TP Inicial - Aplicación de la Inteligencia Artificial

Table of Contents

Integrantes

- Franco Disabato <francodisabatobook@gmail.com>
- Suarez Matias <matiassuarez265@gmail.com>
- Ayza Diego <diegoayza82@gmail.com>

Profesores

- Juan Carlos Monteros <jcmonteros@campus.ungs.edu.ar>
- Evelin Aragon <eve_aragon@hotmail.com>

Objetivos

- Investigar y documentar los conceptos de IA en la vida diaria, Onboarding Digital, su importancia y normativas vigentes.
- Implementar un chatbot que interactue con el usuario y le pida la informacion necesaria para confirmar su identidad y/o responder las preguntas que pueda tener sobre el proceso de Onboarding.

Documentacion

A continuacion se documenta la investigacion realizada acerca de Aplicaciones de IA, Onboarding Digital y Normativas Existentes

Aplicaciones de la IA

Medicina

Busca la eficiencia de los equipos de diagnósticos médicos así se podrán descartar errores humanos en el análisis de datos y se reducirían costos en investigación. Algunos avances en esta materia ya se están viendo al usar programas como el IBM Watson y los chatbots para que interactúe con el paciente realizando una serie de preguntas que le permiten formular una hipótesis del posible estado de salud el paciente o para realizar análisis de seguimiento.

Educación

Su aplicación en esta rama seria vital debido a que con sistemas capaces de evaluar a cada estudiante trazando un plan de trabajo con sus necesidades, permitiría al maestro enfocarse directamente a suplir las verdaderas deficiencias de cada estudiante. Además, lo que hace el machine learning es dotar la educación del recurso faltante que para muchos es la calidad.

Construcción

En este campo ha dado grandes resultados debido a que antes la mayoría de los sistemas en las fábricas eran manipulados por los seres humanos esto se deducía en horarios de trabajo cortos y poca producción, si a eso le sumamos tal vez las peligrosas actividades que se debían realizar en algunas fábricas ya que esto reducía la eficiencia. Lo que cambiaría con la llegada del machine learning en combinación con la robótica, lo cual hoy en día ha permitido la automatización de grandes fábricas generando una mayor rentabilidad y una mayor eficiencia.

Finanzas

En este sistema fue una de sus primeras apariciones debido a que muchas industrias necesitaban una plataforma que fuese intuitiva y que permitiera a sus usuarios revisar y verificar sus trámites y transacciones. Mediante programas que recopilaban información de los clientes de muchas entidades financieras, incluso el software IBM Watson hoy en día es de vital importancia en mercados como el de Wall Street ya que realiza la mayoría de las operaciones.

La Robótica

A nivel de esta industria el impacto del machine learning es sorprendente tanto en la actualidad como en sus predicciones a futuro. Donde industrias como Honda con su robot Asimo, han logrado demostrar como un organismo robótico puede suplir a un hombre en el cumplimiento de labores que hoy en día se creían complejas para las maquinas.

Onboarding Digital

Es el proceso de adquirir un nuevo cliente para una nueva compañía, típicamente usando un teléfono celular. El principal propósito del Onboarding digital es verificar la identidad del nuevo usuario para asegurarse de que dicha persona es quien dice ser. Para verificar la identidad de un usuario digital y remotamente se utilizan datos biométricos faciales y tecnología de "detección de vida", combinados con la capacidad de procesar documentos de identidad digitalmente.

La detección de vida es una técnica en donde un algoritmo detecta con seguridad si una muestra biométrica proviene de una representación falsa(foto quieta) o si es un ser humano vivo. La muestra biométrica es una foto facial tomada por el usuario. El algoritmo es capaz de diferenciar una persona viva de otro tipo de ataques, por ejemplo, máscaras, fotos o videos pregrabados. El test se realiza pidiendo al usuario realizar una serie de acciones mientras mira la cámara de su teléfono como: mover los ojos, pestañear, sonreir, mover la cabeza, hablar, etc.

Durante la verificación de identidad se le pide al usuario que cargue su documento de identidad o pasaporte. Se extrae del documento la información personal y foto del mismo. Se verifica la veracidad del documento y si tiene el chip integrado. Se compara la foto selfie con los datos biométricos extraídos del documento de identificación.

Tipos de Machine Learning

Aprendizaje supervisado

En el aprendizaje supervisado, la máquina se enseña con el ejemplo. De este modo, el operador proporciona al algoritmo de aprendizaje automático un conjunto de datos conocidos que incluye las entradas y salidas deseadas, y el algoritmo debe encontrar un método para determinar cómo llegar a esas entradas y salidas. Mientras el operador conoce las respuestas correctas al problema, el algoritmo identifica patrones en los datos, aprende de las observaciones y hace predicciones. El algoritmo realiza predicciones y es corregido por el operador, y este proceso sigue hasta que el algoritmo alcanza un alto nivel de precisión y rendimiento.

Aprendizaje sin supervisión

El algoritmo de aprendizaje automático estudia los datos para identificar patrones. No hay una clave de respuesta o un operador humano para proporcionar instrucción. En cambio, la máquina determina las correlaciones y las relaciones mediante el análisis de los datos disponibles. En un proceso de aprendizaje no supervisado, se deja que el algoritmo de aprendizaje automático interprete grandes conjuntos de datos y dirija esos datos en consecuencia. Así, el algoritmo intenta organizar esos datos de alguna manera para describir su estructura. Esto podría significar la necesidad de agrupar los datos en grupos u organizarlos de manera que se vean más organizados.

A medida que evalúa más datos, su capacidad para tomar decisiones sobre los mismos mejora gradualmente y se vuelve más refinada.

Aprendizaje por refuerzo

El aprendizaje por refuerzo se centra en los procesos de aprendizajes reglamentados, en los que se proporcionan algoritmos de aprendizaje automáticos con un conjunto de acciones, parámetros y valores finales. Al definir las reglas, el algoritmo de aprendizaje automático intenta explorar diferentes opciones y posibilidades, monitorizando y evaluando cada resultado para determinar cuál es el óptimo. En consecuencia, este sistema enseña la máquina a través del proceso de ensayo y error. Aprende de experiencias pasadas y comienza a adaptar su enfoque en respuesta a la situación para lograr el mejor resultado posible.

Normativas del uso de IA

En el ámbito de la Inteligencia Artificial (IA), es fundamental establecer normativas que garanticen su proporcionalidad e inocuidad. Aunque las tecnologías de IA ofrecen numerosos beneficios, su implementación también puede conllevar riesgos para los seres humanos y el medio ambiente. En este contexto, es esencial reconocer la necesidad de evaluar y prevenir posibles daños, así como de promover la seguridad, la equidad y la sostenibilidad en el desarrollo y uso de sistemas de IA. Los principios más importantes son

Proporcionalidad e Inocuidad

Se requieren procedimientos de evaluación de riesgos para prevenir daños humanos y ambientales

Seguridad y Protección

Se deben evitar los riesgos de seguridad y protección a lo largo del ciclo de vida de los sistemas de IA

Equidad y No Discriminación

Promover la inclusión y combatir la discriminación, minimizando los resultados sesgados de los sistemas de IA.

Sostenibilidad

Evaluar continuamente el impacto humano, social, cultural, económico y ambiental de las tecnologías de IA

Derecho a la Intimidad y Protección de Datos

Recopilar y utilizar datos de manera acorde a los principios éticos y legales, respetando la privacidad de los usuarios

Supervisión y Decisión Humanas

Aunque se puedan utilizar sistemas de IA para decisiones, la responsabilidad final sigue siendo humana.

Transparencia y Explicabilidad

Los sistemas de IA deben ser transparentes y explicables para respetar los derechos humanos y las libertades fundamentales

Responsabilidad y Rendición de Cuentas

Implementar mecanismos de supervisión y rendición de cuentas para evaluar el impacto de los sistemas de IA

Sensibilización y Educación

Promover la sensibilización y la comprensión pública sobre las tecnologías de IA y el valor de los datos mediante la educación y la capacitación

Gobernanza y Colaboración Adaptativas

Fomentar la participación de múltiples partes interesadas en la gobernanza de la IA para garantizar una implementación efectiva y ética.

¿Cuál es el objetivo u objetivos que desean lograr con el Chatbot?

Dentro de los objetivos principales que se pueden lograr utilizando el chatbot con IA se pueden encontrar la resolución de dudas sobre productos de los clientes o usuarios, el envío de mails o avisos a los usuarios y la atención 24hs personalizada e inmediata. Con el Chatbot IA se puede elevar la eficiencia de trabajo, generación de leads, recopilación de datos, reducción de costos a la empresa, mejorar estrategia de ventas y/o reducir los tiempos de interacción con los usuarios, lo que hace que el usuario tenga la sensación de estar hablando con una persona real. Aunque parece algo muy básico, puede ser el puntapié inicial de una conversación que dé como resultado un nuevo lead para tu negocio o inclusive una venta. En resumen, el objetivo es la eficiencia, la productividad y la experiencia del usuario en una amplia variedad de aplicaciones y contextos.

Herramientas a Utilizar

- Github como controlador de fuentes y desarrollar en paralelo.
- Python como lenguaje de programacion
- Scikit-learn como biblioteca de herramientas de machine learning.
- Visual Studio Code como IDE

Documentacion Tecnica

Chatbot

Se cargan desde el archivo data.csv las preguntas y respuestas para entrenar al algorito de machine learning.

```
def load_data_from_csv(csv_file):
    questions = []
    answers = []
    with open(csv_file, 'r', newline='') as file:
        reader = csv.reader(file, quotechar='"', delimiter=',', quoting=csv.QUOTE_ALL,
skipinitialspace=True)
        for row in reader:
            questions.append(row[0])
            answers.append(row[1])
    return questions, answers
```

TfidfVectorizer() es una clase en la biblioteca Scikit-learn que se utiliza para convertir colecciones de documentos de texto en representaciones numéricas utilizando el esquema TF-IDF (Term Frequency-Inverse Document Frequency). Básicamente, TF-IDF aumenta proporcionalmente al número de veces que una palabra aparece en un documento, pero se equilibra mediante la frecuencia de aparición de la palabra en el cuerpo. Esto ayuda a identificar las palabras clave que son importantes para un documento en particular en comparación con el cuerpo completo.

```
# Vectorize the text data
vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(questions)
```

NearestNeighbors(n_neighbors=1, algorithm='brute', metric='cosine') es una clase en la biblioteca scikit-learn de Python que se utiliza para realizar búsquedas de vecinos más cercanos en conjuntos de datos. Específicamente, se utiliza para encontrar los vecinos más cercanos de un punto dado en un conjunto de datos. En nuestro ChatBot tenemos algunos parámetros necesarios para poder realizar la respuesta más acertada para la pregunta del usuario: Sintetizando el trabajo de esta función, esta configuración del modelo NearestNeighbors buscará el vecino más cercano para cada punto de consulta utilizando la distancia del coseno y un enfoque de búsqueda de fuerza bruta.

```
# Train a k-nearest neighbors model
knn_model = NearestNeighbors(n_neighbors=1, algorithm='brute', metric='cosine')
knn_model.fit(X)
```

Al momento que el usuario ingresa una consulta, se vectoriza el texto a numeros que puede entender el modelo y se le pregunta al algoritmo a cual de los "vecinos" (las preguntas en este caso) se le asemeja mas. Esto devuelve el index del resultado mas cercano. Y con el index de la pregunta mas cercana se trae la respuesta correspondiente.

```
# Function to generate response
def generate_response(input_text):
    input_vector = vectorizer.transform([input_text])
    _, index = knn_model.kneighbors(input_vector)
    return answers[index[0][0]]
```

Login con huella digital

Instalar liberia Opencv

```
pip install opencv-python
```

Generamos una base en el codigo, en la cual cargamos las huellas digitales las cuales ya estan validadas como usuarios del sistema. La funcion cv2.imread(image, flag) toma como parametros la ubicacion de imagen de la huella digital y un flag que determina el codec con el que se convierte la imagen (en este caso a una escala de grises). Este hace que por cada usuario se guarde una matriz con las caracteristicas unicas de la huella digital leida.

```
# Base de datos de imágenes de huellas digitales (simulada)
database = {
    'usuario1': cv2.imread('img/huella1.jpg', cv2.IMREAD_GRAYSCALE),
    'usuario2': cv2.imread('img/huella2.jpg', cv2.IMREAD_GRAYSCALE),
    'usuario3': cv2.imread('img/huella3.jpg', cv2.IMREAD_GRAYSCALE)
}
```

Al momento de comparar huellas digitales, se comparan las matrices de las huellas digitales con el metodo absDiff() y el mismo devuelve una nueva matriz con la diferencia entre entre ambas matrices. En caso de ser identicas devuelve 0 en todas las posiciones.

Luego np.sum(M) se le pasa la matriz resultante como parametro y devuelve la suma total de diferencias como int.

```
def compare_fingerprints(fingerprint1, fingerprint2):
    # Calculamos la diferencia absoluta entre las dos imágenes
    difference = cv2.absdiff(fingerprint1, fingerprint2)
    return np.sum(difference)
```

Al momento de autenticar al usuario con su huella digital, se establece primero un minimo margen de diferencia aceptable (min_difference) entre la nueva huella ingresada por el usaurio y la almacenada en la base de datos.

Se recorren las huellas almacenadas en la base y se compara la diferencia con la nueva huella ingresada. Si la diferencia es menor al minimo establecido se la toma como candidata a la que corresponde al usuario iniciando sesion y se actualiza la min_difference al nuevo valor. Se sigue iterando por las huellas almacenadas hasta encontrar el mejor candidato (cuya diferencia este mas

cerca de cero). Una vez finalizada la iteracion tenemos 2 resultados posibles: se encontro el mejor candidato y su correspondiente usuario o no se encontro ningun candidato que tenga el minimo de diferencias definido.

Se devuelve el usuario autenticado o None en caso de existir coincidencias.

```
def authenticate_fingerprint(input_fingerprint):
    min_difference = 150000 #float('inf')
    authenticated_user = None

    for username, stored_fingerprint in database.items():
        difference = compare_fingerprints(input_fingerprint, stored_fingerprint)

        if difference <= min_difference:
            min_difference = difference
            authenticated_user = username

    return authenticated_user
```

Se le pide al usuario que ingrese su huella digital (en este caso en forma de path de la imagen). Se convierte la misma en una matriz y se la compara contra la base de huellas pre-cargadas. En caso de la huella coincida (al menos con el minimo de diferencia que hemos configurado) el usuario sera autenticado.

```
    # Read the image file directly from memory
    nparr = np.fromstring(image_file.read(), np.uint8)
    input_fingerprint = cv2.imdecode(nparr, cv2.IMREAD_GRAYSCALE)

    # Autenticación de la huella digital capturada
    authenticated_user = authenticate_fingerprint(input_fingerprint)

    return authenticated_user;

def auth(input):
    try:
        # Simulación de captura de huella digital (cargando una imagen)
        input_fingerprint = cv2.imread(input, cv2.IMREAD_GRAYSCALE)
```

Chatbot como servicio

Flask

Se instalo Flask usando `pip install Flask`

El mismo permite levantar python como un servicio web y configurar endpoints para hacer llamadas por http/https.

```
app = Flask(__name__)
```

Se configuro la ruta de acceso para el chatbot en '/chatbot' y metodo GET. Se lee el queryParams 'input' el cual contiene la pregunta del usuario.

En caso de no existir texto se devuelve un json con un mensaje de error y codigo 400.

Si existe texto dentro de input, se pasa el mismo al metodo pre-existente `generate_response()` el cual es importado de chatbot.py.

Se adjunta la respuesta del chatbot junto con la pregunta original en un JSON y se responde al http request.

```
@app.route('/chatbot', methods=['GET'])
def chatbot_request():
    text = request.args.get('input')
    if text is None:
        return jsonify({"error": "No input provided"}), 400

    resultado = generate_response(text)
    return jsonify({"request": text, "response": resultado})
```

Se configuro la ruta de acceso para el chatbot en '/upload' y metodo POST. Recibe un archivo con nombre 'image'. En caso de no existir texto se devuelve un json con un mensaje de error y codigo 400.

El mismo se pasa al metodo `decode_and_verify()` el cual decodifica la imagen y la convierte en una matriz. Se compara la misma con el resto de las huellas cargadas en el sistema. En caso de que coincidan devuelve el usuario correspondiente. En el caso contrario devuelve un string vacio.

Luego se arma un objeto con el nombre de usuario (si lo tiene) y un boolean que 'authenticated' que representa si el usuario pudo iniciar sesion exitosamente o no.

```
@app.route('/upload', methods=['POST'])
def upload_file():
    if 'image' not in request.files:
        return jsonify({'error': 'No image part in the request'}), 400

    image_file = request.files['image']

    # If the user does not select a file, the browser submits an empty file without a
filename
    if image_file.filename == '':
        return jsonify({'error': 'No selected file'}), 400


    user = decode_and_verify(image_file)
```

```
    response = {}

    if user:
        response = {
            'authenticated': True,
            'user': user
        }
    else:
        response = {
            'authenticated': False,
            'user': 'none'
        }

    return jsonify(response), 200;
```

CORS

Se instalo Flask-Cors usando `pip install -U flask-cors`

El CORS (Cross-origin resource sharing) permite configurar a que paginas web (front-end) se les da acceso a los endpoint de un servicio web (back-end). En nuestro caso decidimos dejarlo sin restricciones, por lo cual cualquier pagina podria consumir nuestros endpoint.

```
CORS(app)
```

Vercel

Se creo una cuenta en https://vercel.com para generar un deploy automatizado, dandole acceso al servidor de github donde se encuentra nuestra aplicación web. El mismo descarga la aplicacion, compila los html, js y css correspondientes y genera un deploy como un subdominio de vercel. De esta manera queda disponible el servicio web en la siguiente URL: https://ppi-front.vercel.app/

Repositorio del front-end. https://github.com/Francuster/PPI-front

Documentacion Funcional

Chatbot

Carga de datos

El chatbot carga las preguntas y respuestas de un archivo CSV utilizando la función load_data_from_csv. Las preguntas se almacenan en una lista llamada questions y las respuestas en una lista llamada answers.

Vectorización de datos

Utiliza la clase TfidfVectorizer de scikit-learn para vectorizar las preguntas. Esto convierte las preguntas de texto en representaciones numéricas que pueden ser procesadas por el modelo.

Generación de respuestas

La función generate_response toma la entrada del usuario, la vectoriza y encuentra la pregunta más cercana utilizando el modelo de vecinos más cercanos entrenado. Luego, devuelve la respuesta correspondiente a esa pregunta.

Interfaz del chatbot

El chatbot proporciona una interfaz interactiva donde espera la entrada del usuario por consola. Responde a las entradas del usuario utilizando la función generate_response hasta que el usuario escribe "quit", momento en el cual el chatbot se despide y termina.

Ejecución del chatbot

La ejecución principal se realiza verificando si el script se ejecuta directamente (if *name* == "*main*":) y llama a la función chatbot para iniciar la interacción con el usuario.

Login Huella Digital

Definición de la base de datos de huellas digitales

Al inicio del código, se define un diccionario llamado database que contiene imágenes de huellas digitales asociadas a nombres de usuario. Estas imágenes se cargan desde archivos usando la biblioteca OpenCV (cv2).

Comparación de huellas digitales

Se define una función llamada `compare_fingerprints(fingerprint1, fingerprint2)` que calcula la diferencia absoluta entre dos huellas digitales. Esto se hace usando la función `cv2.absdiff()`, que encuentra la diferencia píxel a píxel entre las dos imágenes y devuelve una nueva imagen que contiene estas diferencias. Luego, se suma el valor absoluto de todos los elementos en esta nueva imagen, proporcionando la diferencia total entre las huellas digitales.

Autenticación de huellas digitales

Se define una función llamada `authenticate_fingerprint(input_fingerprint)` que autentica una huella digital de entrada comparándola con las huellas digitales almacenadas en la base de datos. Itera sobre cada huella digital en la base de datos, calcula la diferencia utilizando la función `compare_fingerprints()` y devuelve el nombre de usuario asociado a la huella digital almacenada con la menor diferencia absoluta.

Función de autenticación

La función `auth(input)` toma la ruta de una imagen de huella digital como entrada, carga la imagen utilizando OpenCV y luego llama a `authenticate_fingerprint()` para autenticar la huella digital. Dependiendo del resultado de la autenticación, imprime un mensaje indicando si la autenticación fue exitosa o no.

Interacción con el usuario

La función `login()` solicita al usuario que ingrese la ruta de la imagen de la huella digital. Itera este proceso hasta que el usuario escriba "quit".

Ejecución del programa

Al final del script, se verifica si el script se está ejecutando directamente (`name == "main"`) y, si es así, se llama a la función `login()` para iniciar la interacción con el usuario.

Chatbot como servicio

El código del chatbot lo alojamos en PythonAnywhere por que ofrece un servicio de hosting gratuito de 3 meses y es de sencilla utilización. Dentro del proyecto web creado en PythonAnywhere, armamos una aplicación Flask que proporciona dos endpoints:

El endpoint del chatbot

"`/chatbot`" espera recibir una solicitud GET con un parámetro `input` que contiene el texto con el que se quiere interactuar con el chatbot. Luego, llama a la función `generate_response` del módulo `chatbot` para generar una respuesta basada en el texto proporcionado y devuelve esta respuesta como JSON. A continuación se adjunta una request de ejemplo y su respectiva respuesta respuesta:

Request: https://matizipi.pythonanywhere.com/chatbot?input=What%20cases?

Respuesta: { "request":"What cases?", "response":"Chatbot use cases include virtual assistants, customer support, sales assistance, language learning, and entertainment." }

El endpoint del lector de huellas

"`/upload`" recibe una solicitud POST con un archivo de imagen adjunto llamado `image`. Luego, utiliza la función `decode_and_verify` del módulo `loginHuella` para decodificar y verificar la imagen. Dependiendo del resultado de la verificación, devuelve un JSON indicando si el usuario está autenticado o no, junto con alguna información adicional, como el nombre de usuario. A continuación se adjunta una request de ejemplo y su respectiva respuesta respuesta:

Request: https://matizipi.pythonanywhere.com/upload

Respuesta: { "authenticated": true, "user": "usuario1" } Además, el código utiliza la extensión Flask-CORS para permitir el acceso a estos endpoints desde dominios diferentes al del servidor en el que se ejecuta la aplicación Flask. Esto nos permitió consumir estos endpoints desde otra aplicación que utilizamos como interfaz de usuario.

Referencias

- https://www.innovatrics.com/glossary/digital-onboarding/
- https://www.apd.es/algoritmos-del-machine-learning/#:~:text=Una%20vez%20entendido%20qu%C3%A9%20es,no%20supervisado%20y%20por%20refuerzo
- https://blog.cliengo.com/chatbots-opciones/
- Repositorio donde se estará volcando el trabajo realizado: https://github.com/Francuster/PPI_IA
- Repositorio de front-end en Angular https://github.com/Francuster/PPI-front
- ChatGPT