4222-SURYA GROUP OF INSTITUTION

VIKIRAVANDI-605 652.

NAAN MUDHALVAN PROJECT

CREATE CHATBOT IN PYTHON

PHASE 5 : Project Documentation & Submission

SUBJECT CODE-SB3001

COURSE NAME – EXPERIENCE BASED PRACTICAL LEARNING

Presented by:

S.ABUTHA KIR

REGNO:422221106301

ECE DEPARTMENT

Chatbot In Python

Introduction:

Building on the Proposal Development Workshop that you have just attended as well as the EOI your project team submitted, CPWF is now asking you to complete a formal project proposal as one of five projects that make up the Volta/Limpopo BDC. Background information on the BDC, the CPWF in general.

ML used in chatpot in pythonL:

- NLP(Natural languages processing)
- Transformer-based deep learning algorithm

Scope of chatbot:

Chatbots can provide instant assistance to customers, which can help reduce wait times and improve customer satisfaction. In the future, chatbots may become even more sophisticated and be able to handle more complex customer service interactions. A chatbot is a software or computer program that simulates human conversation or "chatter" through text or voice interactions.

Users in both business-to-consumer (B2C) and business-to-business (B2B) environments increasingly use chatbot virtual assistants to handle simple tasks.

Why chatbot:

Organizations looking to increase sales or service productivity may adopt chatbots for time savings and efficiency, as artificial intelligence (AI) chatbots can converse with users and answer recurring questionsconsumers move away from traditional forms of communication, many experts expect chat-based communication methods to rise. Organizations increasingly use chatbot-based virtual assistants to handle simple tasks, allowing human agents to focus on other responsibilities.

User Interface in Chatbot

A chatbot user interface (UI) is part of a chatbot that users see and interact with. This can include anything from the text on a screen to the buttons and menus that are used to control a chatbot. The chatbot UI is what allows users to send messages and tell it what they want it to do.

Steps taken:

- Step 1: Create a Chatbot Using Python ChatterBot
- Step 2: Begin Training Your Chatbot
- Step 3: Export a WhatsApp Chat
- Step 4: Clean Your Chat Export
- Step 5: Train Your Chatbot on Custom Data and Start Chatting

Program:

Step 1:

```
# bot.py

from chatterbot import ChatBot

chatbot = ChatBot("Chatpot")

exit_conditions = (":q", "quit", "exit")
```

```
while True:
    query = input("> ")
   if query in exit_conditions:
       break
  else:
       print(f"□ {chatbot.get_response(query)}")
```

How do chatbots offer responses?

The bot displays responses in the same order you composed them. You can apply Filters to your bot responses to trigger them only when a condition is met. You can decide how fast your chatbot should respond to a user's question using the Delay feature.

Step 2:

```
# bot.py

from chatterbot import ChatBot
from chatterbot.trainers import ListTrainer

chatbot = ChatBot("Chatpot")

trainer = ListTrainer(chatbot)
trainer.train([
    "Hi",
   "Welcome, friend 🤗",
])
trainer.train([
    "Are you a plant?",
    "No, I'm the pot below the plant!",
])

exit_conditions = (":q", "quit", "exit")
while True:
```

```
query = input("> ")
if query in exit_conditions:
    break
else:
    print(f"□ {chatbot.get_response(query)}")
```

Integrate chatbot on a website:

Inserting the chatbot on your site couldn't be easier. Beneath the chatbot builder, there's a shortcode that you can use to insert the chatbot into a page or post on your WordPress site. You simply copy that code and paste it where you want the chatbot to appear on the page/post.

Step 4:

```
# cleaner.py

import re

def remove_chat_metadata(chat_export_file):
    date_time = r"(\d+\/\d+\/\d+,\s\d+:\d+)" # e.g. "9/16/22, 06:34"
    dash_whitespace = r"\s-\s" # "-"
    username = r"([\w\s]+)"  # e.g. "Martin"
    metadata_end = r":\s" #":"
    pattern = date_time + dash_whitespace + username + metadata_end
```

```
    with open(chat_export_file, "r") as corpus_file:
        content = corpus_file.read()
    cleaned_corpus = re.sub(pattern, "", content)
    return tuple(cleaned_corpus.split("\n"))

if __name__ == "__main__":
    print(remove_chat_metadata("chat.txt"))
```

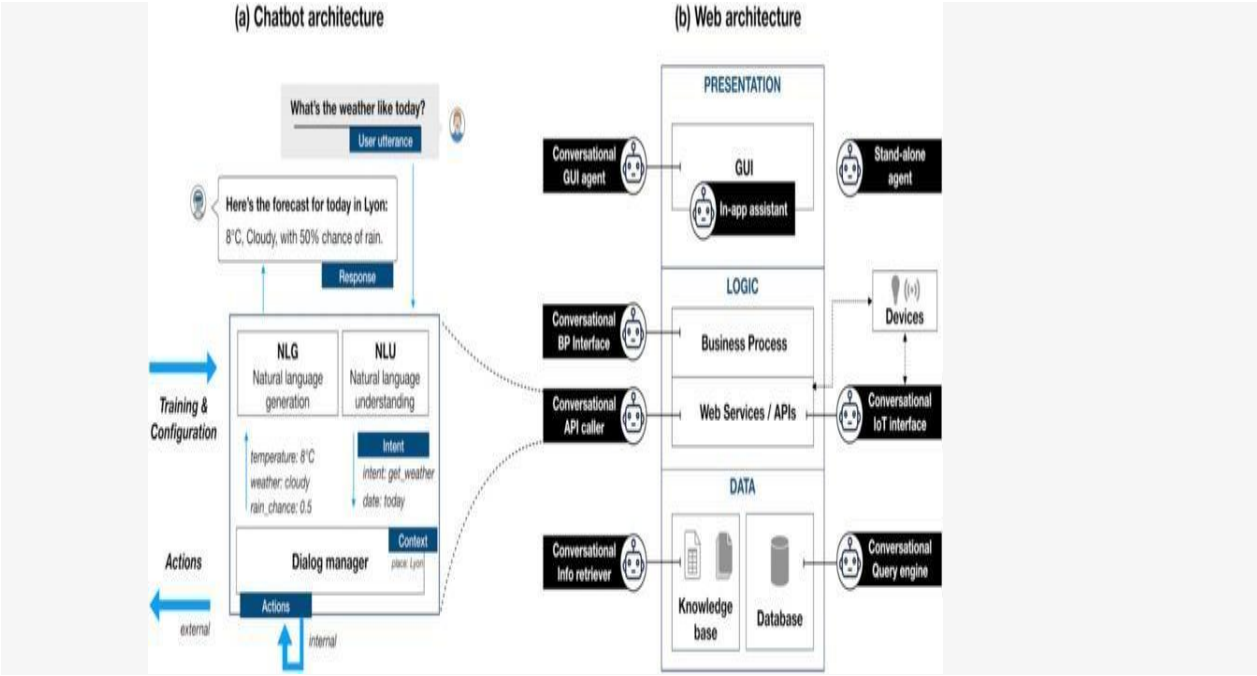
Step 5:

```
# cleaner.py

def remove_non_message_text(export_text_lines):
    messages = export_text_lines[1:-1]

    filter_out_msgs = ("<Media omitted>",)
    return tuple((msg for msg in messages if msg not in filter_out_msgs))

if __name__ == "__main__":
    message_corpus = remove_chat_metadata("chat.txt")
    cleaned_corpus = remove_non_message_text(message_corpus)
    print(cleaned_corpus)
```



Pre-trained language models have achieved striking success in natural language processing (NLP), leading to a paradigm shift from supervised learning to pre- training followed by fine-tuning. The NLP community has witnessed a surge of research of representative work and recent progress in the NLP field and introduces the taxonomy interest in improving pre-trained models. This article presents a comprehensive review of pre-trained models. We then introduce and analyze the impact and challenges of pre-trained models and their downstream applications.

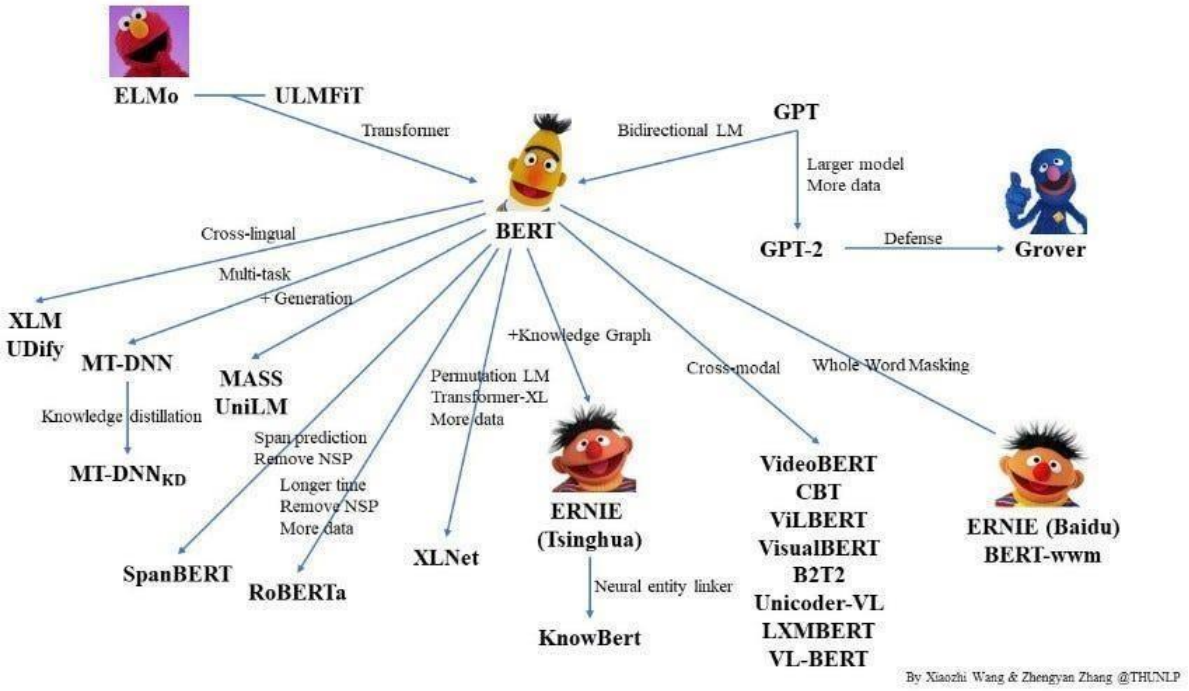
A brief history of pre-trained models

The concept of pre-training is related to transfer learning [1]. The idea of transfer learning is to reuse the knowledge learned from one or more tasks and apply it to new tasks. Traditional transfer learning employs annotated data for supervised training, which has been the common practice for at least a decade. Within deep learning, pre-training with self-supervised learning on massive unannotated data has become the dominant transfer learning approach. The difference is that pre-training methods use unannotated data for self-supervised training and can be applied to various downstream tasks via fine-tuning or few-shot learning.

Although methods that leverage static word embeddings for warm startup can improve the performance of downstream NLP tasks, they lack the ability to represent different meanings of words in context. To solve this problem, context–aware language models were proposed to incorporate the complete context information into the training procedure

Since then, numerous PTMs within the “pre-training then fine- tuning” paradigm have started to emerge. GPT [23], the first model to leverage bidirectional transformers was called Bidirectional Encoder Representations from Transformers (BERT); this model learns bidirectional contexts by means of conditioning on both the left and the right contexts in deep stacked layers. BERT introduced

a denoising autoencoding pre-training task, termed masked language modeling (MLM), to recover the corrupted tokens of input sentences according to their contexts, in what was akin to a cloze task. This approach greatly boosted the performance gain of downstream natural language understanding (NLU) tasks Although the most commonly used pre-training tasks for multimodal context are MLM and masked region prediction, Yu et al.



About GPT-3

GPT-3, or the third-generation Generative Pre-trained Transformer, is a neural network machine learning model trained using internet data to generate any type of text. Developed by OpenAI, it requires a small amount of input text to generate large volumes of relevant and sophisticated machine-generated text.

GPT-3's deep learning neural network is a model with over 175 billion machine learning parameters. To put things into scale, the largest trained language model before GPT-3 was Microsoft's Turing Natural Language Generation (NLG) model, which had 10 billion parameters. As of early 2021, GPT-3 is the largest neural network ever produced. As a result, GPT-3 is better than any prior model for producing text that is convincing enough to seem like a human could have written it.

What can GPT-3 do

GPT-3 processes text input to perform a variety of natural language tasks. It uses both natural language generation and natural language processing to understand and generate natural human language text. Generating content understandable to humans has historically been a challenge for machines that don't know the complexities and nuances of language. GPT-3 has been used to create articles, poetry, stories, news reports and dialogue using a small amount of input text that can be used to produce large amounts of copy.

GPT-3 can create anything with a text structure -- not just human language text. It can also generate text summarizations and even programming code

```
from gpt3 import gpt3


def chat():
    prompt = """Human: Hey, how are you doing?
AI: I'm good! What would you like to chat about?
Human:"""
    while True:
        prompt += input('You: ')
        answer, prompt = gpt3(prompt,
                              temperature=0.9,
                              frequency_penalty=1,
                              presence_penalty=1,
                              start_text='\nAI:',
                              restart_text='\nHuman: ',
                              stop_seq=['\nHuman:', '\n'])
        print('GPT-3:' + answer)
```

```
if_name_== '_main_':
    chat()
```

GPT-3 examples

One of the most notable examples of GPT-3's implementation is the ChatGPT language model. ChatGPT is a variant of the GPT-3 model optimized for human dialogue, meaning it can ask follow-up questions, admit mistakes it has made and challenge incorrect premises. ChatGPT was made free to the public during its research preview to collect user feedback. ChatGPT was designed in part to reduce the possibility of harmful or deceitful responses.

Another common example is Dall-E. Dall-E is an AI image generating neural network built on a 12 billion-parameter version of GPT-3. Dall-E was trained on a data set of text-image pairs and can generate images from user-submitted text prompts. ChatGPT and Dall-E were developed by OpenAI.

➢ Transformer decoders only

- The objective for language modeling is to predict the next token auto-regressively, given its history. The nature of auto-regression entails the future invisibility of input tokens at each position; that is, each token can only attend to the preceding words. GPT [22] was the first model to use the transformer decoder architecture as its backbone.

- n the fine-tuning phase, the pre-trained parameters are set as the initialization of the model for downstream tasks. GPT is pre-trained on the BooksCorpus dataset, which is nearly the same size as the 1B Word Benchmark. It has hundreds of millions of parameters and improves SOTA

/results on nine out of 12 NLP datasets, showing the potential of large-scale PTMs.

➢ Transformer encoders only

Pre-trained transformer encoders, such as BERT [23], have become the standard in NLP systems. BERT uses an MLM framework with a transformer as the backbone. In the pre-training stage, BERT randomly replaces tokens with a special token [MASK] and tries to recover corrupted words based on their contextual representations. It also adopts an objective of next-sentence prediction (NSP) to capture the discourse relations between two sentences, which is helpful for sentence-level tasks, such as question answering.

Another problem with BERT is that it predicts tokens independently without considering other masked tokens.

```
encoder_input = np.array(df.question)
decoder_input = np.array(df.decoder_input)
decoder_label = np.array(df.decoder_label)

n_rows = df.shape[0]
print(f"{n_rows} rows")

indices = np.arange(n_rows)
np.random.shuffle(indices)

encoder_input = encoder_input[indices]
decoder_input = decoder_input[indices]
decoder_label = decoder_label[indices]

train_size = 0.9

train_encoder_input = encoder_input[:int(n_rows*train_size)]
train_decoder_input = decoder_input[:int(n_rows*train_size)]
train_decoder_label = decoder_label[:int(n_rows*train_size)]

test_encoder_input = encoder_input[int(n_rows*train_size):]
test_decoder_input = decoder_input[int(n_rows*train_size):]
test_decoder_label = decoder_label[int(n_rows*train_size):]

print(train_encoder_input.shape)
print(train_decoder_input.shape)
print(train_decoder_label.shape)

print(test_encoder_input.shape)
print(test_decoder_input.shape)
print(test_decoder_label.shape)
```

➢ Transformer encoder–decoders

Transformer encoder–decoder generate a coherent, meaningful, and human-like natural language expression according to specific inputs. For example, the goal of machine translation is to generate a sentence in the target language with the same meaning as the given source language input; for text summarization, the goal is to generate a short version of the input document that captures the core meanings and opinions. The critical point is to model two sequences simultaneously—one for the input and the other for the output.architecture is dedicated to natural language generation (NLG) tasks. Unlike NLU.

Platforms and toolkits for applications

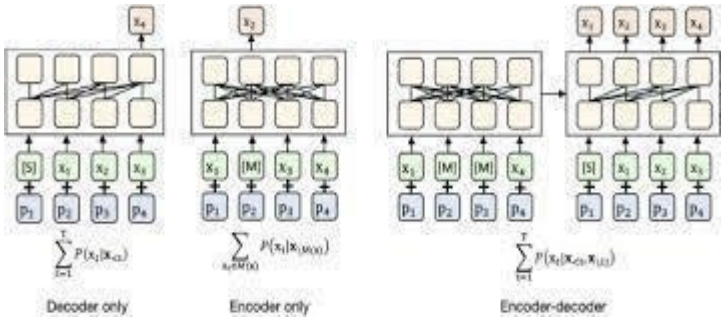
Due to their universality, PTMs have become foundation models in NLP. Many researchers have developed a series of open-source toolkits and platforms to make better use of PTMs. These toolkits and platforms usually contain various PTMs, fine-turning tools, and model-compression tools.

Model	**Number of parameters**	**Model architecture**	**Language**	**Pre-training data**	**Training strategy Trai ning platform**	**Training platform**
DeBERTa1.5B	1.5billion	Encoder only	English	English data (78 GB)	-	PyTorch
T5	11billion	Encoder–decoder (seq2seq)	English	C4 (750 GB)	Model/data parallelism	TensorFlow
GPT-3	175billion	Decoder only	English	Cleaned CommonCrawl, WebText	Model parallelism	-
CPM	2.6billion	Decoder only	Chinese	Chinese corpus (100 GB)	-	PyTorch
PanGu-α	200billion	Decoder only	Chinese	Chinese data (1.1 TB, 250 billion tokens)	MindSpore auto-parallel	MindSpore

Future of GPT-3

There are many open source efforts in play to provide a free and non-licensed model as a counterweight to Microsoft's exclusive ownership. New language models are published frequently on Hugging Face's platform.

It is unclear exactly how GPT-3 will develop in the future, but it is likely that it will continue to find real-world uses and be embedded in various generative AI applications. Many applications already use GPT-3, including Apple's Siri virtual assistant. Where possible, GPT-4 will be integrated where GPT-3 was used.



PTMs can fully exploit unannotated data for self-supervised learning and have become the foundation models in NLP, significantly improving the performance of downstream NLP tasks. The emergence of PTMs opens up a new "pre-training then fine-tuning" paradigm for NLP.

Another promising direction is to incorporate prior knowledge into PTMs to improve their reasoning abilities and efficiency. Existing work on knowledge pre-training, such as K-BERT [33] and ERNIE 3.0 [39], has injected knowledge triplets into pre-training or fine-tuning. However, PTMs have demonstrated limited capability for commonsense awareness and reasoning, there is still a long way to go for PTMs to be able to make reliable decisions and carry out reliable planning, which are essential elements of AI. More efficient and powerful neural networks need to be proposed and developed. Fortunately, the use of PTMs in real applications continues to provide an increased amount of data and address new challenges, potentially promoting the rapid development of new pre-trained methods.

Some common steps in data preprocessing include:

Data preprocessing is an important step in the data mining process that involves cleaning and transforming raw data to make it suitable for analysis Some common steps in data preprocessing include

Data Cleaning:

This involves identifying and correcting errors or inconsistencies in the data, such as missing values, outliers, and duplicates. Various techniques can be used for data cleaning, such as imputation, removal, and transformation.

Data Integration:

This involves combining data from multiple sources to create a unified dataset. Data integration can be challenging as it requires handling data with different formats, structures, and semantics. Techniques such as record linkage and data fusion can be used for data integration.

Data Transformation:

This involves converting the data into a suitable format for analysis. Common techniques used in data transformation include normalization, standardization, and discretization. Normalization is used to scale the data to a common range, while standardization is used to transform the data to have zero mean and unit variance. Discretization is used to convert continuous data into discrete categories.

Data Reduction:

This involves reducing the size of the dataset while preserving the important information. Data reduction can be achieved through techniques such as feature selection and feature extraction. Feature selection involves selecting a subset of relevant features from the dataset, while feature extraction involves transforming the data into a lower-dimensional space while preserving the important information

Data Discretization:

This involves dividing continuous data into discrete categories or intervals. Discretization is often used in data mining and machine learning algorithms that require categorical data. Discretization can be achieved through techniques such as equal width binning, equal frequency binning, and clustering

Data Normalization:

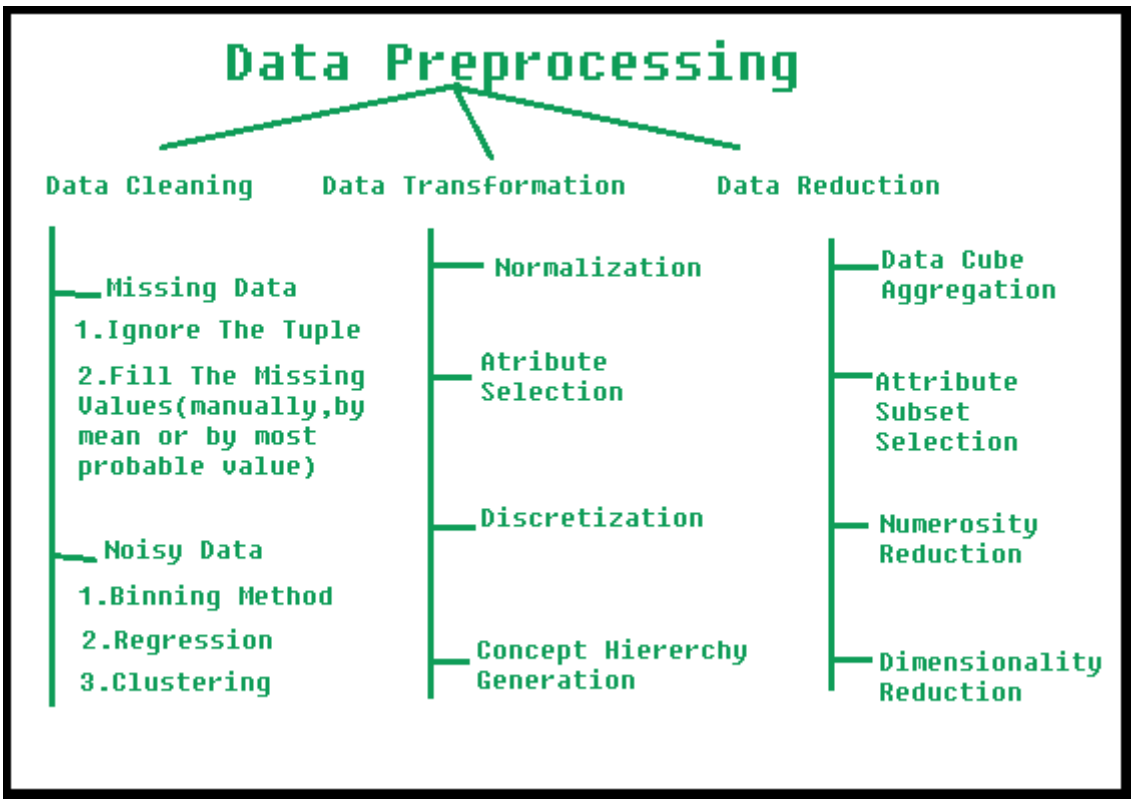
This involves scaling the data to a common range, such as between 0 and 1 or -1 and 1. Normalization is often used to handle data with different units and scales. Common normalization techniques include min-max normalization, z-score normalization, and decimal scaling.

Data preprocessing plays a crucial role in ensuring the quality of data and the accuracy of the analysis results. The specific steps involved in data preprocessing may vary depending on the nature of the data and the analysis goals.

By performing these steps, the data mining process becomes more efficient and the results become more accurata

Preprocessing in Data Mining:

Data preprocessing is a data mining technique which is used to transform the raw data in a useful and efficient format.



Steps Involved in Data Preprocessing:

Data Cleaning:

The data can have many irrelevant and missing parts. To handle this part, data cleaning is done. It involves handling of missing data, noisy data etc.

Missing Data:

This situation arises when some data is missing in the data. It can be handled in various ways.
Some of them are:

Ignore the tuples:

This approach is suitable only when the dataset we have is quite large and multiple values are missing within a tuple**.**

Fill the Missing values:

There are various ways to do this task. You can choose to fill the missing values manually, by attribute mean or the most probable value.

Noisy Data:

Noisy data is a meaningless data that can't be interpreted by machines.It can be generated due to faulty data collection, data entry errors etc. It can be handled in following ways

1. Binning Method:

This method works on sorted data in order to smooth it. The whole data is divided into segments of equal size and then various methods are performed to complete the task. Each segmented is handled separately. One can replace all data in a segment by its mean or boundary values can be used to complete the task.

2. Regression:

Here data can be made smooth by fitting it to a regression function.The regression used may be linear (having one independent variable) or multiple (having multiple independent variables**)**

3. Clustering:

This approach groups the similar data in a cluster. The outliers may be undetected or it will fall outside the clusters**.**

Data Transformation:

This step is taken in order to transform the data in appropriate forms suitable for mining process. This involves following ways

Normalization:

It is done in order to scale the data values in a specified range (1.0 to 1.0 or 0.0 to 1.0)

Attribute Selection:

In this strategy, new attributes are constructed from the given set of attributes to help the mining process.

Discretization:

This is done to replace the raw values of numeric attribute by interval levels or conceptual levels.

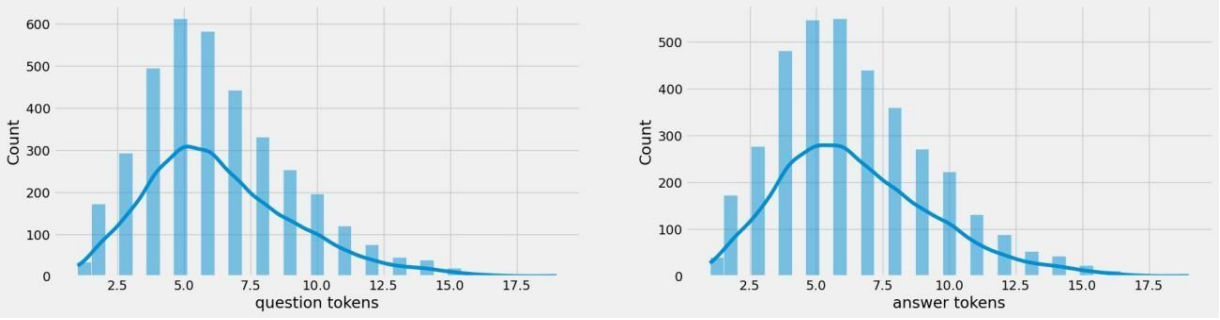
Concept Hierarchy Generation:

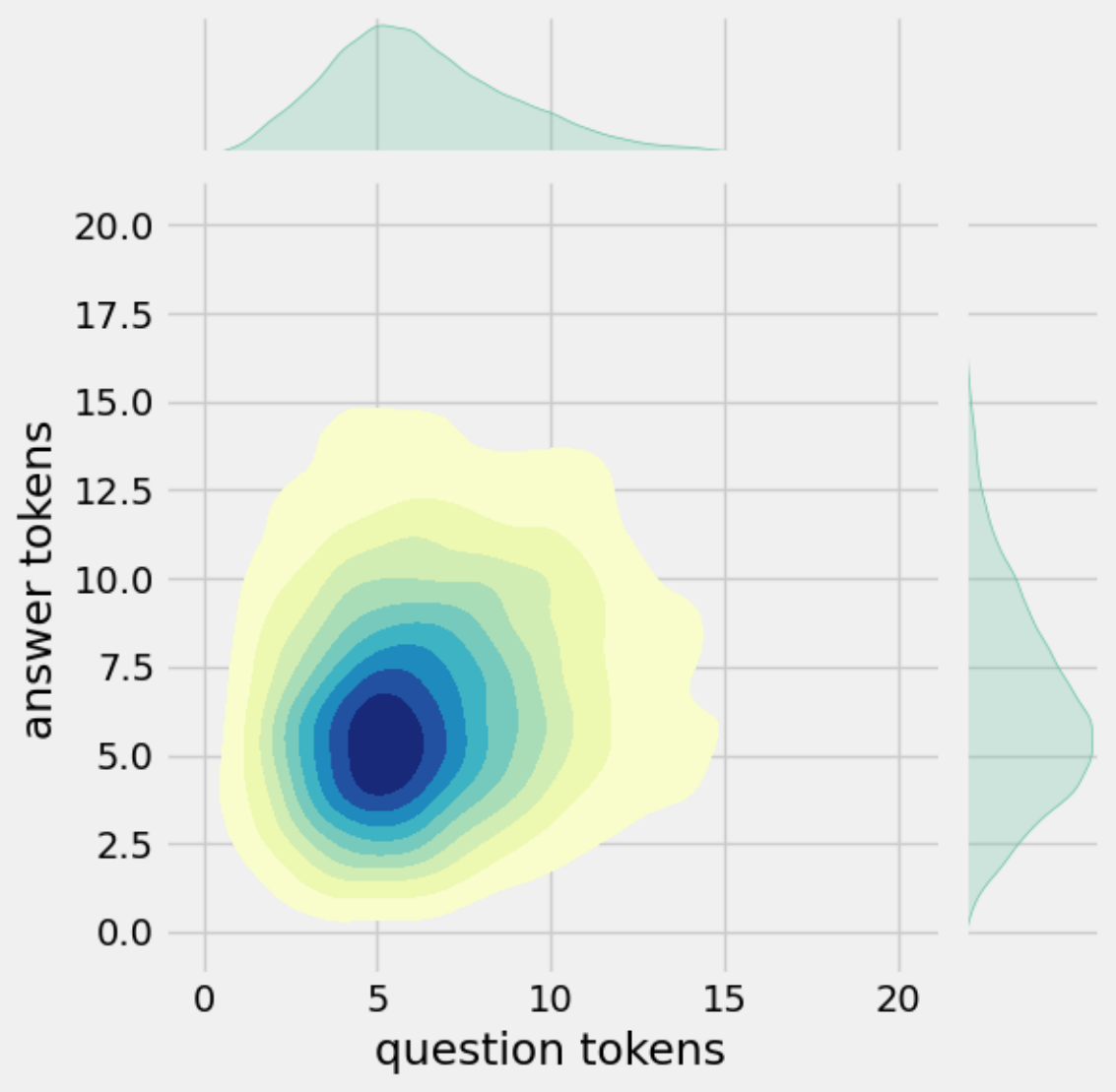
Here attributes are converted from lower level to higher level in hierarchy. For Example-The attribute “city” can be converted to “country”.

Data Preprocessing:

Data Visualization

```
df['question tokens']=df['question'].apply(lambda x:len(x.split()))
df['answer tokens']=df['answer'].apply(lambda x:len(x.split()))
plt.style.use('fivethirtyeight')
fig,ax=plt.subplots(nrows=1,ncols=2,figsize=(20,5))
sns.set_palette('Set2')
sns.histplot(x=df['question tokens'],data=df,kde=True,ax=ax[0])
sns.histplot(x=df['answer tokens'],data=df,kde=True,ax=ax[1])
sns.jointplot(x='question tokens',y='answer tokens',data=df,kind='kde',fill=True,cmap='YlGnBu')
plt.show()
```





Text Cleaning

```
def clean_text(text):
    text=re.sub('-',' ',text.lower())
    text=re.sub('[.]',' . ',text)
    text=re.sub('[1]',' 1 ',text)
    text=re.sub('[2]',' 2 ',text)
    text=re.sub('[3]',' 3 ',text)
```

```
    text=re.sub('[4]',' 4 ',text)
    text=re.sub('[5]',' 5 ',text)
    text=re.sub('[6]',' 6 ',text)
    text=re.sub('[7]',' 7 ',text)
    text=re.sub('[8]',' 8 ',text)
    text=re.sub('[9]',' 9 ',text)
    text=re.sub('[0]',' 0 ',text)
    text=re.sub('[,]',' , ',text)
    text=re.sub('[?]',' ? ',text)
    text=re.sub('[!]',' ! ',text)
    text=re.sub('[$]',' $ ',text)
    text=re.sub('[&]',' & ',text)
    text=re.sub('[/]',' / ',text)
    text=re.sub('[:]',' : ',text)
    text=re.sub('[;]',' ; ',text)
    text=re.sub('[*]',' * ',text)
    text=re.sub('[\']',' \' ',text)
    text=re.sub('[\"]',' \" ',text)
    text=re.sub('\t',' ',text)
    return text


df.drop(columns=['answer tokens','question
tokens'],axis=1,inplace=True)
```

```
df['encoder_inputs']=df['question'].apply(clean_text)
df['decoder_targets']=df['answer'].apply(clean_text)+' <end>'
df['decoder_inputs']='<start> '+df['answer'].apply(clean_text)+' <end>'
df.head(10)
```

After preprocessing: for example , if your birth date is january 1 2 , 1 9 8 7 , write 0 1 / 1 2 / 8 7 .

Max encoder input length: 27

Max decoder input length: 29

Max decoder target length: 28

Tokenization

```
vectorize_layer=TextVectorization(
    max_tokens=vocab_size,
    standardize=None,
    output_mode='int',
    output_sequence_length=max_sequence_length
)
vectorize_layer.adapt(df['encoder_inputs']+' '+df['decoder_targets']+' <start> <end>')
vocab_size=len(vectorize_layer.get_vocabulary())
print(f'Vocab size: {len(vectorize_layer.get_vocabulary())}')
print(f'{vectorize_layer.get_vocabulary()[:12]}')
```

```
Vocab size: 2443
['', '[UNK]', '<end>', '.', '<start>', '"', 'i', '?', 'you', ',', 'the', 'to']
def sequences2ids(sequence):
    return vectorize_layer(sequence)

def ids2sequences(ids):
    decode=''
    if type(ids)==int:
        ids=[ids]
    for id in ids:
        decode+=vectorize_layer.get_vocabulary()[id]+' '
    return decode

x=sequences2ids(df['encoder_inputs'])
yd=sequences2ids(df['decoder_inputs'])
y=sequences2ids(df['decoder_targets'])

print(f'Question sentence: hi , how are you ?')
print(f'Question to tokens: {sequences2ids("hi , how are you ?")[:10]}')
print(f'Encoder input shape: {x.shape}')
print(f'Decoder input shape: {yd.shape}')
print(f'Decoder target shape: {y.shape}')
Question sentence: hi , how are you ?
```

Question to tokens: [1971 9 45 24 8 7 0 0 0 0]

Encoder input shape: (3725, 30)

Decoder input shape: (3725, 30)

Decoder target shape: (3725, 30)

print(f'Encoder input: {x[0][:12]} ...')

print(f'Decoder input: {yd[0][:12]} ...') # shifted by one time step of the target as input to decoder is the output of the previous timestep

print(f'Decoder target: {y[0][:12]} ...')

Encoder input: [1971 9 45 24 8 194 7 0 0 0 0 0] ...

Decoder input: [4 6 5 38 646 3 45 41 563 7 2 0] ...

Decoder target: [6 5 38 646 3 45 41 563 7 2 0 0] ...

data=tf.data.Dataset.from_tensor_slices((x,yd,y))

data=data.shuffle(buffer_size)

train_data=data.take(int(.9*len(data)))

train_data=train_data.cache()

train_data=train_data.shuffle(buffer_size)

train_data=train_data.batch(batch_size)

train_data=train_data.prefetch(tf.data.AUTOTUNE)

train_data_iterator=train_data.as_numpy_iterator()

val_data=data.skip(int(.9*len(data))).take(int(.1*len(data)))

val_data=val_data.batch(batch_size)

```
val_data=val_data.prefetch(tf.data.AUTOTUNE)


_=train_data_iterator.next()
print(f'Number of train batches: {len(train_data)}')
print(f'Number of training data: {len(train_data)*batch_size}')
print(f'Number of validation batches: {len(val_data)}')
print(f'Number of validation data: {len(val_data)*batch_size}')
print(f'Encoder Input shape (with batches): {_[0].shape}')
print(f'Decoder Input shape (with batches): {_[1].shape}')
print(f'Target Output shape (with batches): {_[2].shape}')
Number of train batches: 23
Number of training data: 3427
Number of validation batches: 3
Number of validation data: 447
Encoder Input shape (with batches): (149, 30)
Decoder Input shape (with batches): (149, 30)
Target Output shape (with batches): (149, 30)
```

Data Reduction:

Data reduction is a crucial step in the data mining process that involves reducing the size of the dataset while preserving the important information. This is done to improve the efficiency of data analysis and to avoid overfitting of the model. Some common steps involved in data reduction are:

Feature Selection

This involves selecting a subset of relevant features from the dataset. Feature selection is often performed to remove irrelevant or redundant features from the dataset. It can be done using various techniques such as correlation analysis, mutual information, and principal component analysis (PCA).

Sampling

This involves selecting a subset of data points from the dataset. Sampling is often used to reduce the size of the dataset while preserving the important information. It can be done using techniques such as random sampling, stratified sampling, and systematic sampling.

Clustering

This involves grouping similar data points together into clusters. Clustering is often used to reduce the size of the dataset by replacing similar data points with a representative centroid. It can be done using techniques such as k-means, hierarchical clustering, and density-based clustering.

What is flask chatbot

Chatbots are software tools created to interact with humans through chat. The first chatbots could create simple conversations based on a complex system of rules. You can build intelligent chatbots for WhatsApp using the Python Framework Flask and the Kompose Bot builder.

The Flask is a Python micro-framework used to create small web applications and websites using Python. Flask works on a popular templating engine called Jinja2, a web templating system combined with data sources to the dynamic web pages.

How to deploy a chatbot on Flask

First, we will install the Flask library in our system using the below command:

```
pip install flask
```

And for Google Colab use the below command, mostly Flask comes pre-install on Google Colab.

```
!pip install flask
```

But first, let’s understand what the Flask framework in Python is.

How to create chatbot using Flask Python

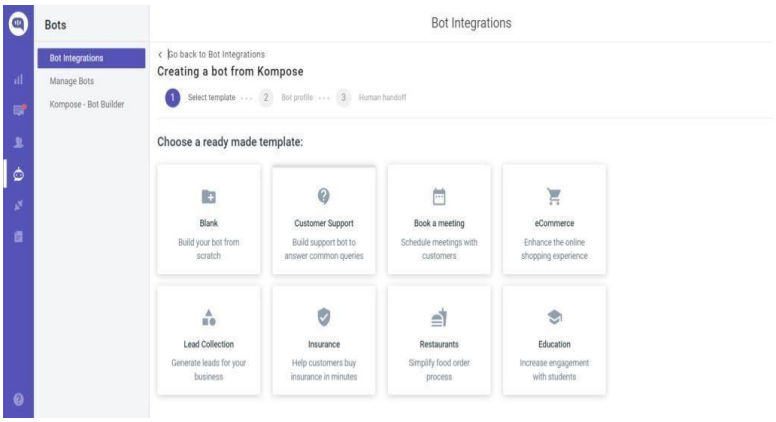
- Step 1: Import necessary methods of Flask and ChatterBot.
- Step 2: Then, we will initialize the Flask app by adding the below code. Flask(name) is used to create the Flask class object so that Python code can initialize the Flask server.
- Step 3: Now, we will give the name to our chatbot.

Developing a chatbot using flask

We will make a Flask chatbot. Flask is a microframework used for web development. We will follow the process given below:

1. Make a web app using the flask.
2. Make a directory for the templates.
3. Train the bot.
4. Make conversation with the bot.

Flask is a Python web framework that is widely used for building web applications. It is a lightweight framework that is easy to learn and has a simple syntax. In this tutorial, you will learn how to use Flask to create a back-end that communicates with a third-party API and renders the response in dynamic HTML using Jinja.



- **Step 1**: Installing Python 3.7 with pip3.7 along with Python3.10

This step is optional if you already have Python Version *<=3.8 or >=3.4*. Please check your python using this command – python -V or python3 -V

```
# Install Python 3.7 Version

sudo apt update

sudo apt install build-essential zlib1g-dev libncurses5-dev libgdbm-dev libnss3-dev libssl-dev libsqlite3-dev libreadline-dev libffi-dev wget libbz2-dev

sudo add-apt-repository ppa:deadsnakes/ppa

sudo apt install python3.7

python3.7 --version

# Install Pip

python3.7 -m pip install pip

# if you are getting distutils error
```

```
sudo apt-get install python3.7-distutilsCopy Code
```

- **Step 2:** Create and Install Virtualenv for Python Version <=3.8 or >=3.4

As we will be using virtual environment for this project. Please read our blog on Using Virtual Environments for Python Projects, if you are not familar with virtualenv in Python.

Let's install virtualenv using Python3.7

```
pip install virtualenvCopy Code
```

Create virtualenv named venv and activate it.

```
python3.7 -m virtualenv venv

source venv/bin/activateCopy Code
```

Please Note: When you have virtualenv activated, you will see python version 3.7. Outside virtualenv it will be different.

- **Step 3**: Installing Required Libraries

```
pip install Flask

pip install ChatterBot==1.0.8

pip install chatterbot-corpus==1.2.0

pip install spacy==2.1.9
```

```
pip install nltk==3.8.1Copy Code
```

- **Step 4**: Initializing Flask App

Create a new Python file (app.py) and add the following code:

```
from chatbot import chatbot

from flask import Flask, render_template, request

app = Flask(__name__)

app.static_folder = 'static'



@app.route("/")

def home():

    return render_template("index.html")

@app.route("/get")

def get_bot_response():

    userText = request.args.get('msg')

    return str(chatbot.get_response(userText))

if __name__ == "__main__":
```

```
app.run()Copy Code
```

- **Step 5**: Interact with Chatterbot Machine Learning Model

Create a new Python file (chatbot.py) and and add the following code:

```
from chatterbot import ChatBot

import spacy

spacy.cli.download("en_core_web_sm")

spacy.cli.download("en")



nlp = spacy.load('en_core_web_sm')

chatbot = ChatBot(

    'CoronaBot',

    storage_adapter='chatterbot.storage.SQLStorageAdapter',

    logic_adapters=[

        'chatterbot.logic.MathematicalEvaluation',

        'chatterbot.logic.TimeLogicAdapter',

        'chatterbot.logic.BestMatch',
```

```
        {
            'import_path': 'chatterbot.logic.BestMatch',
            'default_response': 'I am sorry, but I do not understand. I am still learning.',
            'maximum_similarity_threshold': 0.90
        }
    ],
    database_uri='sqlite:///database.sqlite3'
)
# Training With Own Questions
from chatterbot.trainers import ListTrainer
trainer = ListTrainer(chatbot)
training_data_quesans = open('training_data/ques_ans.txt').read().splitlines()
training_data_personal = open('training_data/personal_ques.txt').read().splitlines()
training_data = training_data_quesans + training_data_personal
trainer.train(training_data)
# Training With Corpus
from chatterbot.trainers import ChatterBotCorpusTrainer
```

```
trainer_corpus = ChatterBotCorpusTrainer(chatbot)

trainer_corpus.train(

    'chatterbot.corpus.english'

)Copy Code
```

In above example, we are loading data from training_data/ques_ans.txt and training_data/personal_ques.txt. You can download the files from here.

- **Step 6:** Creating Chatbot UI

    - HTML Template (index.html) – Create a new HTML file index.html inside the “templates” folder. Please copy the html content from here.
    - CSS File (style.css) – Create a new CSS file inside the “static” folder to style the chat interface. Please copy the css content from here.

Bellow is the folder structure your project should look like:

```
chatbot_project/

├── app.py

├── chatbot.py

├── venv

├── database.sqlite3.py

├── templates/
```

```
│   └── index.html
├── training_data/
│   ├── ques_ans.txt
│   └──personal_ques.txt
└── static/
    └── style.cssCopy Code
```

- **Step 7**: Run the Flask App

Open your terminal, navigate to the chatbot_project directory, and execute the following command:

```
python3.7 app.pyCopy Code
```

Step 8: Interact with the Chatbot

Open your web browser and go to http://localhost:5000 to access the chat interface. Now, you can type messages in the input field, press "Send," and watch your chatbot respond!

Note – If you are not getting right answer for questions, delete the .sqlite3 database file from your folder and re-run the flask app.

Conclusion

A conclusion in a project report is a summary of the main findings and outcomes of the project. It should include a summary of the research question, the methods used, the results obtained, and the main conclusions drawn from the research.

Future work

PTMs can fully exploit unannotated data for self-supervised learning and have become the foundation models in NLP, significantly improving the performance of downstream NLP tasks. The emergence of PTMs opens up a new “pre-training then fine- tuning” paradigm for NLP.

Another promising direction is to incorporate prior knowledge into PTMs to improve their reasoning abilities and efficiency. Existing work on knowledge pre-training, such has injected knowledge triplets into pre-training or fine-tuning. However, PTMs have demonstrated limited capability for commonsense awareness and reasoning, there is still a long way to go for PTMs to be able to make reliable decisions and carry out reliable planning, which are essential elements of AI. More efficient and powerful neural networks need to be proposed and developed.

Fortunately, the use of PTMs in real applications continues to provide an increased amount of data and address new challenges, potentially promoting the rapid development of new pre-trained methods.