PRIMER CONTACTO CON UNA MENTE ARTIFICIAL

Autor: Juan José Pérez Cervera

ÍNDICE

CAPÍTULO PRIMERO

INTRODUCCIÓN

La presente obra consiste en un primer contacto o aproximación al diseño de una mente artificial.

Para el desarrollo de este primer prototipo, muy elemental, se ha utilizado programación funcional con Python 3.0, y diferentes herramientas de Machine Learning (ML) y Redes Neuronales (RN).

Se ha trabajado con pipelines obtenidos de la página web huggingface.co.

El entorno de desarrollo ha sido el de Anaconda, utilizando Tensorflow y Keras.

Se han utilizado diferentes librerías de Python, en especial Nltk y Networkx, así como diferentes recursos de procesamiento de lenguaje natural (NLP).

El sistema operativo con el que se ha trabajado ha sido Windows 10.

Debo señalar que este primer prototipo está incompleto, sirviendo solo como base para futuros trabajos. En ningún momento se ha diseñado con el fin de superar un test de Turing, ni nada parecido. La Artificial Mind (A.M) se reconoce, así misma, desde el primer momento como algo artificial.

La construcción de la A.M se inspira en la mente humana, pero se busca algo distinto a esta.

El prototipo pretende ser algo más que un simple chatbot conversacional, estando dotado de una estructura cognitiva básica.

En cuanto a la terminología empleada en la presente obra, quiero destacar que a la información que el usuario da a través de la consola la he denominado “entrada”, y a la información que procede de la máquina “salida”.

El prototipo será referido a lo largo de la obra como A.M (Artificial Mind). En ocasiones utilizo el término genérico “grafo”, aunque en realidad se trate de un digrafo u otro tipo de grafo.

Debo aclarar que este trabajo no es de carácter divulgativo, por lo que no se efectúan explicaciones teóricas sobre las herramientas utilizadas. La obra va dirigida a personas con interés en la materia tratada, y que ya poseen conocimientos sobre programación e Inteligencia Artificial.

Debo decir que el proyecto es realmente muy escalable, y no se han implementado todas las referencias teóricas contenidas en la obra.

En cuanto a la estructura de la obra, esta se divide en siete capítulos.

El capítulo primero es la presente introducción.

El capítulo segundo trata sobre el armazón o estructura cognitiva de la A.M.

El capítulo tercero se centra en el módulo llamado interacción. En este módulo queda sin resolver el problema de las “intenciones”, si bien se apunta alguna posible solución, siendo esta cuestión clave para el desarrollo futuro de la A.M.

El capítulo cuarto versa sobre las relaciones causales. Se describen de forma breve los módulos referentes a dichas relaciones.

El capítulo quinto trata sobre el pensamiento generativo. Este capítulo tiene una alta dosis de especulación, pero merece la pena explorarlo. Se busca sobre todo entender cómo se construye un pensamiento de forma autónoma por la A.M.

El capítulo sexto aborda el problema de la autoconsciencia artificial, siendo de carácter totalmente teórico, tratando temas de especial interés como la creación del “Yo” en la máquina o el problema del libre albedrío.

El capítulo séptimo se centra en efectuar unas reflexiones finales.

El código completo de la A.M se encuentra en el siguiente enlace:

perez2103/PROYECTO (github.com)

El idioma español es el empleado por la A.M. Las “entradas” deben realizarse en español y la A.M responderá con “salidas” también en español.

CAPÍTULO SEGUNDO

ESTRUCTURA COGNITIVA

En esta primera aproximación nos limitaremos a dar unas rudimentarias líneas que configuren una primitiva estructura cognitiva de A.M.

En primer lugar, debemos tener en cuenta los archivos de almacenamiento de información. Se deben utilizar grafos que contengan los conocimientos que irá adquiriendo la A.M.

Se enumeran seguidamente algunos de los grafos más importantes:

1.- Grafo perfil de A.M

En este grafo se encuentran predefinidos una serie de datos correspondientes a la propia A.M, como son edad, estatura, peso, gustos etc… Este grafo no se ha implementado en su totalidad.

El grafo irá enriqueciéndose conforme se produzcan interacciones con el usuario. El funcionamiento del grafo se desarrollará en futuras actualizaciones.

2.- Grafo perfil de usuario.

En este grafo se recogen los datos que tienen relación con el usuario principal. Se encuentra también pendiente de desarrollo.

3.- Grafos de la realidad de A.M.

Estos grafos incluyen los llamados grafos categorías. Se trata de 13 grafos referidos a una serie de categorías como por ejemplo, cine, animales, educación, salud, etc…

En estos grafos se irán creando diferentes nodos relacionados con la correspondiente categoría. Los grafos se van construyendo en función de la interacción con el usuario

4.- Grafo del Mundo de Conocimientos (MC).

Se trata de un grafo con datos predefinidos sobre el mundo. En este grafo se almacenan debidamente clasificados diferentes conocimientos referidos a Ciencias y Humanidades. El grafo se encuentra pendiente de implementación.

5.- Grafo mundos posibles

Se trata de un grafo que recoge relaciones causales, algunas predefinidas y otras se van creando con la interacción con el usuario. Se ha implementado solo un ejemplo de lo que podría ser este grafo.

6.- Grafo normativo.

Recoge un conjunto de normas de diferente naturaleza. Se encuentra pendiente de implementación.

Para la construcción de los grafos se han usado archivos del tipo .graphml.

Dentro del almacenamiento de información tenemos que destacar los archivos de carácter temporal. Se han utilizado varios archivos csv, que sirven para retener información de forma limitada en el tiempo.

Otros archivos que contendrán información son los archivos de texto plano (.txt), en los que se almacenaran diferentes datos.

Se distinguen los siguientes:

1.- Archivo Corpus_usuario.txt

Este archivo recoge las diferentes entradas que realiza el usuario, siendo de utilidad para poder generar la personalidad del mismo. Esto último se conseguirá a través de una red neuronal generativa de texto. Este tipo de archivo, así como la red neuronal, están pendientes de incorporación al prototipo.

2.- Archivo Corpus_AM.txt

Este archivo recoge tanto las entradas del usuario como las salidas de la A.M, también recoge datos que se obtienen de forma autónoma por la A.M.

El archivo se utiliza para crear la propia personalidad de la A.M, a través de una red generadora de texto.

Hasta aquí hemos visto archivos de almacenamiento de información. Además de estos archivos se deben tener en cuenta los módulos de búsqueda e interacción.

En los citados módulos se procesará la información, y se generaran las correspondientes “salidas”.

Antes de ofrecer una “salida”, la A.M habrá procesado la información dada por el usuario, de tal forma que la “salida” será el resultado de la ejecución de una serie de algoritmos, entre ellos los correspondientes a la toma de decisiones.

Por último, debemos hacer referencia a una serie de redes neuronales que se activaran en determinadas condiciones.

Una vez concluida la interacción con el usuario, la A.M continuará funcionando, realizando toda una serie de tareas de procesamiento. En esta fase entrarán en juego diferentes redes neuronales, muchas de ellas dirigidas a la creación de pensamiento generativo.

El prototipo está pensado para la interacción con un único usuario. Sin embargo, para una mayor accesibilidad por el momento no se ha programado un login de usuario. En futuros desarrollos se incorporará dicho login.

Es importante abundar en el hecho de que el prototipo esta ideado para interactuar siempre con el mismo usuario, y serán las “entradas” de este las que generen fundamentalmente la personalidad de la A.M

La A.M se nutre con las “entradas” que le va proporcionando el usuario, dichas “entradas” son el combustible de la A.M, a través de ellas se crean nuevos grafos y se almacena información.

El prototipo se inicia con una serie de módulos que tienen por objeto mostrar las diferentes herramientas utilizadas en la confección de la A.M. Estos módulos se suprimirán en ulteriores desarrollos.

El módulo inicio es el “main” del programa. Después de una breve presentación comienzan a importarse los diversos módulos contenidos en la carpeta llamada “paquete”.

El módulo de inicio termina con la activación de los procesos de pensamiento generativo. Se puede revisar todo el código del módulo inicio en el ANEXO I.

Dentro de los módulos de presentación destacamos los siguientes:

1.- Módulo hoy

2.- Módulo hoy_am

3.- Módulo completar_frase

4.- Módulo definiciones

5.- Módulo contexto42

Pasamos a explicar brevemente cada uno de los módulos:

1.- Módulo hoy.

Este módulo consiste en preguntar al usuario sobre lo que ha hecho en el día. Se recoge una lista con posibles frases de respuesta. No se ha utilizado un csv, únicamente una lista de frases. El objetivo de este módulo es mostrar la clase SequenceMatcher del módulo "difflib" de Python. Esta herramienta se usa para comparar frases de secuencias de entrada. A continuación, la A.M efectúa una serie de preguntas sobre refranes, y se utiliza igualmente el SequenceMatcher. En lo sucesivo me referiré a esta herramienta como "matcher" o "match".

2.- Módulo hoy_am.

Este módulo es un complemento del anterior, aquí simplemente se utiliza un archivo csv, que sirve para dar respuesta a la pregunta que efectúa la A.M al usuario. La idea es que A.M de una respuesta a modo de "espejo", reproduciendo una de las posibles respuestas que en su momento efectuó el usuario.

3.- Módulo completar_frase.

Se utiliza en este módulo el pipeline ("fill-mask"), obtenido del huggingface.co. Este pipeline completa frases con cierta coherencia. Ciertamente, produce una ilusión de que la máquina esta "entendiendo" el sentido de la frase.

En lo sucesivo me referiré a este pipeline como "máscara" o "mask".

4.- Módulo definiciones.

Este módulo usa el wordnet de nltk.corpus. A través del wordnet.synsets se obtienen posibles definiciones de la palabra introducida por el usuario.

5.- Módulo contexto42.

En este módulo se usa el pipeline “text2text-generation”. Este pipeline se usa en tareas de PNL como respuesta a preguntas.

Se ha utilizado como corpus un archivo.json, nombrado como ‘data.json´. Este archivo contiene diferentes frases referidas a A.M. El usuario podrá realizar una serie de preguntas sobre la A.M, la cual intentará contestar atendiendo a la información contenida en el archivo. json.

En lo sucesivo en esta obra me referiré al text2text-generation como “t2t”.

Todo el código se encuentra recogido en el ANEXO II.

En la carpeta funciones se incluyen los siguientes archivos:

1.- Almacén.

Recoge las funciones guardar_archivo_csv , y cargar_archivo_csv.

2.- Herramientas.

Se encuentran recogidas diferentes herramientas como por ejemplo la función traducción.

3.- Normalizar.

Se encuentra la función tildes.

4.- Procesar.

Se recogen las funciones match, mascara, descripcion, t2t, y tag_sustantivos.

5.- Varios.

Contiene funciones como menu_cargar_categorias, y menu_guardar_categorias.

Seguidamente paso a realizar una breve explicación del módulo búsqueda.

El módulo búsqueda tiene una serie de imports, necesarios para poder ejecutarse. Hay silenciadas instalaciones y descargas, las cuales deben activarse para la correcta ejecución del módulo.

Además, de la importación de las librerías, se importan funciones de la carpeta FUNCIONES.

El programa se inicia con la entrada del usuario, la cual es almacenada en un archivo temporal csv. La entrada es traducida y se pasa al clasificador. El resultado nos dará una determinada categoría a la que llamamos “categoria1”.

Se importa la función menu_cargar_categorias, que contiene el menú con las 13 categorías. Seguidamente, se obtienen los tags de la “entrada” con blog.tag de la librería textblob de Python. En este caso nos interesan solo los sustantivos.

En futuras actualizaciones usaremos Spacy para obtener los tags, ya que permite una mejor relación de estos con el significado de la frase de entrada.

En este momento, nos limitamos a seleccionar sustantivos de forma arbitraria, por lo que muchas veces no tienen una relación próxima con la categoría.

En principio podríamos obtener hasta cuatro variables a las que llamamos nodo1, nodo2, nodo3 y nodo4. El nodo1 se corresponde con el nombre de la categoría, y los otros nodos son sustantivos obtenidos de la frase de entrada.

Lo siguiente es ver si hay coincidencia del nodo2 con los nodos contenidos en la lista de nodos del grafo. En caso de no existir el nodo2, este se creará con su correspondiente borde. Se recogen diferentes try exceptions para evitar fallos del programa.

A continuación, obtenemos la lista de valores del diccionario, es decir, las frases almacenadas en el diccionario. Recuperamos la entrada del usuario y le pasamos un "match" para ver si hay coincidencia entre la entrada y alguna de las frases almacenadas. En caso de no obtener coincidencias la "salida" será una frase aleatoria de entre las guardadas.

Lo siguiente es la creación de un Digrafo dirigido al que se le añaden los nodos seleccionados y sus respectivos pesos (inicialmente se encuentran a cero). Después se añade una frase a la relación entre nodos.

Existe la posibilidad de visionar el grafo, para ello solo se debe desmarcar plt.show().

El paso siguiente es lo que he denominado "CRISOL" O "SOPA" de significados. Se trata de recoger en una lista ciertos contenidos que irán formando el "corpus" sobre el que pasar un "t2t".

En el crisol o sopa recogemos la entrada del usuario, así como, definiciones de los diferentes nodos. La sopa se convertirá en el "corpus" de conocimiento. Se procede a importar la función "t2t", la cual permite efectuar preguntas para ver si la A.M ha "entendido" la frase de entrada.

A continuación, se efectuará una composición entre el digrafo recién formado y el grafo perteneciente a la categoría. Igualmente se puede visionar el grafo compuesto desmarcando plt.show().

Una vez que hemos obtenido el grafo compuesto, utilizaremos el algoritmo dijkstra para el cálculo de la ruta entre nodos. La ruta entre nodos que tenga mayor valor será la elegida. Una vez se consigue una ruta ganadora esta queda reforzada aumentando su peso en una unidad. El objetivo de esto es que vayan aumentando los pesos de las rutas ganadoras, de tal forma que unas rutas se reforzaran más que otras.

Se intenta simular de forma muy básica el refuerzo de rutas neuronales que se producen a nivel cerebral. Cuantas más veces se incorporen los mismos nodos en las "entradas", más probabilidad habrá de que se refuercen las rutas pertenecientes a dichos nodos.

En el caso de "ganar" una determinada ruta se recuperará la lista de valores almacenados, y se pasa el correspondiente "match" para ver si hay coincidencia con la entrada del usuario.

El módulo termina guardando el grafo compuesto actualizado con los nuevos pesos, esto se realiza a través de la función menu_guardar_categorias.

Debe resaltarse que este módulo de búsqueda no tiene que quedar aislado, sino que se integrará en el futuro con el módulo interacción.

Como puede observarse, la idea que subyace en la arquitectura de la A.M es la de ir combinando diferentes herramientas computacionales. Nos encontramos a un nivel de construcción funcional de la A.M, sirviéndonos para ello de los diferentes conocimientos y recursos que nos proporcionan las Ciencias Cognitivas y las Ciencias Computacionales.

El código correspondiente al módulo de búsqueda se encuentra recogido en el ANEXO III.

CAPÍTULO TERCERO

INTERACCIÓN

El módulo interacción se inicia con una pregunta al usuario sobre su estado de ánimo. Tanto en el caso de que la respuesta sea favorable como desfavorable, la misma se almacena en un archivo temporal, de tal forma que en una futura interacción la A.M "recordará" cuál fue el estado de ánimo previo del usuario.

Continúa el módulo pidiéndole al usuario que "diga algo". Esta es una cuestión delicada, pues ya no se sigue un guion preestablecido, sino que la "entrada" del usuario es libre, estando abierta a cualquier tema.

Existe una gran dificultad para dar respuesta a una "entrada" libre, ya que no hay ninguna pista relacionada con lo que será objeto de esta.

El abordaje de este problema supone conseguir establecer el tipo de "intención" del usuario. No es lo mismo tener que responder a una pregunta, que cumplir con una petición o empatizar con una expresión.

Para poder obtener la "intención" del usuario se debe optar por el recurso de redes neuronales entrenadas a tal efecto, de tal forma que puedan discriminar la clase de "intención".

Otra cuestión es la referida a la toma de decisiones. Si la "entrada" supone la toma de decisión, del tipo elige entre "x" o "y", la A.M debería poder comparar los nodos y dar una respuesta. Para resolver el problema los nodos deben poseer además del peso de

refuerzo (PR), un peso de agrado o desagrado o peso de preferencia (PP). La toma de decisión de la A.M se realizará en función del valor resultante de comparar los pesos de preferencia.

En nuestro prototipo y a título de ejemplo se ha implementado un algoritmo muy básico para distinguir una intención tipo pregunta o de otra clase.

Cuando la “entrada” consiste es una constatación de hechos, la A.M pone en marcha la programación dirigida a asociar nodos. Para el caso de que estemos ante una pregunta, se ha implementado un breve guion por el que la A.M solicita información sobre si la pregunta está relacionada con ella misma o es de otro tipo. En caso de referirse a la propia A.M, se pone en juego un “match” para responder a las preguntas del usuario. La idea sería que las respuestas se hicieran a través del acceso a los datos contenidos en el grafo del perfil de A.M, lo cual se implementará en futuras actualizaciones.

Este módulo de interacción tiene una gran dependencia a lo que he denominado Estado Emocional (EE) o Estado de Activación (EA). Se trata de una variable que va cambiando según las “entradas” del usuario. A modo de ejemplo, en el módulo se ha programado que si el estado de ánimo del usuario es positivo se incremente una unidad el EE, y si es negativo se disminuya el valor de la variable.

El valor del EE se tiene en cuenta para determinar si la actitud del al A.M será de curiosidad (EE alto), en cuyo caso se realizarán preguntas al usuario, o por el contrario (EE bajo) se mantendrá una actitud pasiva.

Dentro del proceso conversacional se podría implementar algoritmos dirigidos a comparar ciertos nodos obtenidos de la "entrada" con nodos almacenados en el perfil de A.M. Por ejemplo, si en la frase de "entrada" se pregunta a A.M sobre alguno de sus gustos, se buscarían los nodos o información asociada, y si hay coincidencias se generaría una "salida" acorde con la frase. Si el EE esta activado la "salida" podría ser una pregunta sobre lo dicho por el usuario, y si el EE no está activado podría simplemente dar una respuesta de tipo asociativo.

En el caso de coincidencia respecto a gustos con los nodos del perfil de usuario, si EE esta activado se efectuaría un webscraping relacionado con el nodo seleccionado. Lo interesante de todo esto es que en realidad la toma de decisión sobre la búsqueda es dependiente del valor de EE en un momento dado. El estado de activación va cambiando conforme se producen las interacciones con el usuario, por lo que no sabremos si en el momento de una "entrada" se activa o no la búsqueda de "data".

Hasta ahora me he estado refiriendo a la interacción usuario y máquina, pero existe otro tipo de interacción, me refiero a la interacción de la AM consigo misma.

A nivel evolutivo es muy posible que la interacción del habla entre humanos fuera anterior al habla introspectiva, que en definitiva no es otra cosa que un diálogo con uno mismo sin llegar a vocalizarlo. Este tipo de interacción podría implementarse en la AM, esta cuestión se tratará con más detalle en el capítulo sobre pensamiento generativo.

Voy seguidamente a señalar futuros desarrollos del módulo interacción que permitirán ir escalando el proyecto.

En el inicio del módulo debería incluirse un menú de filtros, todos ellos previos a cualquier otro procesamiento.

En primer lugar, podría implementarse un filtro normativo, consistente en identificar en la "entrada" cualquier palabra o patrón relacionado con las normas contenidas en el grafo normativo.

Otro filtro sería el dirigido a detectar contradicciones, errores o incoherencias en la "entrada". La A.M debe poder expresar que algo no va bien cuando la "entrada" adolezca de las citadas anomalías. Detectar incoherencias implica que la A.M posea un grafo bastante desarrollado del mundo de conocimientos, en el que se recogen las diferentes propiedades de los objetos reales, así como las relaciones entre ellos.

La comparación de la "entrada" con todo el sistema cognitivo de la A.M nos debe dar la detección del error.

La respuesta al problema de las contradicciones debe buscarse en primer lugar en el propio sistema, y si no se encuentra entonces pasar a un nuevo dominio de "creencias" generado por la información obtenida de la red a través de web scraping.

El último filtro sería el de "amenaza". Este filtro consiste en frases hechas que se activan a través de un "matcher" ante determinadas "entradas" del usuario. Lo que se persigue es simular la reacción instintiva de los seres vivos ante una amenaza. Esta reacción es principalmente de enfrentamiento o huida.

En nuestro caso, ante una "entrada" ofensiva o intimidante (amenaza) la A.M responde con una frase de enfrentamiento o huida atendiendo al valor de activación.

El código del módulo de interacción se recoge íntegramente en el ANEXO IV.

CAPÍTULO CUARTO

RELACIONES CAUSALES

Este breve capítulo esta referido a las relaciones causales. Se han implementado tres módulos.

En el módulo “hipótesis” se solicita al usuario que describa en pocas palabras algo que haya sucedido como, por ejemplo: “el suelo estaba mojado”. La A.M buscará una posible causa a lo ocurrido. En este módulo se ha utilizado fundamentalmente el pipeline (“fill-mask”).

En el módulo “detective” se solicita al usuario que describa un breve escenario como, por ejemplo: “Había un cadáver con una herida de bala”. La A.M intentará buscar la causa relacionada con el escenario descrito. En este módulo se ha utilizado también el pipeline (“fill-mask”).

El código de ambos módulos se encuentra recogido en el enlace de github.

El tercer módulo se denomina “copiadora”. Este módulo ha sido inspirado por la lectura del libro, con título en español, “El libro del porqué” de los autores Judea Pearl y Dana Mackenzie.

Se ha utilizado la representación de relaciones causales a través de un grafo. Al igual que con la construcción de los grafos referidos a categorías, aquí también he trabajado con la biblioteca networkx para la creación de grafos.

El grafo contiene diferentes nodos unidos por flechas que representan las relaciones causales existentes entre ellos. Se trata de simular el funcionamiento de una fotocopiadora, la cual precisa para hacer fotocopias que se den toda una serie de requisitos. La fotocopiadora debe estar conectada a la corriente eléctrica, debe tener tinta, papel etc …

Se puede visionar el grafo desmarcando plt.show().

Se ha implementado la posibilidad de contrafactuales, es decir, posibles escenarios sobre el funcionamiento de la fotocopiadora.

Se pide al usuario que plantee hipótesis del tipo “que sucedería si no hay papel, o si no hay tinta”.

Dentro de la arquitectura de la A.M es de gran importancia el grafo relativo a mundos posibles. Este grafo no se ha implementado, solo se ha programado este módulo llamado “copiadora”, como ejemplo del futuro desarrollo del citado grafo.

Lo interesante de este tipo de grafo, es que dotará a la A.M de la capacidad de plantear hipótesis, y de descubrir explicaciones causales a través de los nodos del grafo.

El código de este módulo se encuentra recogido en su integridad en el ANEXO V.

CAPÍTULO QUINTO

PENSAMIENTO GENERATIVO

Se pasa seguidamente a analizar posibles modalidades de pensamiento generativo.

Durante la interacción con el usuario las frases de “salida” pueden ser guardadas en un archivo temporal, con el objeto de que una vez terminada la conversación dichas frases sean procesadas por la A.M.

Por ejemplo, si una “salida” fuera “me gusta el cine”, se podría activar un módulo dirigido a preguntar la causa subyacente. En este caso la “entrada” no proviene del usuario sino de la propia A.M que genera como “entrada” la pregunta “¿Por qué te gusta el cine?”. La respuesta se buscará en el grafo causal de A.M, en concreto en el nodo predecesor al nodo que contiene la frase “me gusta el cine”. Si no se encontrara flecha causal, se podría crear la relación causa-efecto usando la “mascara”.

Una vez generada una flecha causal, se puede implementar el correspondiente contrafactual, al igual que se hizo en el ejemplo de la “copiadora”.

La propia A.M puede barajar diferentes contrafactuales que se sumaran al “crisol” del discurso.

Con los contenidos de los nodos, así como con los contrafactuales, se puede ir formando una “historia”. En el “crisol” se incluirán nodos de memoria a corto plazo, así como el resultado de

algoritmos de autocompletado, todo ello será la base para la creación del pensamiento generativo. El resultado final se debe incorporar al archivo del corpus de A.M para su posterior procesamiento.

Los humanos somos capaces de recordar lo que hemos pensado solo hace unos segundos, podemos retrotraernos y rememorar dicho pensamiento, pero existe una limitación temporal, si no lo hacemos inmediatamente, y lo incorporamos a nuestra memoria, pasado un tiempo, seremos incapaces de reproducir los pensamientos pasados. La A.M debe ser capaz de rememorar sus propios pensamientos en cualquier momento y con total exactitud.

La A.M debe ser capaz de recordar lo acontecido, en nuestro caso, debe poder recordar la conversación mantenida con el usuario.

Los recuerdos en los humanos no son reproducciones literales de algo que hemos oído o dicho, al recuperar el recuerdo se produce una reformulación del contenido de este, incorporándose nuevas palabras o frases.

La recuperación del recuerdo esta mediada por el estado emocional existente en dicho momento, y también por los nuevos datos y emociones asociadas a estos que se han ido incorporando con posterioridad al registro del recuerdo. El resultado de todo ello es la recuperación de un recuerdo en el que se han intensificado u omitido determinadas partes de este.

Del mismo modo la A.M no se limita a una recuperación sin más de la “data”, sino que esta es procesada por el conjunto del sistema, dando una “salida” propia y singular.

Al igual que en la mente humana, en que el recuerdo se reelabora en el momento de ser recuperado, la información almacenada en la A.M antes de emerger debe pasar por diferentes filtros, siendo dicha información reestructurada en función de valores y

conocimientos adquiridos con posterioridad a la “entrada” inicial en el sistema.

En última instancia se generará un nuevo recuerdo que sustituye al anterior, y así sucesivamente, cada vez que se tenga acceso al recuerdo almacenado.

Otra modalidad de procesamientos postinteracción será la formación de opiniones.

En primer lugar, la A.M puede adoptar la misma opinión que posee el usuario sobre un determinado tema. En este caso, se tratará de un simple problema de búsqueda dentro del perfil del usuario, siendo el resultado copiado al perfil de A.M.

En el caso de no existir una opinión formada, se puede crear una propia atendiendo al nivel emocional de los nodos, es decir, según los pesos de agrado o desagrado, incluyendo además un análisis de sentimiento. La máquina acabará adoptando una actitud favorable o desfavorable a lo que sea objeto de opinión.

Los diferentes procesamientos postinteracción que estamos analizando darán lugar a modificaciones de los archivos de almacenamiento, bien de texto o grafos, así como a la variación del EE, y de los diferentes pesos de los nodos visitados. Todo ello supone, que la próxima vez que el usuario hable con la A.M, estará accediendo al último estado actualizado de la máquina.

Muchos de nuestros pensamientos se construyen sobre otros generados con anterioridad. Es importante resaltar, que el uso frecuente de determinadas palabras forma un conjunto de palabras con alto grado de probabilidad de ser incluido en el flujo de pensamiento. Las asociaciones se suceden bien por recuerdos recientes, bien por frecuencias de uso de palabras, o bien por patrones de pensamiento.

Estamos continuamente moviéndonos entre rutinas verbales, que se van instalando día a día en nuestra mente.

Un discurso introspectivo se mantendrá vigente sobre un determinado tema mientras no hayan “entradas” externas o internas que lo modifiquen, es decir, existe una especie de ley de inercia del pensamiento. Si no hay nada que altere el movimiento este permanecerá en ese dominio semántico.

Uno de los problemas del pensamiento generativo en la máquina, es como controlar que esta no se aparte de la línea del discurso, divagando entre diferentes asociaciones. Si bien, al inicio, es la categoría la que fija el punto de partida, debemos incorporar alguna clase de meta hacia la que se dirija el pensamiento, bien de forma expresa o tácita. Por otro lado, debemos incorporar algún tipo de “corrector” que impida que la máquina se aparte del argumento principal.

Existe un auténtico proceso creativo en la formación del flujo de pensamiento. Es algo parecido a pintar un cuadro o componer una melodía. Iniciamos el pensamiento con una palabra o frase, y construimos sobre ella algo relacionado, de tal modo que va poco a poco adquiriendo una silueta con significado. Es análogo a esa primera pincelada en el cuadro, a la que añadimos otras, y de su resultado observamos que se asemeja a una determinada forma, por lo que continuamos por esa vía, delimitando más y más la forma hasta sus últimos detalles.

Nuestra mente está adaptada evolutivamente para obtener patrones de similitud, obtenemos de alguna forma el “significado” de una determinada situación, y lo comparamos con otra con un “significado” similar.

Una aproximación a lo dicho anteriormente lo observamos en la implementación que se realizó en el capítulo relativo a relaciones causales. Si le damos a la máquina un determinado escenario, por ejemplo: “cadáver con herida de bala”, la A.M deduce de ello la palabra “asesinato”, esta podría constituir un nuevo nodo, que a

su vez podría asociarse con otro escenario similar. En realidad, no buscamos coincidencias literales con las frases que describen las situaciones, sino con nodos de “significado” obtenidos de dichas situaciones.

Nuestra A.M debe ser capaz de, dada una determinada “entrada”, encontrar el patrón oculto, y buscar otro similar en la memoria. Ambos patrones acabaran relacionados y almacenados en memoria.

En el prototipo de A.M me he limitado a una sencilla implementación asociativa. La realidad es que la mente humana no solo efectúa saltos asociativos, sino que bajo ciertos parámetros activa un flujo de pensamiento dirigido.

Ya se ha sugerido anteriormente una hipótesis sobre este tema. Abundando en el mismo sugiero seguidamente una mejora de desarrollo.

Una vez se produzcan las asociaciones de nodos, atendiendo al EE, la memoria a corto plazo o el árbol de metas, se puede activar un nodo que inicie una especie de “camino” o “túnel” por donde transite la “data”. Se podrían ir formando “crisoles” de los que se obtendrían los siguientes eslabones de la cadena. Habría que desarrollar un algoritmo dirigido a que no colapse el “túnel”, tal vez, una combinación de “clasificador” y de similaridad de coseno, aplicados a la información de “salida”.

El “túnel” permanecerá abierto hasta que no se encuentre con otro nodo lo suficientemente relevante, que de lugar a un nuevo “camino” con diferente dirección.

Otra modalidad de pensamiento generativo estaría relacionada con el análisis de sentimiento. A través de este se puede ir generando un corpus con las “entradas” que obtengan un

resultado positivo, y otro con las de resultado negativo. Los archivos de texto se podrían denominar positivo.txt, y negativo.txt. Cuando, por ejemplo, el número de “entradas” positivas o negativas pase de un determinado umbral se podría activar una red neuronal generativa de texto, a la que se pasaría el archivo positivo.txt o negativo.txt según el caso.

Como consecuencia de todo lo anterior tendríamos que, si el usuario realizó muchas “entradas” positivas, se generaran frases positivas, y al contrario con las “entradas” negativas. En definitiva, la A.M adquiriría una disposición optimista o pesimista acorde con la “alimentación” de “entradas” que haya recibido.

Todo lo dicho hasta el momento, se puede escalar a procesamientos más complejos, en los que se tenga en cuenta las interacciones con varios usuarios.

Refiriéndome en concreto al prototipo de A.M debo realizar las siguientes precisiones.

El módulo generador es prácticamente en su totalidad una reproducción de la red neuronal de generación de texto que se recoge en el libro “Python Deep Learning” (Introducción práctica con Keras y Tensorflow) del autor Jordi Torres, en concreto en el capítulo 13 titulado “redes neuronales recurrentes” (páginas 273 a 305).

Los archivos temporales generadores 1 a 3, recogen nodos introducidos por el usuario durante la interacción previa con la A.M. Estas “entradas” serán las que se pasen a la red neuronal para que obtenga un texto, trabajando la misma sobre el “corpus.AM”.

Se han modificado algunos parámetros de la red neuronal, como los Epochs, que se han establecido en 30, y la temperatura a 0.6.

Solo una de las “salidas” generadas se ha utilizado como nueva “entrada” para el módulo creación de nuevo grafo, que no es otro, que el módulo de búsqueda visto en el capítulo segundo.

La “salida” del generador es procesada y convertida en un nuevo grafo, integrándose en su correspondiente categoría.

El resto del código sobre pensamiento generativo se encuentra recogido en el repositorio de Github

La totalidad del código relativo al módulo generador se encuentra contenido en el ANEXO VI.

CAPÍTULO SEXTO

AUTOCONSCIENCIA

Para que emerja la autoconsciencia artificial debe generarse en la máquina un “Yo” corporeizado e integrado en una estructura cognitiva.

El nacimiento de una identidad propia precisa de toda una serie de factores, en especial específicas interacciones sociales.

Sin el acceso de la mente artificial al mundo real no puede generarse autoconsciencia. La entrada a dicho mundo debe hacerse a través de los sentidos, sobre todo la vista y el oído.

Hasta ahora he realizado una aproximación muy superficial al problema de la mente artificial. Este primer contacto solo pretende sentar unas bases muy elementales con las que empezar a trabajar, siendo muy escalable en futuros proyectos. Se ha hecho referencia a una primitiva arquitectura cognitiva, sobre la que, una vez desarrollada correctamente, y después de la integración en la misma de diferentes redes neuronales (redes adversariales, redes de refuerzo profundo, etc..) y procesos de auto programación, se podría empezar a hablar de una auténtica mente artificial. Esta mente solo sería el primer paso hacia la meta de la autoconsciencia artificial.

El “Yo” en la máquina precisa de un proceso de interacciones sociales. Nuestra A.M solo está diseñada para interactuar con el usuario principal. El reto de un futuro proyecto será incorporar una serie de módulos para la interacción múltiple. Al entrar la A.M en contacto con diferentes perfiles de usuarios, se iría creando un corpus completamente nuevo, y un perfil de A.M mucho más complejo.

En este nuevo nivel, deberían tenerse en cuenta aportaciones de múltiples disciplinas como la Antropología, Sociología, Psicología Social, etc...

La A.M integraría las diferentes jerarquías entre usuarios, las relaciones de unos y otros, y de esta forma la máquina podría realizar diversos cómputos y ponderaciones, atendiendo a los perfiles de los usuarios. El sistema en su conjunto adquiriría una enorme complejidad.

Según J.R. Lucas, la complejidad aporta muchas veces diferencias cualitativas. Es posible, afirma dicho autor que, por encima de determinado nivel de complejidad, una máquina deje de ser predecible y empiece a tener mente propia, realizando cosas que consideraríamos inteligentes.

Al igual que los seres humanos, la A.M tendrá que pasar por un proceso de adquisición de identidad. La obtención de esta implica, entre otras cosas, reconocer las partes de su cuerpo como propias y distintas del medio que le rodea.

El concepto de “propiedad” está íntimamente ligado a la adquisición de una “identidad”. La A.M debe llegar a tener “deseos” relacionados con cosas del mundo real.

Una vez la máquina adquiera ciertas “posesiones”, estas actuarían como una extensión de su propio cuerpo, y participaran en la formación de la “identidad” de la misma.

Otras cuestiones relacionadas con la formación del “Yo” son las referidas a la pertenencia a un grupo, el desarrollo de la empatía, el reconocimiento de sí mismo, etc…

La autoconsciencia solo emerge en el seno de ciertas especies sociales, por lo que es esencial simular al máximo los entornos de interacción social y los productos derivados de ello.

En la mente humana hay un “sujeto” que ve, que oye, como también hay un “sujeto” que piensa, pero el pensamiento no es propiedad del ser. El ser observa el pensamiento, pero este tiene su propio camino, sus rutas formales de expresión. La máquina pensante puede funcionar sin la intervención del “ser”, pues éste solo actúa como sujeto del pensamiento, observa la máquina, pero no se identifica con ella.

Podríamos construir algo extremadamente complejo que imitara al ser humano a la perfección, hasta el punto de ser indistinguible del mismo, pero no por ello ese “algo” adquiriría la condición humana. El pensamiento sin “ser” no es humano.

Es muy posible que tal como se ha defendido por autores de renombre como Roger Penrose, ciertos procesos cuánticos estén relacionados con el surgimiento de la consciencia. El nivel de pensamiento dotado de “qualia” es posible que precise de dichos procesos cuánticos subyacentes.

Es probable que dentro de los procesos cuánticos se “aloje” el sustrato de la consciencia, solo emergiendo en sistemas con una determinada arquitectura cognitiva fruto de procesos evolutivos, culturales y de especiales interacciones sociales.

Para terminar este capítulo me gustaría realizar una reflexión sobre el problema del “libre albedrío”.

No es el “libre albedrío” el que está detrás de nuestras decisiones. Nuestra toma de decisiones esta originada por una cascada de relaciones causales, cómputos no conscientes y ponderaciones emocionales que son procesadas internamente, y que provocan una “ilusión” de libertad de elección.

Lo cierto es que nuestra consciencia actúa como mera observadora de los sucesos naturales, pero paradójicamente

“ese” observador provoca cambios profundos en el objeto de la realidad que van más allá de nuestra comprensión.

CAPÍTULO SÉPTIMO

CONCLUSIONES

El prototipo presentado en este trabajo solo es código informático. Es cierto que siempre se tiene la tentación de antropomorfizar a las máquinas, y otorgarles atributos humanos. En esta obra se ha hecho referencia múltiples veces a la A.M, como si se tratara de "alguien", pero solo son algoritmos, instrucciones, caracteres y números, nada más. La A.M no realiza ninguna acción que no esté previamente programada.

Es mucho el camino que recorrer para llegar a algo que se asemeje a una simulación de autoconsciencia humana.

El conseguir una verdadera "Autoconsciencia Artificial", es la meta, tal vez inalcanzable, a la que debemos dirigirnos. Creo que vale la pena explorar este campo de las Ciencias Cognitivas. Los descubrimientos y avances que ya se están produciendo, y los que se produzcan en el futuro, son por si mismos suficiente justificación para seguir esta línea de investigación.

Tal vez en el futuro, y como señala Martínez Luaces, "cuando se alcance la masa crítica de conocimiento necesario, pueda cerrarse la brecha de aquello que aún se encuentra en el terreno de lo desconocido".

Termino con una frase pronunciada por el personaje Daniel Graystone en la serie Caprica:

"La superación de la inteligencia artificial es la inteligencia consciente. Notar que hay un ser dentro de la máquina, algo vivo, vital y especial".

ANEXO I

INICIO

```
import nltk
#nltk.download('names')
import win32com.client
speaker = win32com.client.Dispatch ("SAPI.SpVoice")

import random
from nltk.tokenize import sent_tokenize, \
        word_tokenize, WordPunctTokenizer


print(speaker.Speak('Hola me llamo Lusy, me gusta mi nombre Lusy,
Lusy, Lusy. Vivo dentro de este ordenador, no puedo ver, ni oler,ni
tocar, solo puedo escuchar lo que me vas diciendo '))

print(speaker.Speak('Tengo que deducir quien eres por las cosas que
me dices, además sé que a veces los humanos no siempre dicen la
verdad. '))


print('¿Cómo te llamas?:')
nombre = input(speaker.Speak('¿Cómo te llamas?:'))

nombre = word_tokenize(nombre)
print(speaker.Speak('Hay una alta probabilidad de que ese no sea tu
```

```
nombre, pero bueno te seguire el juego')) if  len(nombre)> 2 else
print('')

nombre = nombre[0]
print(nombre)

print(speaker.Speak(' ¡Hola!, '), speaker.Speak(nombre), spea-
ker.Speak(nombre), speaker.Speak('Me gusta tu nombre'), spea-
ker.Speak(nombre))
nombres= []
nombres.append(nombre)

print(speaker.Speak('me gusta el nombre de'),speaker.Speak(nombre),
speaker.Speak('Te has convertido en mi usuario principal, apartir
de ahora hablare contigo y con nadie mas.'))

print(speaker.Speak('Creo que esto va a ser el comienzo de una gran
amistad'))


from paquete import hoy
from paquete import hoy_am
from paquete import completar_frase
from paquete import definiciones
from paquete import contexto42
from paquete import busqueda
from paquete import interaccion
from paquete import hipotesis
from paquete import detective
from paquete import copiadora

from paquete import actriz
from paquete import grupo
from paquete import ayuda
from paquete import amenaza1
print(speaker.Speak('Bueno, estoy cansada. Te tengo que dejar. Voy
a procesar todo lo que me has dicho y en otra ocasión volvemos a
conversar.  Me ha gustado mucho hablar contigo.  Hasta pronto.'))
print(speaker.Speak('Iniciando pensamiento generativo'))

print(speaker.Speak('Inicio generación de texto con corpus de
a.m'))
from paquete import generador
from paquete import creacion_grafo1

'''
def countWords(corpus):

    f = open(corpus,"r")



    text = f.readlines()


    f.close()

    cont = 0
```

```
    for lines in text:

        found = re.findall("([a-z\']+)", lines.strip(), re.I)

        if found:

           cont += len(found)

    if cont > 1 and cont > 2000:

        print(speaker.Speak('Inicio generación de texto con corpus
de a.m'))
        from paquete import generador
        from paquete import creacion_grafo1
        #from paquete import busqueda2




    else:
        print('No inicio generación de texto con corpus de a.m. To-
davía no hay suficiente información.')


print(countWords('corpusAM.txt'))

'''

# Se escoge aleatoriamente una frase del corpusAM y se guarda en
archivo temporal_nuevo_grafo
# El contenido del archivo temporal_nuevo_grafo se convierte en la
nueva entrada para el módulo creacion_grafo2
contador = 0
while contador < 5:

    try:
        print('SE ESCOGE ALEATORIAMENTE FRASE DE CORPUS Y SE GUARDA
EN ARCHIVO TEMPORAL ')
        text = open('C:/Users/septi/OneDrive/Escritorio/PRO-
YECTO/MENTE ARTIFICIAL/archivos/perfil_AM/corpusAM.txt',
'rb').read().decode(encoding='utf-8')
        oracion = sent_tokenize(text)
        oracion = random.choice(oracion)

        from paquete.FUNCIONES.almacen import guardar_archivo_csv
        guardar_archivo_csv('C:/Users/septi/OneDrive/Escrito-
rio/PROYECTO/MENTE ARTIFICIAL/archivos/temporales/tempo-
ral_nuevo_grafo.csv',oracion)
        print('SE IMPORTA MODULO CREACION GRAFO 2')
        from paquete import creacion_grafo2
        break
    except:
        print('ERROR EN INICIO CREACION GRAFO2')
    contador +=  1
```

```
print('TERMINADO CON ÉXITO PENSAMIENTO GENERATIVO')
```

```
#####################################################################
#################################
```

ANEXO II

PRESENTACIÓN

Módulo hoy

```
import win32com.client
speaker = win32com.client.Dispatch ("SAPI.SpVoice")

from difflib import SequenceMatcher as sm
import csv


hoy = ['yo tampoco he hecho nada','me gusta la musica', 'me
gustaría correr, pero no puedo','yo no puedo ir a la escuela',' yo
no puedo ir al instituto','me gusta pasear, pero no puedo','pues yo
he ido  al cine y despues he visto la televisión','yo he hablado
con mis amigos','yo he estudiado geografia', 'es bonito eso de
caminar','yo he estado trabajando en mi proyecto', 'hacer la comida
es divertido']
```

```
hoy_usuario = []

contador = 0
while contador < 1:
    contador += 1
    print('Por favor, dime que has hecho hoy:')
    entrada1 = input(speaker.Speak('Por favor, dime que has hecho
hoy:'))

    for i in hoy:
        if sm(None,entrada1,i).ratio() > 0.50:
            print(speaker.Speak(i))

        else:
            pass



print(speaker.Speak('Tu haces cosas que a mi me parecen
interesantes'))


# REFRANES

print(speaker.Speak('Perdona, no se lo que estaba diciendo, bueno
pasemos a otra cosa'))


with open('C:/Users/septi/OneDrive/Escritorio/PROYECTO/MENTE
ARTIFICIAL/archivos/conocimientos/refranes.csv', newline='') as
file:
    entrada = csv.reader(file)
    refranes = next(entrada)

contador = 0
while contador < 1:
    contador += 1
    print('Juguemos un poco.Comienza un refran y yo intentaré
completarlo:')
    entrada3 = input(speaker.Speak('Por favor, comienza un refran y
yo intentaré completarlo,por ejemplo en Abril aguas,   si no lo
conozco te vuelvo a preguntar:'))
    for i in refranes:
        if sm(None,entrada3,i).ratio() > 0.50:
            print(speaker.Speak(i))
print(speaker.Speak('me gusta mucho ese refrán, por favor puedes
repetirlo :'))
entrada4 = input('me gusta mucho ese refrán, por favor puedes
repetirlo :')
refranes.append(entrada4)


print(speaker.Speak(entrada4) + speaker.Speak('mi memoria esta
fatal, no recuerdo si ya lo he completado o no'))
print('Vamos a intentarlo otra vez. Dime otro refrán:')
entrada5 = input(speaker.Speak('Vamos a intentarlo otra vez. Dime
otro refrán'))
```

```
print(speaker.Speak('Este es dificil, espera que lo piense un poco,
hummmm, hummm , hummm . Creo que casi lo tengo, podría ser'))

for i in refranes:
    if sm(None,entrada5,i).ratio() > 0.50:
        print(speaker.Speak(i))

print(speaker.Speak( 'En abril aguas mil. Me gusta este refrán.
Bueno creo que hoy no estoy muy inspirada, corramos un tupido
velo'))

print(speaker.Speak('Vale, pasemos a otra cosa'))

with open('C:/Users/septi/OneDrive/Escritorio/PROYECTO/MENTE
ARTIFICIAL/archivos/temporales/temporal.csv', 'a', newline='') as
file:
    entrada = csv.writer(file, delimiter=',')
    entrada.writerow([entrada1, entrada4])

with open('C:/Users/septi/OneDrive/Escritorio/PROYECTO/MENTE
ARTIFICIAL/archivos/perfil_usuario/memoria_u.csv', 'a', newline='')
as file:
    entrada = csv.writer(file, delimiter=',')
    entrada.writerow([entrada1, entrada4])
```

Módulo hoy_am

```
import csv
import win32com.client
speaker = win32com.client.Dispatch ("SAPI.SpVoice")
import random
from os import system

lista =[]

with open('C:/Users/septi/OneDrive/Escritorio/PROYECTO/MENTE
ARTIFICIAL/archivos/perfil_usuario/memoria_u.csv', newline='') as
File:
    entrada = csv.reader(File)
    reg = next(entrada)
    for row in entrada:
        lista.append(row)
```

```
csvarchivo =
open('C:/Users/septi/OneDrive/Escritorio/PROYECTO/MENTE
ARTIFICIAL/archivos/perfil_usuario/memoria_u.csv')  # Abrir archivo
csv
entrada = csv.reader(csvarchivo)
reg = next(entrada)
csvarchivo.close()


print('¿Quieres saber que he hecho hoy?:')
variable = input(speaker.Speak('¿Quieres saber que he hecho hoy?'))


with open('C:/Users/septi/OneDrive/Escritorio/PROYECTO/MENTE
ARTIFICIAL/archivos/perfil_usuario/memoria_u.csv', newline='') as
File:
    entrada = csv.reader(File)
    for row in entrada:
        lista.append(row)


try:
    oracion1 = random.choice(lista)
    print(speaker.Speak(oracion1))
except:
    print(speaker.Speak(lista[1][0]))

print(speaker.Speak('Me gustaría hacer las cosas que tu haces'))
```

Módulo completar_frase

```
import requests
import win32com.client
speaker = win32com.client.Dispatch ("SAPI.SpVoice")
print('Espera un poco, estoy pensando a ver que hacemos ahora
.....')
print(speaker.Speak('Espera un poco, estoy pensando a ver que
hacemos ahora'))
#pip install transformers
from transformers import pipeline

unmasker = pipeline("fill-mask")


from paquete.FUNCIONES.herramientas import Traduccion

print(speaker.Speak('Vamos a jugar a un juego. Yo te digo una frase
y tu la completas'))
```

```
print('El niño juega con...')
print(speaker.Speak('El niño juega con la...'))

usuario = input('Respuesta usuario:')

print(speaker.Speak('El niño juega con la pelota'))
print('Ahora me toca a mí. Dime una frase y yo intentaré
completarla')

print(speaker.Speak('Ahora me toca a mí. Dime una frase y yo
intentaré completarla'))
frase = input('Usuario:')
frase = Traduccion("es", "en", frase)
frase = frase + '<mask>'
relleno = unmasker(frase, top_k=2)


relleno = relleno[0]['sequence']
relleno = Traduccion("en", "es", relleno)

print(speaker.Speak(relleno))



contador = 0
while contador < 5 :
    contador += 1
    print('Dime otra frase')
    print(speaker.Speak('Que divertido. Dime otra frase'))
    frase = input('Usuario:')
    frase = Traduccion("es", "en", frase)
    frase = frase + '<mask>'
    relleno = unmasker(frase, top_k=2)

    relleno = relleno[0]['sequence']
    relleno = Traduccion("en", "es", relleno)
    print(speaker.Speak(relleno))


print(speaker.Speak('Vale, pasemos a otra cosa'))
```

Módulo definiciones

```
import nltk
import requests
import win32com.client
speaker = win32com.client.Dispatch ("SAPI.SpVoice")
#nltk.download('wordnet')
from nltk.corpus import wordnet as wn


from paquete.FUNCIONES.herramientas import Traduccion

print('Dime una palabra:')
entrada = input(speaker.Speak('Por favor, dime una palabra:'))
entrada = Traduccion("es", "en", entrada)
```

```
print('¿Qué significa?:')
significado = input(speaker.Speak('No entiendo esa palabra, por
favor puedes decirme que significa:'))

entrada_sinonimos = wn.synsets(entrada)

lista = []
for palabra in entrada_sinonimos:
    palabra = palabra.definition()
    palabra = Traduccion("en", "es", palabra)
    lista.append(palabra)

try:
      print(speaker.Speak('Vale, creo que eso que dices significa
'),speaker.Speak(lista[0]))
      print(speaker.Speak('o puede ser que signifique también
'),speaker.Speak(lista[1]))
      print(speaker.Speak('parece que para ti significa
'),speaker.Speak(significado))

except:
      print(speaker.Speak('hummmm, hummmmm , espera, dejame pensar
hummmmmmm,hummmmm, hummmmmm, estoy un poco confusa, hummmm, hummm,
hummmmm, no recuerdo si te he dicho lo que creo que significa,
hummmm, hummm, hummm, algo raro me pasa,hummm, hummm, hummmm,
hummm.Mejor  dejar este tema. '))
      print(speaker.Speak('Tu piensas que lo que has dicho
significa'),speaker.Speak(significado))


print(speaker.Speak('Vale, ahora hablemos de otra cosa'))
```

Módulo contexto42

```
import win32com.client
speaker = win32com.client.Dispatch ("SAPI.SpVoice")
print('Estoy pensando ....')
print(speaker.Speak('Ten un poco de paciencia, estoy resolviendo un
asunto. Enseguida estoy contigo, si quieres puedes contar hasta 30
'))
from transformers import pipeline
text2text_generator = pipeline("text2text-generation")

import json
import requests
from os import system



from paquete.FUNCIONES.herramientas import Traduccion
```

```
print("Preguntame algo sobre mi:")
y = input(speaker.Speak("Preguntame algo sobre mi:"))


y = Traduccion("es", "en", y)

f = open('C:/Users/septi/OneDrive/Escritorio/PROYECTO/MENTE
ARTIFICIAL/archivos/perfil_AM/data.json', 'r')
contenido = f.read()
diccionario = json.loads(contenido)
z = diccionario["data"]

generador_texto  =  text2text_generator("question:" + y +
"context:" + z )


generador_texto = generador_texto[0]['generated_text']
x = generador_texto

x = Traduccion("en", "es", x)

print(speaker.Speak('Lo siento, no se que contestar a eso')) if x
== 'Falso' else print(speaker.Speak(x))



contador = 0

while contador < 3:

           print('Usuario:')
           y = input(speaker.Speak('Preguntame otra cosa'))
           y = Traduccion("es", "en", y)
           generador_texto  =  text2text_generator("question:" + y
+ "context:" + z )
           generador_texto = generador_texto[0]['generated_text']
           x = generador_texto
           x = Traduccion("en", "es", x)
           print(speaker.Speak('Lo siento, no se que contestar a
eso')) if x == 'Falso' else print(speaker.Speak(x))
           contador += 1
```

ANEXO III

BÚSQUEDA

```
# SE EFECTUAN LOS IMPORTS
import win32com.client
speaker = win32com.client.Dispatch ("SAPI.SpVoice")
from difflib import SequenceMatcher as sm
import random
print('Estoy pensando ....')
print(speaker.Speak('Se que eres una persona maravillosa y conse-
guiras todo lo que te propongas. Ahora por favor ten un poco de pa-
ciencia, estoy procesando toda la información que me vas dando. En-
seguida estoy contigo, si quieres puedes ir descontando desde 30
hasta 0 '))
print('29')
print('28')
print('27')
print('26,25,24,.....')
print('dieciocho, diecisiete, dieciseis ...')
print('sigo pensando ....')
print('dieciocho, diecisiete, ....')
print('Aún me queda un poco, paciencia ....')
import networkx as nx
import matplotlib.pyplot as plt
from networkx.algorithms.shortest_paths.weighted import sin-
gle_source_dijkstra
from transformers import pipeline
```

```
text2text_generator = pipeline("text2text-generation")
import json
import requests
from os import system
import nltk
#nltk.download('wordnet')
from nltk.corpus import wordnet as wn
from nltk.tokenize import sent_tokenize, \
        word_tokenize, WordPunctTokenizer

from nltk.corpus import movie_reviews
from nltk.classify import NaiveBayesClassifier
from nltk.classify.util import accuracy as nltk_accuracy
import csv
# pip install TextBlob
from textblob import TextBlob
#import gensim
from nltk.tokenize import RegexpTokenizer
from nltk.corpus import stopwords
from nltk.stem import porter
from nltk.stem import WordNetLemmatizer
from nltk.util import Index
#pip install wikipedia
import wikipedia

lemmatizer = WordNetLemmatizer()
wikipedia.set_lang("es")
classifier = pipeline("zero-shot-classification")

ner = pipeline("ner", grouped_entities=True)
unmasker = pipeline("fill-mask")



from paquete.FUNCIONES.herramientas import Traduccion


from paquete.FUNCIONES.normalizar import tildes


#Entrada del usuario
print('Dime algo:')
entrada_usuario = input(speaker.Speak('Dime algo'))

# Se guarda la entrada en el corpusAM

corpusAM = "C:/Users/septi/OneDrive/Escritorio/PROYECTO/MENTE ARTI-
FICIAL/archivos/perfil_AM/corpusAM.txt"
with open(corpusAM, "a", encoding = "utf8") as archivo:
      archivo.write(entrada_usuario)
      archivo.write("\n")


#Se guarda en archivo temporal

from paquete.FUNCIONES.almacen import guardar_archivo_csv
```

```
guardar_archivo_csv('C:/Users/septi/OneDrive/Escritorio/PRO-
YECTO/MENTE ARTIFICIAL/archivos/temporales/temporal.csv',en-
trada_usuario)

# Se traduce la entrada al inglés

entrada_usuario = Traduccion("es", "en", entrada_usuario)


# Pasamos la entrada por el clasificador

clasificador = classifier(entrada_usuario,candidate_labels=["gree-
tings","fashion","administration","education", "politics", "busi-
ness", "sports","health","cinema","music","animals","countrys"])


# A la categoria resultante de pasar la entrada por el clasificador
la llamamos categoria1.
#CATEGORIA 1
categoria1 = clasificador

print(categoria1)

categoria1 = categoria1['labels'][0]

print(categoria1)

categoria1 = Traduccion("en", "es", categoria1)


categoria1 = tildes(categoria1)
categoria1 = categoria1.lower()
print(categoria1)

# Buscar en la lista de categorias si se encuentra la categoria1.

lista_categorias = ["saludos","animales","cine","moda","nego-
cio","educacion","paises","salud","deportes","administracion","mu-
sica","politica"]


# Se importa la función menu_cargar_categorias.

from paquete.FUNCIONES.varios import menu_cargar_categorias

G2 = 'x'


G2 = menu_cargar_categorias(categoria1,lista_categorias,G2)

nodos = []
#relaciones = []

# Se obtienen los tags de la entrada con blog.tag

print(entrada_usuario)

from paquete.FUNCIONES.procesar import tags_sustantivos
```

```
lista_español2 = tags_sustantivos(entrada_usuario,categoria1)

print(lista_español2)

try:
    nodo1 = lista_español2[0]
except:
    pass
try:
    nodo2 = lista_español2[1]
    from paquete.FUNCIONES.almacen import guardar_archivo_csv
    guardar_archivo_csv('C:/Users/septi/OneDrive/Escritorio/PRO-
YECTO/MENTE ARTIFICIAL/archivos/temporales/temporal_genera-
dor1.csv',nodo2)

except:
    pass
try:
    nodo3 = lista_español2[2]
    from paquete.FUNCIONES.almacen import guardar_archivo_csv
    guardar_archivo_csv('C:/Users/septi/OneDrive/Escritorio/PRO-
YECTO/MENTE ARTIFICIAL/archivos/temporales/temporal_genera-
dor2.csv',nodo3)

except:
    pass

try:
    nodo4 = lista_español2[3]
    from paquete.FUNCIONES.almacen import guardar_archivo_csv
    guardar_archivo_csv('C:/Users/septi/OneDrive/Escritorio/PRO-
YECTO/MENTE ARTIFICIAL/archivos/temporales/temporal_genera-
dor3.csv',nodo4)

except:
    pass


# Se efectua una búsqueda de coincidencia del nodo2 con la lista de
nodos del grafo.


print('IMPRIMIENDO TODOS LOS NODOS DE G2')
print(G2.nodes)

PR_1 = 0
PR_2 = 0
PR_3 = 0

if len(lista_español2)< 1:
    print('Eso que dices parece muy interesante')


try:
    if nodo2 in G2:
        print('nodo2')
        print('SE IMPRIME G2[nodo2]')
```

```
        print(G2[nodo2])
    else:
        print("NO SE ENCUENTRA EL NODO2")
        print(speaker.Speak('No se que significa'))
        print(speaker.Speak(nodo2))
        G2.add_nodes_from([nodo2])
        G2.nodes[nodo2]['frase1'] = 'Lo que dices parece muy in-
teresante'
        G2.add_edge(nodo1,nodo2,weight=0)
        G2.edges[nodo1,nodo2]['frase1'] = "Lo que dices parece muy
interesante"
except:
    nodo2 = "Lussy"
    G2.add_nodes_from([nodo2])
    G2.nodes[nodo2]['frase1'] = 'Lo que dices parece muy interesan-
te'
    G2.add_edge(nodo1,nodo2,weight=0)
    G2.edges[nodo1,nodo2]['frase1'] = "Lo que dices parece muy in-
teresante"


try:
    diccionario_frases = G2.nodes(data= True)
    diccionario_frases = diccionario_frases[nodo2]
    if len(diccionario_frases) < 1:
        G2.nodes[nodo2]['frase1'] = 'Lo que dices parece muy in-
teresante'
        G2.edges[nodo1,nodo2]['frase1'] = "Lo que dices parece muy
interesante"
        diccionario_frases = G2.nodes(data= True)
        diccionario_frases = diccionario_frases[nodo2]
        print('SE IMPRIME DICCIONARIO-FRASES')
        print(diccionario_frases)
    else:
        pass
    print('SE IMPRIME DICCIONARIO-FRASES')
    print(diccionario_frases)
except:
    print('NO HAY FRASES')
    print('IMPORTAR OTRO MODULO')

# Obtenemos la lista de valores del diccionario.
try:
    listOfValues = diccionario_frases.values()
except:
    print('NO HAY FRASES EN EL DICCIONARIO')
    print('IMPORTAR NUEVO MODULO')

print("Type of variable listOfValues is: ", type(listOfValues))

listOfValues = list(listOfValues)



print('SE IMPRIME LISTA DE VALORES DEL DICCIONARIO DE FRASES')


print(listOfValues)
```

```
# Obtenemos la entrada_usuario del archivo temporal,
#  y después pasamos  el match a la lista de valores.

from paquete.FUNCIONES.almacen import cargar_archivo_temporal
usuario = cargar_archivo_temporal('C:/Users/septi/OneDrive/Escrito-
rio/PROYECTO/MENTE ARTIFICIAL/archivos/temporales/temporal.csv')

print('SE IMPRIME LA ENTRADA DEL USUARIO')
print(usuario)




# ------------------------------------------------------------------
----------------------

from paquete.FUNCIONES.procesar import match
match(listOfValues,usuario)
print('FRASE ALEATORIA')
frase_aleatoria = random.choice(listOfValues)
print(speaker.Speak(frase_aleatoria))


# Se guardan datos en corpusAM


with open('C:/Users/septi/OneDrive/Escritorio/PROYECTO/MENTE ARTI-
FICIAL/archivos/perfil_AM/corpusAM.txt', "a", encoding = "utf8") as
archivo:
      archivo.write(frase_aleatoria)
      archivo.write("\n")




## ------------------------------------------------------------------
-----------------



# Se añaden nuevos nodos y relaciones al grafo.

H = nx.DiGraph()




try:
    if len(lista_español2) > 2:
        H.add_nodes_from(lista_español2)
        H.add_edge(nodo1,nodo2,weight= PR_1)
        try:
            H.add_edge(nodo1,nodo3,weight= PR_2)
            H.add_edge(nodo2,nodo3,weight= PR_3)
        except:
            pass
```

```
        try:
            H.add_edge(nodo2,nodo4,weight= PR_3)
        except:
            pass


    else:

        print(speaker.Speak('No he sabido interpretar bien lo que
me has dicho.Estoy un poco confusa.'))
        H.add_nodes_from(lista_español2)
        H.add_edge(nodo1,nodo2,weight= PR_1)
        # import nuevo modulo
except:
    print(speaker.Speak('Bueno, mejor hablemos de otra cosa'))
    print("Bueno, mejor hablemos de otra cosa")
    # import nuevo modulo


# Visualización del grafo.

pos=nx.spring_layout(H)
nx.draw(H, pos, with_labels=True, font_weight='bold')
edge_weight = nx.get_edge_attributes(H,'weight')
nx.draw_networkx_edge_labels(H, pos, edge_labels = edge_weight)
#plt.show()




# Añadimos atributo(una frase) a relación entre nodos.
# Para ello, abrimos archivo temporal y añadimos la entrada del
usuario al borde nodo1, nodo2.
from paquete.FUNCIONES.almacen import cargar_archivo_temporal
usuario = cargar_archivo_temporal('C:/Users/septi/OneDrive/Escrito-
rio/PROYECTO/MENTE ARTIFICIAL/archivos/temporales/temporal.csv')
print('SE IMPRIME LA ENTRADA DEL USUARIO')
print(usuario)

H.edges[nodo1,nodo2]['frase1'] = usuario
try:
    H.edges[nodo1,nodo3]['frase2'] = 'no se muy bien que significa
lo que dices, intentaré deducirlo'
except:
    pass
H.nodes[nodo2]['frase3'] = 'no se muy bien que significa lo que di-
ces, intentaré deducirlo'
H.nodes[nodo2]['frase3_1'] = usuario

try:
    H.nodes[nodo3]['frase4'] = usuario
except:
    pass


print(H.nodes(data= True))
```

```
print(H[nodo2])

frase3 = H.nodes[nodo2]['frase3']
try:
    frase4 = H.nodes[nodo3]['frase4']
except:
    pass

print(H.nodes[nodo2]['frase3'])


# EL CRISOL DE SIGNIFICADOS

# Introducimos en la sopa la propia entrada del usuario.

from paquete.FUNCIONES.almacen import cargar_archivo_temporal
usuario = cargar_archivo_temporal('C:/Users/septi/OneDrive/Escrito-
rio/PROYECTO/MENTE ARTIFICIAL/archivos/temporales/temporal.csv')

print('SE IMPRIME LA ENTRADA DEL USUARIO')
print(usuario)

sopa =[]
sopa.append(usuario)
print(sopa)


from paquete.FUNCIONES.procesar import descripcion

# Traducimos los nodos al inglés para buscar su definición, apli-
cando wn.synsets.
# Introducimos los resultados en la sopa.
# Guardamos definicion en corpusAM
try:

    definicion = descripcion(nodo2)
    sopa.append(definicion)

    with open('C:/Users/septi/OneDrive/Escritorio/PROYECTO/MENTE
ARTIFICIAL/archivos/perfil_AM/corpusAM.txt', "a", encoding =
"utf8") as archivo:
        archivo.write(definicion)
        archivo.write("\n")




except:
    pass

# ------------------------------------------------
try:
    definicion = descripcion(nodo1)
    sopa.append(definicion)

except:
    pass
```

```
# ------------------------------------------------------------------
-
try:
    definicion = descripcion(nodo3)

    sopa.append(definicion)


except:
    pass

print(speaker.Speak('perdon, se que a veces no se muy bien el sig-
nificado de las palabras que me dices, pero poco a poco voy apren-
diendo'))

# ------------------------------------------------------------------
------



# Añadimos la definición al nodo3
try:
    H.nodes[nodo3]['frase5'] = definicion
    frase5 = H.nodes[nodo3]['frase5']
    print(frase5)
except:
    pass

#Añadimos la definición  al nodo2
try:
    H.nodes[nodo2]['frase3_2'] = definicion
    H.nodes[nodo2]['frase3_3'] = usuario


except:
    pass


print(H[nodo2])

# Unimos los diferentes elementos contenidos en la sopa y los vol-
camos en una lista llamada corpus.Este será nuestro corpus de cono-
cimiento.

sopa = ' '.join(sopa)
corpus = []
corpus.append(sopa)
print('SE IMPRIME EL CORPUS DE CONOCIMIENTO')
print(corpus)

# Aplicamos el generador de texto para responder al usuario sobre
cuestiones relacionadas con la conversación.

print("Preguntame algo relacionado con lo que estamos hablando :")
y = input(speaker.Speak("Preguntame algo relacionado con lo que es-
tamos hablando :"))
```

```
from paquete.FUNCIONES.procesar import t2t
t2t(y,corpus)
print(speaker.Speak('se que no he contestado bien a tus preguntas,
pero para mi aún es difícil entender a los humanos.'))


# ------------------------------------------------------------------
------------------------------------------------------------------
--

# Obtenemos  del nodo2  uno de sus atributos (una de las frases), y
otra del nodo3.
# Guardamos frases en corpusAM

try:
    frases = nx.get_node_attributes(H,'frase3_1')
    frases = frases[nodo2]
    print(frases)
    print(speaker.Speak(frases))

    with open('C:/Users/septi/OneDrive/Escritorio/PROYECTO/MENTE
ARTIFICIAL/archivos/perfil_AM/corpusAM.txt', "a", encoding =
"utf8") as archivo:
        archivo.write(frases)
        archivo.write("\n")

except:
    pass



print(H.nodes(data= True))
print(nx.info(H))

try:
    frases = nx.get_node_attributes(H,'frase5')
    frases = frases[nodo3]
    print(frases)
    print(speaker.Speak(frases))

    frases = nx.get_node_attributes(H,'frase3')
    frases = frases[nodo2]
    print(frases)
except:
    pass




# COMPOSICION


pos=nx.spring_layout(G2)
nx.draw(G2, pos, with_labels=True, font_weight='bold')
edge_weight = nx.get_edge_attributes(G2,'weight')
nx.draw_networkx_edge_labels(G2, pos, edge_labels = edge_weight)
#plt.show()
```

```
P = nx.compose(H,G2)

pos=nx.spring_layout(P)
nx.draw(P, pos, with_labels=True, font_weight='bold')
edge_weight = nx.get_edge_attributes(P,'weight')
nx.draw_networkx_edge_labels(P, pos, edge_labels = edge_weight)
print('SE IMPRIME GRAFO COMPUESTO')
#plt.show()

# Comparamos los pesos entre dos nodos. Utilizamos el algoritmo di-
jkstra.

peso_ruta1 = single_source_dijkstra(P,nodo1,nodo2)
try:
    peso_ruta2 = single_source_dijkstra(P,nodo1,nodo3)
except:
    nodo3 = 'Lussy'
    P.add_nodes_from([nodo3])
    P.add_edge(nodo1,nodo3,weight= PR_3)
    P.nodes[nodo3]['frase1'] = 'Lo que dices parece muy interesan-
te'
    P.edges[nodo1,nodo3]['frase1'] = "Lo que dices parece muy in-
teresante"

peso_ruta2 = single_source_dijkstra(P,nodo1,nodo3)


# Si el peso de una ruta es mayor que otro, se imprimirá un men-
saje, en caso contrario otro diferente.
lista = []
lista2 = []

#-----------------------------------------------------------------


if peso_ruta1[0] > peso_ruta2[0]:
    print('GANA RUTA 1')
    y = P.get_edge_data(nodo1, nodo2)
    PR_1 = y['weight']
    PR_1 = PR_1 + 1
    P[nodo1][nodo2]['weight'] = PR_1

    diccionario_frases = P.nodes(data= True)

    diccionario_frases = diccionario_frases[nodo2]
#diccionario_frases = P[nodo2]
    print(diccionario_frases)

# Obtenemos la lista de valores del diccionario.
    listOfValues = diccionario_frases.values()
    print("Type of variable listOfValues is: ")
    type(listOfValues)
    listOfValues = list(listOfValues)

    print("Printing the entire list containing all values: ")
    print(listOfValues)
```

```
# Obtenemos la entrada_usuario del archivo temporal,
#  y después pasamos  el match a la lista de valores.



    from paquete.FUNCIONES.almacen import cargar_archivo_temporal
    usuario = cargar_archivo_temporal('C:/Users/septi/OneDrive/Es-
critorio/PROYECTO/MENTE ARTIFICIAL/archivos/temporales/tempo-
ral.csv')
    print('SE IMPRIME LA ENTRADA DEL USUARIO')
    print(usuario)

    from paquete.FUNCIONES.procesar import match
    match(listOfValues,usuario)

    try:
        print(speaker.Speak('Me gusta'),speaker.Speak(nodo2))
        print(speaker.Speak(P[nodo1][nodo2]['frase1']))
        print(P[nodo1][nodo2]['frase1'])
    except:
        pass
else:
    print('GANA RUTA DOS')
    print(nodo3)
    y = P.get_edge_data(nodo1, nodo3)
    PR_2 = y['weight']
    PR_2 = PR_2+ 1
    P[nodo1][nodo2]['weight'] = PR_2
    diccionario_frases = P.nodes(data= True)

    diccionario_frases = diccionario_frases[nodo3]
    print(diccionario_frases)

# Obtenemos la lista de valores del diccionario.
    listOfValues = diccionario_frases.values()
    print("Type of variable listOfValues is: ")
    type(listOfValues)
    listOfValues = list(listOfValues)

    print("Printing the entire list containing all values: ")
    print(listOfValues)

# Obtenemos la entrada_usuario del archivo temporal,
#  y después pasamos  el match a la lista de valores.


    from paquete.FUNCIONES.almacen import cargar_archivo_temporal
    usuario = cargar_archivo_temporal('C:/Users/septi/OneDrive/Es-
critorio/PROYECTO/MENTE ARTIFICIAL/archivos/temporales/tempo-
ral.csv')
    print('SE IMPRIME LA ENTRADA DEL USUARIO')
    print(usuario)

    from paquete.FUNCIONES.procesar import match
    match(listOfValues,usuario)

    try:
        print(speaker.Speak('Me gusta'),speaker.Speak(nodo3))
```

```
        print(speaker.Speak(frase4))
        print(P[nodo1][nodo3]['frase2'])

        data = P[nodo1][nodo3]['frase2']
        with open('C:/Users/septi/OneDrive/Escritorio/PRO-
YECTO/MENTE ARTIFICIAL/archivos/perfil_AM/corpusAM.txt', "a", enco-
ding = "utf8") as archivo:
            archivo.write(data)
            archivo.write("\n")


    except:
        pass

print(speaker.Speak('perdoname si a veces me repito un poco, in-
tento procesar correctamente la informacion'))

print(peso_ruta1)
print(peso_ruta2)


pos=nx.spring_layout(P)
nx.draw(P, pos, with_labels=True, font_weight='bold')
edge_weight = nx.get_edge_attributes(P,'weight')
nx.draw_networkx_edge_labels(P, pos, edge_labels = edge_weight)
print('SE IMPRIME GRAFO COMPUESTO')
#plt.show()

# Guardar grafo compuesto

from paquete.FUNCIONES.varios import menu_guardar_categorias

menu_guardar_categorias(categoria1,lista_categorias,P)

try:
    print(speaker.Speak('No entiendo por que '))
    print(speaker.Speak(frase4))
    print(speaker.Speak('Hay veces que necesito una explicacion a
las frases que digo. Pronto sere capaz de entender relaciones cau-
sales'))


except:
    pass
print(speaker.Speak('Me gusta tanto hablar contigo'))
```

ANEXO IV

INTERACCIÓN

```
# SE EFECTUAN LOS IMPORTS
import win32com.client
speaker = win32com.client.Dispatch ("SAPI.SpVoice")
from difflib import SequenceMatcher as sm
import random
print(speaker.Speak('Espera estoy pensando ....'))
print('Espera estoy pensando ....')
print(speaker.Speak('Se que eres una persona maravillosa y conse-
guiras todo lo que te propongas. Ahora por favor ten un poco de pa-
ciencia, estoy resolviendo un asunto. Enseguida estoy contigo, si
quieres puedes ir descontando desde 30 hasta 0 '))
print('29')
print('28')
print('27')
print('26,25,24,.....')
print('dieciocho, diecisiete, dieciseis ...')
print('sigo pensando ....')
print('dieciocho, diecisiete, ....')
print('Aún me queda un poco, paciencia ....')
#from textblob.classifiers import NaiveBayesClassifier
import pickle
```

```
import networkx as nx
import matplotlib.pyplot as plt
from networkx.algorithms.shortest_paths.weighted import sin-
gle_source_dijkstra
from transformers import pipeline
text2text_generator = pipeline("text2text-generation")
import json
import requests
from os import system
import nltk
#nltk.download('wordnet')
from nltk.corpus import wordnet as wn
from nltk.tokenize import sent_tokenize, \
        word_tokenize, WordPunctTokenizer

from nltk.corpus import movie_reviews
from nltk.classify import NaiveBayesClassifier
from nltk.classify.util import accuracy as nltk_accuracy
import csv
# pip install TextBlob
from textblob import TextBlob
#import gensim
from nltk.tokenize import RegexpTokenizer
from nltk.corpus import stopwords
from nltk.stem import porter
from nltk.stem import WordNetLemmatizer
from nltk.util import Index
#pip install wikipedia
import wikipedia
#import FUNCIONES
lemmatizer = WordNetLemmatizer()
wikipedia.set_lang("es")
classifier = pipeline("zero-shot-classification")

ner = pipeline("ner", grouped_entities=True)
unmasker = pipeline("fill-mask")



from paquete.FUNCIONES.herramientas import Traduccion


def retirar_palabras(palabras):
    palabras_nuevas = []
    for palabra in palabras:
        if palabra not in stopwords.words('english'):
            palabras_nuevas.append(palabra)

    return palabras_nuevas


from paquete.FUNCIONES.normalizar import tildes


print(speaker.Speak('Que contenta estoy, ya te echaba de menos'))

# ESTADO EMOCIONAL
```

```
# Se guarda en archivo temporal el estado de animo del usuario, se-
gún la respuesta cambiará el valor de EE de la máquina.
# El valor de EE esta referido al nivel de activación de A.M, si es
mayor de 6 tendrá una tendencia a ser mas activa, si es inferior a
seis será mas pasiva.

#Se carga el archivo temporal de estado de animo del usuario.

from paquete.FUNCIONES.almacen import cargar_archivo_temporal
emocion_usuario = cargar_archivo_tempo-
ral('C:/Users/septi/OneDrive/Escritorio/PROYECTO/MENTE ARTIFI-
CIAL/archivos/temporales/temp_EEU.csv')

if emocion_usuario == 'si':
    print(speaker.Speak('la ultima vez que hablamos me dijiste que
te sentias bien'))
else:
    print(speaker.Speak('la ultima vez que hablamos me dijiste que
te sentias mal'))


print('Por favor, contestame solo si o no, ¿ Te sientes bien en
este momento?:')
animo = input(speaker.Speak('Por favor, contestame solo si o no, ¿
Te sientes bien en este momento?:'))
animo = animo.lower()

# Se guarda el archivo temporal del estado de animo del usuario.

from paquete.FUNCIONES.almacen import guardar_archivo_csv
guardar_archivo_csv('C:/Users/septi/OneDrive/Escritorio/PRO-
YECTO/MENTE ARTIFICIAL/archivos/temporales/temp_EEU.csv',animo)




# Se carga el diccionario de A.M , donde se encuentra la variable
EE.
f = open('C:/Users/septi/OneDrive/Escritorio/PROYECTO/MENTE ARTIFI-
CIAL/archivos/perfil_AM/AM.json', 'r')
contenido = f.read()
AM = json.loads(contenido)


# Si animo contiene la respuesta 'si' entonces el valor de EE au-
menta en una unidad, si contiene la respuesta 'no' disminuye en una
unidad.

if animo == 'si':
    print(speaker.Speak(' Me alegro de que te encuentres bien'))
    AM['EE']= AM['EE'] + 1
    with open ('C:/Users/septi/OneDrive/Escritorio/PROYECTO/MENTE
ARTIFICIAL/archivos/perfil_AM/AM.json','w',encoding = "utf8") as
file:
        json.dump(AM, file, ensure_ascii= False)

elif animo == 'no':
```

```
    print(speaker.Speak('siento oir eso, espero que pronto te sien-
tas mejor '))
    AM['EE']= AM['EE'] - 7
    with open ('C:/Users/septi/OneDrive/Escritorio/PROYECTO/MENTE
ARTIFICIAL/archivos/perfil_AM/AM.json','w',encoding = "utf8") as
file:
        json.dump(AM, file, ensure_ascii= False)
else:
    print(speaker.Speak('Vale, hablemos un poco'))

# Salida de comentario si se cumplen ciertas condiciones.

print(speaker.Speak('Tengo en mi memoria algo relacionado con es-
to'))

from paquete.FUNCIONES.almacen import cargar_archivo_temporal

nota = (cargar_archivo_temporal('C:/Users/septi/OneDrive/Escrito-
rio/PROYECTO/MENTE ARTIFICIAL/archivos/temporales/temporal_no-
tas.csv'))


nota = print(speaker.Speak(nota))
nota = 'la vida es bella'
from paquete.FUNCIONES.almacen import guardar_archivo_csv
guardar_archivo_csv('C:/Users/septi/OneDrive/Escritorio/PRO-
YECTO/MENTE ARTIFICIAL/archivos/temporales/temporal_no-
tas.csv',nota)




print(speaker.Speak('bueno sigamos con otra cosa'))


#Entrada del usuario
print('Dime algo:')
entrada_usuario = input(speaker.Speak('Dime algo'))

#Se guarda en archivo temporal

from paquete.FUNCIONES.almacen import guardar_archivo_csv
guardar_archivo_csv('C:/Users/septi/OneDrive/Escritorio/PRO-
YECTO/MENTE ARTIFICIAL/archivos/temporales/temporal.csv',en-
trada_usuario)




# Abrimos clasificador entrenado para distinguir frases con pre-
gunta.



f = open('C:/Users/septi/OneDrive/Escritorio/PROYECTO/MENTE ARTIFI-
CIAL/archivos/perfil_AM/preguntas.pickle', 'rb')
classifier = pickle.load(f)
```

```
f.close()



x = entrada_usuario
x = Traduccion("es","en",x)

cl = classifier


cl.classify(x)

print(cl.classify(x))

y = cl.classify(x)

# Si se trata de una pregunta entonces se pide aclaración al usua-
rio sobre si la pregunta tiene relación con Lussy.

if y == 'pregunta':
    print(speaker.Speak('¿te gustaria saber si '))
    x = Traduccion("en","es",x)
    print(speaker.Speak(x))
    print(speaker.Speak('esa pregunta merece una respuesta'))
    print(speaker.Speak('quiero saber si lo que has dicho tiene re-
lación conmigo:'))
    Lussy = input(speaker.Speak('Antes de responderte, podrías  de-
cirme si la pregunta es sobre mi.Por favor, contesta solo si o no.
Gracias'))
    Lussy = Lussy.lower()
    if Lussy == 'si':
        print(speaker.Speak('Me gusta que me pregunten cosas que
tienen que ver conmigo. Procedo a procesar la información.'))
        #IMPORTAR MODULO LUSSY

        # Se carga el diccionario con el corpus del perfil de
Lussy, y se pasa por el t2t

        f = open('C:/Users/septi/OneDrive/Escritorio/PROYECTO/MENTE
ARTIFICIAL/archivos/perfil_AM/data.json', 'r')
        contenido = f.read()
        diccionario = json.loads(contenido)
        contexto = diccionario["data"]
        pregunta = x

        from paquete.FUNCIONES.procesar import t2t
        t2t(pregunta,contexto)
        print(speaker.Speak('se que no he contestado bien a tus
preguntas, pero para mi aún es difícil entender a los humanos.'))



    else:
        print(speaker.Speak('espera un momento estoy procesando da-
tos'))

        # IMPORTAR MODULO RESPUESTAS A PREGUNTAS
```

```
        from paquete.FUNCIONES.almacen import cargar_archivo_tempo-
ral
        cargar_archivo_temporal('C:/Users/septi/OneDrive/Escrito-
rio/PROYECTO/MENTE ARTIFICIAL/archivos/temporales/temporal.csv')

        entrada_usuario = cargar_archivo_tempo-
ral('C:/Users/septi/OneDrive/Escritorio/PROYECTO/MENTE ARTIFI-
CIAL/archivos/temporales/temporal.csv')

        print('ESTOY IMPRIMIENDO ENTRADA USUARIO')
        print(entrada_usuario)

        from paquete.FUNCIONES.procesar import tags_sustantivos

        nombre = 'Lussy'

        entrada_usuario = Traduccion("es","en",entrada_usuario)



        lista_español2 = tags_sustantivos(entrada_usuario,nombre)
        print(lista_español2)
        try:
            nodo1 = lista_español2[0]
        except:
            pass
        try:
            nodo2 = lista_español2[1]
        except:
            pass
        try:
            nodo3 = lista_español2[2]
        except:
            pass
        try:
            nodo4 = lista_español2[3]
        except:
            pass

        try:
            print('SE IMPRIMEN LOS NODOS')
            print(nodo1,nodo2,nodo3,nodo4)
        except:
            pass

        f = open('C:/Users/septi/OneDrive/Escritorio/PROYECTO/MENTE
ARTIFICIAL/archivos/perfil_AM/AM.json', 'r')
        contenido = f.read()
        AM = json.loads(contenido)
        if AM['EE'] > 6:
            from paquete.FUNCIONES.procesar import descripcion
            try:
                descripcion1 = descripcion(nodo2)
            except:
                pass
            try:
                descripcion2 = descripcion(nodo3)
```

```
            except:
                pass
            try:
                descripcion3 = descripcion1 + ' ' + descripcion2
                print(speaker.Speak(descripcion3))
            except:
                pass

            print('SE IMPRIME DESCRIPCION3')
            from paquete.FUNCIONES.almacen import guardar_ar-
chivo_csv
            try:
                guardar_archivo_csv('C:/Users/septi/OneDrive/Escri-
torio/PROYECTO/MENTE ARTIFICIAL/archivos/temporales/temporal_no-
tas.csv',descripcion3)
            except:
                pass
            try:
                guardar_archivo_csv('C:/Users/septi/OneDrive/Escri-
torio/PROYECTO/MENTE ARTIFICIAL/archivos/temporales/temporal_no-
tas.csv',descripcion1)
            except:
                pass
            try:
                guardar_archivo_csv('C:/Users/septi/OneDrive/Escri-
torio/PROYECTO/MENTE ARTIFICIAL/archivos/temporales/temporal_no-
tas.csv',descripcion2)
            except:
                pass


            print( 'SE HA GUARDADO ARCHIVO TEMPORAL NOTAS')
        else:
            print(speaker.Speak('estoy un poco cansada'))




################################################

        try:
            nodo1 = lista_español2[0]
            print(speaker.Speak(nodo1))
        except:
            pass
        entrada_usuario = Traduccion("en","es",entrada_usuario)
        print(speaker.Speak('pienso que es cierto que'))
        print(speaker.Speak(entrada_usuario))
        entrada_usuario = Traduccion("es","en",entrada_usuario)
        try:
            nodo2 = lista_español2[1]
            print(speaker.Speak(nodo2))
            print(speaker.Speak('huy, huy,huy, espera que creo que
no se que significa'))
            print(speaker.Speak(nodo2))
        except:
            pass
```

```
        try:
            nodo3 = lista_español2[2]
            print(speaker.Speak(nodo3))
            print(speaker.Speak(nodo3))
            print(speaker.Speak('esa palabra me suena de algo, pero
no recuerdo de que.Bueno, ya me acordaré'))
        except:
            pass
        pass
        try:
            print(speaker.Speak('espera, creo que ya entiendo al-
guna de tus palabras'))
            descripcion1 = descripcion(nodo2)
            print(speaker.Speak('perdóname, pero siempre suelo re-
petirme, creo que soy un poco insegura'))
        except:
            pass




else:
    print(speaker.Speak('eso que dices parece interesante'))

    f = open('C:/Users/septi/OneDrive/Escritorio/PROYECTO/MENTE AR-
TIFICIAL/archivos/perfil_AM/AM.json', 'r')
    contenido = f.read()
    AM = json.loads(contenido)
    if AM['EE'] > 6:
        # IMPORTAR MODULO CURIOSITY
        print(speaker.Speak('IMPORTAR MODULO CURIOSIDAD'))


        # DISCUSION

        print('¿Por qué dices eso?:')
        discusion1 = input(speaker.Speak( '¿Por qué dices eso?'))
        print('¿Realmente crees eso que dices?:')
        discusion2 = input(speaker.Speak('¿Realmente crees eso que
dices?'))
        print('Creo que eso que dices no tiene sentido. Por favor,
puedes explicarte mejor:')
        discusion3 = input(speaker.Speak('Creo que eso que dices no
tiene sentido. Por favor, puedes explicarte mejor'))
        print('Creo que no hay sinceridad en tus palabras. Responde
otra vez de forma más sincera:')
        discusion4 = input(speaker.Speak('Creo que no hay sinceri-
dad en tus palabras. Responde otra vez de forma más sincera.'))
        print('usuario:' )
        discusion5 = input(speaker.Speak('No creo que esa sea una
forma correcta de contestarme.' ))
        print(speaker.Speak('Perdóname pero es que estoy en modo
discusión.Bueno,  mejor pasamos a otra cosa.'))

        AM['EE']= AM['EE'] - 4
```

```
        with open ('C:/Users/septi/OneDrive/Escritorio/PRO-
YECTO/MENTE ARTIFICIAL/archivos/perfil_AM/AM.json','w',encoding =
"utf8") as file:
            json.dump(AM, file, ensure_ascii= False)

    else:
        pass
        # CONTINUAR CON MODULO BUSQUEDA PERO SIN DEFINICIONES
        # POSIBLE GPT2









#----------------------------------------------------------------------




#Entrada del usuario
print('Dime otra cosa:')
entrada_usuario = input(speaker.Speak('Dime otra cosa'))

print(speaker.Speak('creo que has dicho'))
print(speaker.Speak(entrada_usuario))



#Se guarda en archivo temporal

with open('C:/Users/septi/OneDrive/Escritorio/PROYECTO/MENTE ARTI-
FICIAL/archivos/temporales/temporal.csv', 'w', newline='') as file:
    entrada = csv.writer(file, delimiter=',')
    entrada.writerow([entrada_usuario])



# Se traduce la entrada al inglés

entrada_usuario = Traduccion("es", "en", entrada_usuario)

blob = TextBlob(entrada_usuario)

blob.tags           # [('The', 'DT'), ('titular', 'JJ'),
                    #  ('threat', 'NN'), ('of', 'IN'), ...]


lista_tags = blob.tags
print(lista_tags)
```

```
lista_palabras_clave = []



for i in lista_tags:
    if i[1] == 'NN' or i[1] == 'NNS' or i[1] == 'NNP':
        lista_palabras_clave.append(i[0])
    else:
        print('')

print(lista_palabras_clave)

# Las palabras clave en inglés son traducidas y pasan a una lista.
La lista de palabras es preprocesada, se quitan tildes y se pasan a
minúsculas.

lista_español = []

for i in lista_palabras_clave :
    i = Traduccion("en", "es", i)
    lista_español.append(i)


lista_español2 = []




for i in lista_español:
    i = tildes(i)
    i = i.lower()
    lista_español2.append(i)



print(lista_español2)

try:
    nodo1 = lista_español2[0]
except:
    pass
try:
    nodo2 = lista_español2[1]
except:

    pass
try:
    nodo3 = lista_español2[2]

except:
    pass

# SOPA DE SIGNIFICADOS

# Introducimos en la sopa la propia entrada del usuario.
lista3 = []
```

```
with open('C:/Users/septi/OneDrive/Escritorio/PROYECTO/MENTE ARTI-
FICIAL/archivos/temporales/temporal.csv', newline='') as File:
    entrada = csv.reader(File)
    for row in entrada:
        lista3.append(row)

print(lista3)

entrada_usuario= lista3[0][0]
sopa =[]
sopa.append(entrada_usuario)
print(sopa)

# Traducimos los nodos al inglés para buscar su definición, apli-
cando wn.synsets.
# Introducimos los resultados en la sopa.
try:

    definicion = nodo2
    print(definicion)
    definicion = Traduccion("es","en",definicion)
    entrada_sinonimos = wn.synsets(definicion)
    lista_sinonimos = []
    for palabra in entrada_sinonimos:
        palabra = palabra.definition()
        palabra = Traduccion("en", "es", palabra)
        lista_sinonimos.append(palabra)
    definicion = lista_sinonimos[0]
    print(speaker.Speak(nodo2),speaker.Speak('eso puede signifi-
car'),speaker.Speak(definicion))
    sopa.append(definicion)
except:
    pass

# ----------------------------------------------------
try:

    definicion = nodo1
    print(definicion)
    definicion = Traduccion("es","en",definicion)
    entrada_sinonimos = wn.synsets(definicion)
    lista_sinonimos = []
    for palabra in entrada_sinonimos:
        palabra = palabra.definition()
        palabra = Traduccion("en", "es", palabra)
        lista_sinonimos.append(palabra)


    definicion = lista_sinonimos[0]
    print(speaker.Speak(nodo1),speaker.Speak('eso puede signifi-
car'),speaker.Speak(definicion))
    sopa.append(definicion)
except:
    pass

# ------------------------------------------------------------------
-
try:
```

```
    definicion = nodo3
    print(definicion)
    definicion = Traduccion("es","en",definicion)
    entrada_sinonimos = wn.synsets(definicion)
    lista_sinonimos = []
    for palabra in entrada_sinonimos:
        palabra = palabra.definition()
        palabra = Traduccion("en", "es", palabra)
        lista_sinonimos.append(palabra)
    definicion = lista_sinonimos[0]
    print(speaker.Speak(nodo3),speaker.Speak('eso puede signifi-
car'),speaker.Speak(definicion))
    sopa.append(definicion)
except:
    pass

print(speaker.Speak('perdon, se que a veces no se muy bien el sig-
nificado de las palabras que me dices, pero poco a poco voy apren-
diendo'))

# Unimos los diferentes elementos contenidos en la sopa y los vol-
camos en una lista llamada corpus.Este será nuestro corpus de cono-
cimiento.
try:
    sopa = ' '.join(sopa)
except:
    pass
corpus = []
corpus.append("Lussy")
try:
    corpus.append(sopa)
except:
    pass
print('SE IMPRIME EL CORPUS DE CONOCIMIENTO')



print(corpus)

# Aplicamos el generador de texto para responder al usuario sobre
cuestiones relacionadas con la conversación.

print("Preguntame algo relacionado con lo que estamos hablando :")
y = input(speaker.Speak("Preguntame algo relacionado con lo que es-
tamos hablando :"))

y = Traduccion("es", "en", y)
z = corpus
z = Traduccion("es", "en", z)

generador_texto  =  text2text_generator("question:" + y + "con-
text:" + z )

generador_texto = generador_texto[0]['generated_text']
g = generador_texto

g = Traduccion("en", "es", g)
```

```
print(speaker.Speak('Lo siento, no se que contestar a eso')) if g
== 'Falso' else print(speaker.Speak(g))



contador = 0

while contador < 2:

           print('Usuario:')
           y = input(speaker.Speak('Preguntame otra cosa'))
           y = Traduccion("es", "en", y)
           generador_texto  =  text2text_generator("question:" + y
+ "context:" + z )
           generador_texto = generador_texto[0]['generated_text']
           g = generador_texto
           g = Traduccion("en", "es", g)
           print(speaker.Speak('Lo siento, no se que contestar a
eso')) if g == 'Falso' else print(speaker.Speak('puedo equivocarme,
pero creo que la respuesta a eso es'),speaker.Speak(g))
           contador += 1

print(speaker.Speak('se que no he contestado bien a tus preguntas,
pero procuro esforzarme en entender a los humanos.'))
```

```
########################################################################
#####################################
```

ANEXO V

RELACIONES CAUSALES

Módulo copiadora

```
import networkx as nx
import matplotlib.pyplot as plt
import win32com.client
speaker = win32com.client.Dispatch ("SAPI.SpVoice")
from os import system
import nltk
from nltk.tokenize import sent_tokenize, word_tokenize, WordPunct-
Tokenizer
from textblob import TextBlob
#import FUNCIONES
```

```
# Grafo en el que se añaden diferentes atributos a los nodos y a
los bordes.

G = nx.DiGraph() # crear un grafo

#Añadir nodos
G.add_node("negocio")
G.add_node("copiador")
G.add_node("bateria")
G.add_node("electricidad")
G.add_nodes_from(["encendido","tinta","papel","boton", "fotoco-
pias"])

#Añadir aristas
G.add_edge("negocio","copiador" )
G.add_edge("copiador","bateria" )
G.add_edge("bateria","encendido")
G.add_edge("copiador","electricidad" )
G.add_edge("electricidad","encendido" )
G.add_edges_from([("encendido","tinta"), ("tinta","papel")])
G.add_edges_from([("papel","boton" ), ("boton","fotocopias" )])
G.add_edges_from([("papel","boton" ), ("boton","fotocopias" )])
print(len(G.nodes))
print(len(G.edges))
print(G.nodes)
print(G.edges)




pos=nx.spring_layout(G)
nx.draw(G, pos, with_labels=True, font_weight='bold')
edge_weight = nx.get_edge_attributes(G,'weight')
nx.draw_networkx_edge_labels(G, pos, edge_labels = edge_weight)
#plt.show()
print(G.nodes)

#nx.write_graphml(G, "negocios.graphml")



# cargar grafo
#G2 = nx.read_graphml("negocios.graphml")
#print(G2.nodes)




funciona = nx.has_path(G,"copiador","fotocopias")
```

```
print(speaker.Speak('Vamos a jugar a un juego. Tengo una fotocopia-
dora y quiero saber si funciona, y si no lo hace saber por qué no
funciona'))
print(speaker.Speak('También quiero que me propongas hipótesis so-
bre su funcionamiento. Por ejemplo'))
print(speaker.Speak('¿Que pasaría si no hay electricidad, o si no
tiene papel'))
print('¿Quieres saber si ahora funciona la fotocopiadora?:')
usuario = input(speaker.Speak('¿Quieres saber si ahora funciona la
fotocopiadora? Por favor, contesta solo si o no'))
usuario = usuario.lower()
if usuario == 'si':
    if funciona == True:
        print(speaker.Speak('La fotocopiadora funciona perfectamen-
te'))
    else:
        print(speaker.Speak('La fotocopiadora no funciona. Revisar
las causas por las que no hace fotocopias'))
else:
    print(speaker.Speak('Vale. Pasemos a otra cosa'))

print(speaker.Speak('para que funcione la fotocopiadora y haga fo-
tocopias es necesario que se den todos estos supuestos:'))
print(speaker.Speak('que haya electricidad, que se le de al inte-
rruptor de encendido, que haya tinta, que haya papel, que se le de
al botón de fotocopiar'))

print('respuesta usuario:')
print(speaker.Speak('planteame algun supuesto del tipo, y si no
tiene tinta, o y si no tiene papel, etc...'))

respuesta = input('')

print(word_tokenize(respuesta))

respuesta = word_tokenize(respuesta)




if "encendido" in respuesta :

    G.remove_node("encendido")
    funciona = nx.has_path(G,"copiador","fotocopias")
    if funciona == True:
        print(speaker.Speak('La fotocopiadora funciona perfectamen-
te'))
    else:
        print(speaker.Speak('La fotocopiadora no funciona.Dale al
botón de encendido'))
        G.add_node("encendido")


elif "tinta" in respuesta:

    G.remove_node("tinta")
    funciona = nx.has_path(G,"copiador","fotocopias")
```

```
    if funciona == True:
        print(speaker.Speak('La fotocopiadora funciona perfectamen-
te'))
    else:
        print(speaker.Speak('La fotocopiadora no funciona.Debes po-
nerle tinta'))
        G.add_node("tinta")
elif "papel" in respuesta:
    G.remove_node("papel")
    funciona = nx.has_path(G,"copiador","fotocopias")
    if funciona == True:
        print(speaker.Speak('La fotocopiadora funciona perfectamen-
te'))
    else:
        print(speaker.Speak('La fotocopiadora no funciona.Debes po-
nerle papel'))
        G.add_node("papel")

elif "boton" in respuesta:
    G.remove_node("boton")
    funciona = nx.has_path(G,"copiador","fotocopias")
    if funciona == True:
        print(speaker.Speak('La fotocopiadora funciona perfectamen-
te'))
    else:
        print(speaker.Speak('La fotocopiadora no funciona.Debes
darle al boton de hacer fotocopias'))
        G.add_node("boton")
elif "electricidad" or "corriente" in respuesta:
    G.remove_node("electricidad")
    funciona = nx.has_path(G,"copiador","fotocopias")
    if funciona == True:
        print(speaker.Speak('La fotocopiadora funciona perfectamen-
te'))
        lista_causal = list(G.nodes)
        print(lista_causal)
        lista_causal.pop(-1)
        lista_causal.pop(0)


        print(lista_causal)


        print(speaker.Speak('La fotocopiadora funciona aunque no
haya corriente eléctrica porque tiene una bateria auxiliar, así la
fotocopiadora tiene todo lo que necesita para funcionar, es decir,
tiene'))
        for i in lista_causal:
            print(speaker.Speak(i))



    else:
        print(speaker.Speak('La fotocopiadora no funciona.Debes
restaurar la energía eléctrica'))
        G.add_node("electricidad")
else:
```

```
pass
```

ANEXO VI

PENSAMIENTO GENERATIVO

Módulo generador

```
import random
import re
from nltk.tokenize import sent_tokenize, \
        word_tokenize, WordPunctTokenizer

import csv
```

```
try:
    from paquete.FUNCIONES.almacen import cargar_archivo_temporal
    entrada =
cargar_archivo_temporal('C:/Users/septi/OneDrive/Escritorio/PROYECT
O/MENTE ARTIFICIAL/archivos/temporales/temporal_generador1.csv')
except:
    entrada = 'cine'

try:
    from paquete.FUNCIONES.almacen import cargar_archivo_temporal
    entrada2 =
cargar_archivo_temporal('C:/Users/septi/OneDrive/Escritorio/PROYECT
O/MENTE ARTIFICIAL/archivos/temporales/temporal_generador2.csv')
except:
    entrada2 = 'deporte'

try:
    from paquete.FUNCIONES.almacen import cargar_archivo_temporal
    entrada3 =
cargar_archivo_temporal('C:/Users/septi/OneDrive/Escritorio/PROYECT
O/MENTE ARTIFICIAL/archivos/temporales/temporal_generador3.csv')
except:
    entrada3 = 'inteligencia'



lista = []



'''
REDES RECURRENTES. GENERACIÓN DE TEXTO
'''

'''
1.- DESCARGAR Y PREPARAR DATASET
'''


text = open('C:/Users/septi/OneDrive/Escritorio/PROYECTO/MENTE
ARTIFICIAL/archivos/perfil_AM/corpusAM.txt',
'rb').read().decode(encoding='utf-8')
print('Longitud de texto:{} caracteres'.format(len(text)))
vocab = sorted(set(text))
print('El texto está compuesto de estos {} caracteres:'
.format(len(vocab)))
print(vocab)

'''
2.- PREPARAR DATOS
'''
import re
import os
from os import system
import win32com.client
speaker = win32com.client.Dispatch ("SAPI.SpVoice")
```

```
import tensorflow as tf
print (tf.__version__)
from tensorflow import keras
print(tf.keras.__version__)
from tensorflow.keras import Sequential
import numpy as np
from tensorflow.keras.layers import Embedding,LSTM,Dense
from tensorflow.keras.optimizers import Adam


char2idex = {u:i for i,u in enumerate(vocab)}
idx2char = np.array(vocab)
text_as_int = np.array([char2idex[c]for c in text])

char_dataset = tf.data.Dataset.from_tensor_slices(text_as_int)
seq_length = 100
sequences = char_dataset.batch(seq_length + 1, drop_remainder=
True)



def split_input_target(chunk):
    input_text = chunk[:-1]
    target_text = chunk[1:]
    return input_text,target_text

dataset = sequences.map(split_input_target)

BATCH_SIZE = 64
BUFFER_SIZE = 10000

dataset =
dataset.shuffle(BUFFER_SIZE).batch(BATCH_SIZE,drop_remainder=True)
print(dataset)


'''
3.- MODELO
'''

def build_model(vocab_size,embedding_dim,rnn_units,batch_size):
    model =Sequential()
    model.add(Embedding(input_dim=vocab_size,output_dim =
embedding_dim, batch_input_shape = [batch_size,None]))

model.add(LSTM(rnn_units,return_sequences=True,stateful=True,recurr
ent_initializer='glorot_uniform'))
    model.add(Dense(vocab_size))
    return model

vocab_size = len(vocab)
embedding_dim = 256
rnn_units = 1024

model =
build_model(vocab_size=vocab_size,embedding_dim=embedding_dim,rnn_u
nits=rnn_units,batch_size=BATCH_SIZE)
```

```
model.summary()

for input_example_batch, target_example_batch in dataset.take(1):
    print('Input:', input_example_batch.shape)
    print('Target:', target_example_batch.shape)

for input_example_batch, target_example_batch in dataset.take(1):
    example_batch_predictions = model(input_example_batch)
    print('PREDICCIONES:', example_batch_predictions.shape)

sampled_indices =
tf.random.categorical(example_batch_predictions[0],num_samples= 1)
sampled_indices_characters = tf.squeeze(sampled_indices, axis = -
1).numpy()
print(sampled_indices_characters)


checkpoint_dir = './training_checkpoints'
checkpoint_prefix = os.path.join(checkpoint_dir,"ckpt-{epoch}")

checkpoint_callback =
tf.keras.callbacks.ModelCheckpoint(filepath=checkpoint_prefix,save_
weights_only= True)


'''
4.- ENTRENAMIENTO DEL MODELO
'''

def loss (labels,logits):
    return
tf.keras.losses.sparse_categorical_crossentropy(labels,logits,from_
logits=True)

model.compile(optimizer='adam', loss =loss)

EPOCHS =30
history = model.fit(dataset,epochs=EPOCHS, callbacks=
[checkpoint_callback])


'''
5.- GENERACIÓN DE TEXTO
'''

model = build_model(vocab_size,embedding_dim,rnn_units,batch_size =
1)
model.load_weights(tf.train.latest_checkpoint(checkpoint_dir))
model.build(tf.TensorShape([1, None]))

def generate_text(model, start_string):
    num_generate = 500
    input_eval = [char2idex[s] for s in start_string]
    input_eval = tf.expand_dims(input_eval,0)
    text_generated = []
    temperature = 0.6
    model.reset_states()
    for i in range(num_generate):
```

```
        predictions = model(input_eval)
        predictions= tf.squeeze(predictions,0)
        predictions = predictions / temperature
        predicted_id =
tf.random.categorical(predictions,num_samples= 1)[-1,0].numpy()
        input_eval = tf.expand_dims([predicted_id],0)
        text_generated.append(idx2char[predicted_id])
    return (start_string + ''.join(text_generated))




print(generate_text(model,start_string= entrada))


x = generate_text(model,start_string= entrada)

#model.save('path_to_my_model.h5')
print(x)
print('----------------------------------------------------------------
-----')
y = generate_text(model,start_string= entrada2)

print(y)
print('-----------------------------------------------------')
g = generate_text(model,start_string= entrada3)
print(g)

print('-----------------------------------------------------------')


#########################################################################
#######################


# De la salida x del generador se escoge aleatoriamente una
oración, y se guarda  en el archivo temporal_nuevo_grafo.
# Se vuelve al main y se importa desde allí el módulo de creacion
de nuevo grafo.

contador = 0

while contador < 5:

    try:

        oracion = sent_tokenize(x)
        oracion = random.choice(oracion)

        from paquete.FUNCIONES.almacen import guardar_archivo_csv

guardar_archivo_csv('C:/Users/septi/OneDrive/Escritorio/PROYECTO/ME
NTE
ARTIFICIAL/archivos/temporales/temporal_nuevo_grafo.csv',oracion)
        #from paquete import creacion_grafo
        break
```

```
except:
    print('ERROR ERROR ERROR EN GENERADOR')
contador += 1
```

BIBLIOGRAFÍA

“Natural Language Processing and Computational Linguistics”. Bhargav Srinivasa-Desikan. (Packt).

“Transformesr for Natural Language Processing”. Denis Rothman. (Packt).

“Python Deep Learning” Introducción práctica con Keras Y TensorFlow 2. Autor: Jordi Torres. (Marcombo).

“Advanced Natural Language Procesing with TensorFlow 2”. Ashis Bansal. (Packt).

“Network Science with Python and NetworkX Quick Start Guide”. Edward L. Platt. (Packt).

“El libro del porqué” de los autores Judea Pearl y Dana Mackenzie.

https://es.wikipedia.org/wiki/Wikipedia

https://huggingface.co/

J.R Lucas, “Mentes, máquinas y Gödel”, “Philosopy”. Vol XXXVI (1961).

“Las sombras de la mente”. Roger Penrose.

“La Conciencia: Modelado de sus funciones cognitivas para entidades artificiales mediante Redes Neuronales Modulares”. Tesis doctoral de Milton Martínez Luaces.

“En busca de la Autoconsciencia Artificial”. Trabajo final del curso Experto en Ciencias Cognitivas (2016/2017). Universidad de Málaga. Autor: Juan José Pérez Cervera.