

5. **Write a brief report or summary in a PDF file**: sentiment_analysis_report.pdf that must include:
   5.1. A description of the dataset used.
   5.2. Details of the preprocessing steps.
   5.3. Evaluation of results.
   5.4. Insights into the model's strengths and limitations.

Additional instructions:

- Some helpful guidelines on cleaning text:
  - To remove stopwords, you can utilise the `.is_stop` attribute in spaCy. This attribute helps identify whether a word in a text qualifies as a stop word or not. Stopwords are common words that do not add much meaning to a sentence, such as "the", "is", and "of". Subsequently, you can then employ the filtered list of tokens or words (words with no stop words) for conducting sentiment analysis.
  - You can also make use of the `lower()`, `strip() and str()` methods to perform some basic text cleaning.
- You can use the spaCy model and the `.sentiment` attribute to analyse the review and determine whether it expresses a positive, negative, or neutral sentiment. To use the `.polarity` attribute, you will need to install the **TextBlob library**. To install the TextBlob library and download additional data, run the following commands **from the terminal**:

Python

```
# Install spacytextblob
pip install spacytextblob

# Download additional data for TextBlob
Python -m textblob.download_corpora
```

- Once you have installed TextBlob, you can use the .sentiment and .polarity attribute to analyse the review and determine whether it expresses a positive, negative, or neutral sentiment. When you are ready to determine sentiment add the code below to your `sentiment_analysis.py` script.

Python

```
# Using the polarity attribute
polarity = doc._.blob.polarity

# Using the sentiment attribute
sentiment = doc._.blob.sentiment
```

FYI: The underscore in the code just above is a **Python convention** for naming private attributes. Private attributes are not meant to be accessed directly by the user, but can be accessed through public methods.

- You can use the `.polarity` attribute to measure the strength of the sentiment in a product review. A polarity score of 1 indicates a very positive sentiment, while a polarity score of -1 indicates a very negative sentiment. A polarity score of 0 indicates a neutral sentiment.
- You can also use the `similarity()` function to compare the similarity of two product reviews. A similarity score of 1 indicates that the two reviews are more similar, while a similarity score of 0 indicates that the two reviews are not similar.
    - Choose two product reviews from the 'review.text' column and compare their similarity. To select a specific review from this column, simply use indexing, as shown in the code below:

      ```
      my_review_of_choice = data['reviews.text'][0]
      ```

    - The above code retrieves a review from the 'review.text' column at index 0. You can select two reviews of your choice using indexing. However, please be cautious not to use an index that is out of bounds, meaning it exceeds the number of data points or rows in our dataset.

- Include informative comments that clarify the rationale behind each line of code.

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.

ACKNOWLEDGEMENTS

We worked with the University of Edinburgh, who created **NLTK** and contributed to **spaCy**, to create this task. This task uses content adapted with permission from the University of Edinburgh's Informatics department, one of the leading NLP research departments in the world. You can see their related course content at **Informatics 2A: Processing Formal and Natural Languages**.

REFERENCES

Honnibal, M., & Montani, I. (2017). *spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing.* **https://www.google.com/url?q=https://sentometrics-research.com/publication/72/&sa=D&source=docs&ust=1709558857059237&usg=AOvVaw2V4KDB-mTYWz0CxO33tach**

Biggs, C. (2022). 20 Grammatically-Correct Sentences You Won't Read Right The First Time. Apartment Therapy. Retrieved from **https://www.apartmenttherapy.com/garden-sentences-262915**

Expert.ai Team. (2020). Natural language processing applications. Expert.ai. Retrieved from **https://expertsystem.com/natural-language-processing-applications/**

Netflix. (n.d.). *How Netflix's recommendations system works.* **https://help.netflix.com/en/node/100639**