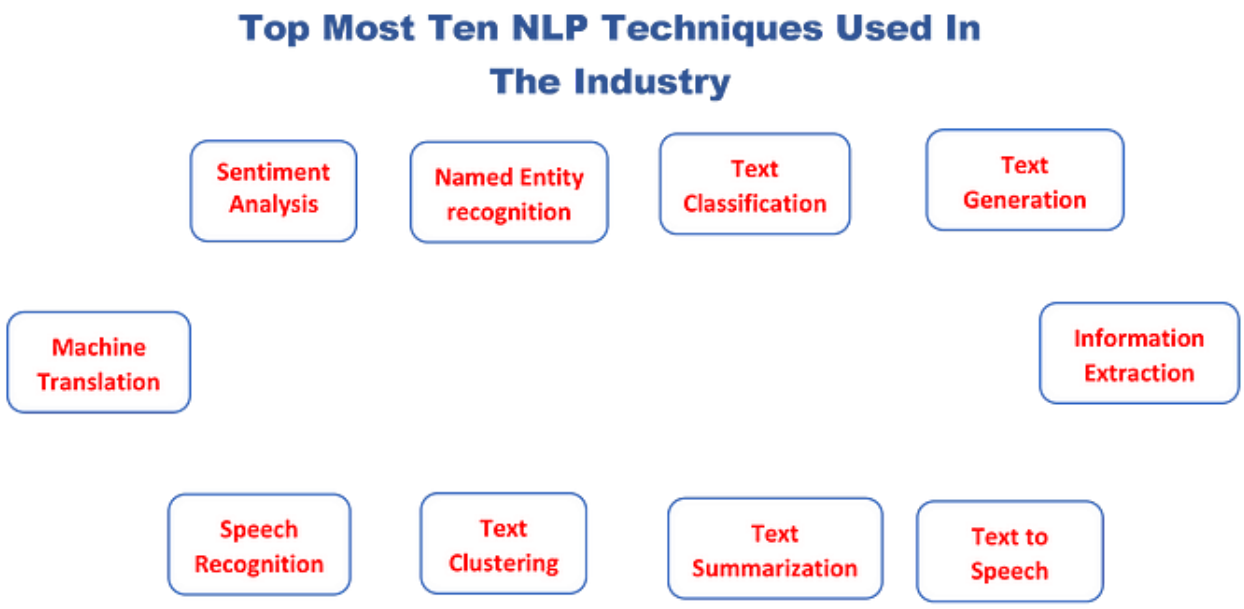

Self made image

Sentiment Analysis

Sentiment analysis is the process of determining the emotional tone behind a piece of text, such as a tweet, a product review, or customer feedback.

The goal of sentiment analysis is to classify the text as positive, negative, or neutral. For example, if a customer writes a review of a product saying, “I love this product, it’s amazing”, the sentiment analysis algorithm would classify the text as positive. Sentiment analysis is widely used in industries such as e-commerce, social media, and customer service to gain insights into customer opinions and preferences.

One way to perform sentiment analysis is by using a pre-trained model such as the one provided by the `nltk` library in python. Here's an example of how to use the `nltk` library to classify the sentiment of a piece of text as positive, negative, or neutral

```
import nltk
from nltk.sentiment import SentimentIntensityAnalyzer

# Initialize sentiment analyzer
sia = SentimentIntensityAnalyzer()

# Define text to be analyzed
text = "I love this product, it's amazing"

# Get sentiment score
sentiment_score = sia.polarity_scores(text)

# Print sentiment score
print(sentiment_score)
```

```
 sentiment_score[] > :     () sentiment_score[] < -:     ():     ()
```

This example uses the **`SentimentIntensityAnalyzer`** class from the **`nltk.sentiment`** module to analyze the sentiment of the text "I love this product, it's amazing". The **`polarity_scores()`** method returns a dictionary containing the sentiment scores for the text, with the 'compound' score is a value between -1 and 1, where -1 is negative, 1 is positive and 0 is neutral. Based on the compound score, we can classify the sentiment as positive, negative, or neutral.

Note that this is a simple example and in practice, Sentiment analysis is an area that require a lot of fine-tuning and adjusting to achieve good results. A pre-trained model might not perform well on certain types of texts (e.g. sarcasm) and might require additional fine-tuning or pre-processing steps to improve its performance.

Named Entity Recognition (NER)

Named Entity Recognition (NER) is a technique used to extract entities such as people, organizations, and locations from unstructured text. One way to perform NER is by using pre-trained models, such as the one provided by the **`spacy`** library in Python. Here's an example of how to use the **`spacy`** library to extract named entities from a piece of text

```
import spacy

# Load the pre-trained model
nlp = spacy.load("en_core_web_sm")

# Define text to be analyzed
text = "Barack Obama visited the White House today"

# Process the text with the model
doc = nlp(text)

 ent  doc.ents:    (ent.text, ent.label_)
```

example uses the **`en_core_web_sm`** model from **`spacy`** to analyze the text "Barack Obama visited the White House today". The **`ents`** attribute of the processed text returns an iterator of named entities, and each entity has the attributes **`text`** and **`label_`** that represent the text and the label of the entity respectively. In this example, the output will be

```
Barack Obama PERSONWhite House FAC
```

It shows that “Barack Obama” is a person, and “White House” is a facility.

There are multiple pre-trained models available in `spacy` for different languages, and some of them are more accurate than others. In addition, Named Entity Recognition is an area that require a lot of fine-tuning and adjusting to achieve good results. A pre-trained model might not perform well on certain types of texts (e.g. technical texts) and might require additional fine-tuning or pre-processing steps to improve its performance.

Text Classification

Text classification is the process of automatically categorizing text into predefined classes or categories. For example, a text classification algorithm might be used to classify emails as spam or not spam, or to categorize news articles by topic. Text classification is used in a variety of applications, including natural language processing, information retrieval, and machine learning.

Here is an **example** of text classification using the Python library scikit-learn. This example uses the 20 Newsgroups dataset, which contains texts from 20 different newsgroups. The goal is to train a classifier to predict the newsgroup a text belongs to based on its content.

```
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score

# Load the 20 Newsgroups dataset
newsgroups_train = fetch_20newsgroups(subset='train')
newsgroups_test = fetch_20newsgroups(subset='test')

# Transform the texts into TF-IDF vectors
vectorizer = TfidfVectorizer()
X_train = vectorizer.fit_transform(newsgroups_train.data)
X_test = vectorizer.transform(newsgroups_test.data)
y_train = newsgroups_train.target
y_test = newsgroups_test.target

# Train a Multinomial Naive Bayes classifier
clf = MultinomialNB()
clf.fit(X_train, y_train)

# Predict the newsgroup of the test texts
y_pred = clf.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)(.(accuracy * ))
```

This code will load the 20 Newsgroups dataset and split it into training and test sets. Then it will transform the texts into numerical representation using TfidfVectorizer and train a Multinomial Naive Bayes classifier using the training set. Finally, it will use the trained classifier to predict the newsgroup of the test texts and evaluate the classifier's accuracy.

Machine Translation

Machine translation is the process of automatically translating text from one language to another. For example, a machine translation algorithm might translate a news article from Spanish to English. Machine translation is used in a variety of industries, including e-commerce, international business, and government.

Here is an **example** of using the OpenNMT library to translate text from English to French:

```
from opennmt import tokenizers
from opennmt import models
import torch
```

```
# Tokenize the source and target text.
source_tokenizer = tokenizers.new("text", "en")
source_text = "Hello, how are you?"
source_tokens = source_tokenizer.tokenize(source_text)

target_tokenizer = tokenizers.new("text", "fr")
target_text = "Bonjour, comment vas-tu?"
target_tokens = target_tokenizer.tokenize(target_text)

# Build the translation model.
model = models.Transformer(
    source_vocab_size=len(source_tokenizer.vocab),
    target_vocab_size=len(target_tokenizer.vocab),
    num_layers=6,
    hidden_size=512,
    dropout=0.1,
    attention_dropout=0.1,
    relu_dropout=0.1)
model.eval()

# Convert the tokens to a tensor.
source_tokens = torch.tensor(source_tokenizer.encode(source_text)).unsqueeze(0)

# Generate a translation.
with torch.no_grad():
    log_probs, _, _ = model(source_tokens, None, None)
    tokens = log_probs.argmax(-1)

translation = target_tokenizer.decode(tokens[])(translation)
```

This code will output: “Bonjour, comment vas-tu?”

Please note that this is a very simple example and will not work out of the box as it requires a pre-trained model to be loaded. Also, this example uses a small dataset as input, and a pre-trained model might not be available for the specific case. for more about machine learning follow here

Text Summarization

Text summarization is the process of automatically generating a condensed version of a longer piece of text. For example, a text summarization algorithm might take a long news article and generate a shorter summary of the main points. Text summarization is used in a variety of applications, including natural language processing, information retrieval, and machine learning.

Please note that this is a very simple example and will not work out of the box as it requires a pre-trained model to be loaded. Also, this example uses a small dataset as input, and a pre-trained model might not be available for the specific case.

```
from gensim.summarization import summarize
```

```
text = "In computing, stop words are words which are filtered out before or after processing of text. Though stop words usually refer to the most common words in a language, there is no single universal list of stop words used by all natural language processing tools, and indeed not all tools even use such a list. Some tools specifically avoid removing these stop words to support phrase search."

(summarize(text, ratio=))
```

This code will output a summarized version of the text, keeping only the 20% most important sentences: “Some tools specifically avoid removing these stop words to support phrase search.”

You can adjust the ratio parameter to change the amount of text that is summarized, or use the `word_count` parameter to specify the number of words to include in the summary.

Information Extraction

Information extraction is the process of extracting structured data from unstructured text. For example, an information extraction algorithm might extract product information, such as price and availability, from an e-commerce website. Information extraction is used in a variety of industries, including e-commerce, finance, and healthcare, to extract structured data from unstructured text.

Here is an **example** of information extraction using Python and the Natural Language Toolkit (NLTK) library:

```
import nltk
from nltk import word_tokenize, pos_tag, ne_chunk

# Sample text
text = "Barack Obama was the 44th President of the United States, serving from 2009 to 2017."

# Tokenize the text
tokens = word_tokenize(text)

# POS tagging
tagged_tokens = pos_tag(tokens)

entities = ne_chunk(tagged_tokens)print(entities)
```

The above code first tokenizes the text into individual words, then performs POS tagging to identify the part of speech for each word, and finally performs named entity recognition to identify entities such as people, organizations, and locations.

The output of the `ne_chunk` function is a tree structure that can be further processed to extract the entities of interest.

```
(S  (PERSON Barack/NNP)  Obama/NNP  was/VBD  the/  th/JJ  (ORGANIZATION President/NNP)  of/IN  the/  (GPE United/NNP States/NNPS)  ,/,  serving/VBG  /IN  /CD  /  /CD  ./.)
```

> Note that the above example is a very simple example, and in real-world cases, you will have to work with a lot of pre-processing and fine-tuning of the models.

Text Generation

Text generation is the process of automatically generating text, such as creating product descriptions or writing news articles. For example, a text generation algorithm might take a product image as input and generate a product description. Text generation is used in a variety of industries, including e-commerce, marketing, and content creation.

Here is an **example** of text generation using the **GPT-2** model in the Python library, Hugging Face's transformers:

```
from transformers import GPT2Tokenizer, GPT2LMHeadModel

# Load the GPT-2 model and tokenizer
tokenizer = GPT2Tokenizer.from_pretrained("gpt2")
model = GPT2LMHeadModel.from_pretrained("gpt2")

# Define the prompt and generate text
prompt = "Once upon a time in a land far, far away"
generated_text = model.generate(input_ids=tokenizer.encode(prompt))

generated_text = tokenizer.decode(generated_text)print(generated_text)
```

This code will generate text based on the provided prompt "Once upon a time in a land far, far away" using the GPT-2 model. The generated text will be printed on the console.

Please note that you may require internet connection to download the pre-trained model and also a powerful GPU to generate the text.

Text Clustering

Text clustering is the process of grouping similar text documents together. For example, a text clustering algorithm might take a collection of news articles and group them into categories such as "sports", "politics", and "entertainment". Text clustering is used in a variety of applications, including natural language processing, information retrieval, and machine learning.

```
import nltk
from nltk import word_tokenize, pos_tag, ne_chunk

# Sample text
text = "Barack Obama was the 44th President of the United States, serving from
2009 to 2017."

# Tokenize the text
tokens = word_tokenize(text)

# POS tagging
tagged_tokens = pos_tag(tokens)

entities = ne_chunk(tagged_tokens)(entities)
```

The above code first tokenizes the text into individual words, then performs POS tagging to identify the part of speech for each word, and finally performs named entity recognition to identify entities such as people, organizations, and locations.

The output of the `ne_chunk` function is a tree structure that can be further processed to extract the entities of interest.

Speech Recognition

Speech recognition is the process of converting spoken words into written text. For example, a speech recognition algorithm might be used in a voice-controlled system, such as a virtual assistant, to transcribe spoken commands into text that can be understood by a computer. Speech recognition is used in a variety of industries, including healthcare, finance, and customer service.

There are many libraries and frameworks available for speech recognition in various programming languages. Here is an example of how to use the Speech Recognition library in Python to transcribe speech from a microphone:

```
import speech_recognition as sr

# create a recognizer object
r = sr.Recognizer()

# create a microphone object
mic = sr.Microphone()

 mic  source:    r.adjust_for_ambient_noise(source)    audio = r.listen(source)
transcribed_text = r.recognize_google(audio)    (transcribed_text)
```

This example uses the `recognize_google()` function, which utilizes the Google Web Speech API to transcribe speech. Other options for transcribing speech include using the `recognize_sphinx()` function (which uses the CMU Sphinx engine) or the `recognize_wit()` function (which uses the Wit.ai API).

You can also use this library to recognize speech from a file:

```
with sr.AudioFile() as :    audio = r.record()    transcribed_text =
r.recognize_google(audio)    (transcribed_text)
```

Note that you need to have internet connection to use Google Web Speech API, and you may need to setup the credentials and install some additional package depend on the transcribing engine you choose.

Text-to-Speech (TTS)

Text-to-speech (TTS) is a technology that converts written text into spoken words. It is commonly used in applications such as speech synthesis for the visually impaired, voice assistants, and automated customer service systems.

TTS systems use a combination of techniques, such as natural language processing and machine learning, to produce realistic-sounding speech. Some examples of TTS software include Google Text-to-Speech, Amazon Polly, and Apple’s Siri.

Here is an **example** of using the gTTS (Google Text-to-Speech) library in Python to convert text to speech:

```
from gtts import gTTS
import os

text = "Hello, this is an example of text to speech using the gTTS library in
Python."

# Language in which you want to convert
language = 'en'

# Passing the text and language to the engine,
# here we have marked slow=False. Which tells
# the module that the converted audio should
# have a high speed
myobj = gTTS(text=text, lang=language, slow=False)

# Saving the converted audio in a mp3 file named
# welcome
myobj.save("welcome.mp3")

os.system()
```

This code uses the gTTS library to convert the text “Hello, this is an example of text to speech using the gTTS library in Python.” to speech and save it to an mp3 file called “welcome.mp3”.

The last line os.system(“mpg321 welcome.mp3”) plays the mp3 file using the command line tool mpg321. If you don’t have mpg321 installed in your system, you could use other player to play the mp3 file.

Follow to given link for advance NLP AnkushMulkar/Natural-Language-processing (github.com)

Click below links to know more about “*Ankush Mulkar*”