Tecnológico de Monterrey

Campus Monterrey

Inteligencia artificial avanzada para la ciencia de datos I (Gpo 101)

Reporte de “Reto Datos”

Juan Pablo Castañeda Serrano	A01752030
Aldo Daniel Villaseñor Fierro	A01637907
Francisco Castorena Salazar	A00827756
José Alfredo García Rodríguez	A00830952

Link al repositorio: https://github.com/a01752030/PortafolioRetoFC

1. Herramientas tecnológicas

Front-end: React

React utiliza un DOM virtual, que optimiza la actualización y renderizado de componentes, haciendo que la aplicación sea rápida y eficiente.

Ventajas:

- **Componentización:** React se basa en componentes, permitiéndonos reutilizar código y facilitando la mantenibilidad.
- **Activo Comunitario:** Una vasta comunidad apoya a React, lo que significa una gran cantidad de bibliotecas, tutoriales y soporte disponible.
- **Flexibilidad:** React se integra fácilmente con diversas bibliotecas y frameworks back-end, como Flask en este caso.

Back-end: Flask

Es ligero, fácil de usar y extensible.

Ventajas:

- **Sencillez:** Flask es fácil de configurar y empezar a trabajar.
- **Extensible:** Aunque es un microframework, se puede ampliar con varias extensiones para agregar funcionalidades adicionales.
- **Python:** Al ser basado en Python, podemos utilizar diversas bibliotecas de reconocimiento facial y procesamiento de datos disponibles para este lenguaje, lo que facilita la implementación del núcleo de nuestra aplicación.

Base de datos: MongoDB

En lugar de usar tablas y filas como en las bases de datos relacionales, MongoDB se basa en colecciones y documentos, lo que puede ser más adecuado para ciertos tipos de aplicaciones.

Ventajas:

- **Esquemas flexibles:** Puede adaptarse fácilmente a cambios en la estructura de datos sin necesidad de migraciones complicadas.
- **Escalable:** Es fácil de escalar horizontalmente.

- **Rápido:** Al ser orientado a documentos y permitir búsquedas indexadas, suele ofrecer tiempos de respuesta rápidos para ciertos tipos de consultas.

Infraestructura: EC2 (Amazon Elastic Compute Cloud)

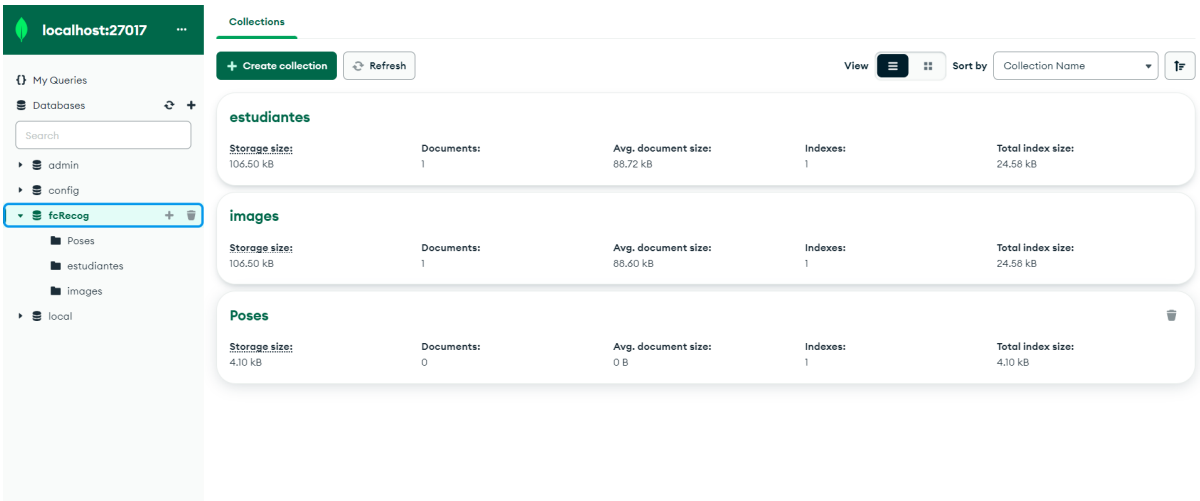
Amazon EC2 es un servicio que ofrece capacidad de computación en la nube. Nos permite ejecutar servidores y escalar recursos según las necesidades de nuestra aplicación.

Ventajas:

- **Escalabilidad:** Podemos iniciar o detener instancias según la demanda, pagando solo por lo que usamos.
- **Integración:** EC2 se integra con otros servicios de AWS, lo que nos facilita implementar soluciones más avanzadas, como bases de datos gestionadas, almacenamiento en la nube, entre otros.
- **Seguridad:** EC2 ofrece robustas características de seguridad, incluidos grupos de seguridad y redes VPC.

En resumen, la combinación de estas tecnologías ofrece un balance entre rendimiento, escalabilidad, seguridad y facilidad de desarrollo. Al trabajar juntas, estas herramientas nos permiten crear una solución robusta y eficiente para la toma de asistencia por reconocimiento facial.

2. Modelo de almacenamiento



Nuestro modelo de almacenamiento consiste en 3 colecciones, la primera de estas es la de estudiantes:

```
student = {
    'nombre_del_alumno': nombre_del_alumno,
    'matricula': matricula,
    'fecha_de_registro': fecha_de_registro,
    'clase': clase,
    'image': image_bytes,
    'asistencias': 0
}
```

Esta consiste en los anteriores campos, ya que utilizando mongodb, tenemos la ventaja de poder declarar como estos campos se comportan de manera dinámica. Por ejemplo en la imagen anterior, podemos observar que guardamos la imagen en forma de “image_bytes”, pero una vez que implementemos más seguridad este tipo de dato puede ser cambiado con facilidad desde el backend.

La segunda colección, es la de imagen. Esta es utilizada para guardar los bytes de las imágenes y para ser correlacionada con el estudiante correspondiente. La razón por la que tenemos una colección diferente donde guardamos la imagen se debe a que decidimos tener un reemplazo de imagen original en la colección de estudiante en caso que alguna de las imágenes se corrompa en la manipulación de sus datos, ya que estamos probando con diferentes algoritmos y formas de encriptación, queremos tener un fallback y una forma principal de manipulación.

```
_id: ObjectId('65241c907733b0b0309c98d8')
image: Binary.createFromBase64('/9j/4AAQSkZJRgABAQAAAQABAAD/2wBDAAMCAgICAgMCAgIDAwMDBAYEBAQEBAgGBgUGCQgKCgkICQkKDA8MCgsOCwkJDRENDg8Q…', 0)
```

Nuestra última colección es donde almacenamos la información de las poses, específicamente el guardado de las participaciones e información relacionada.

---

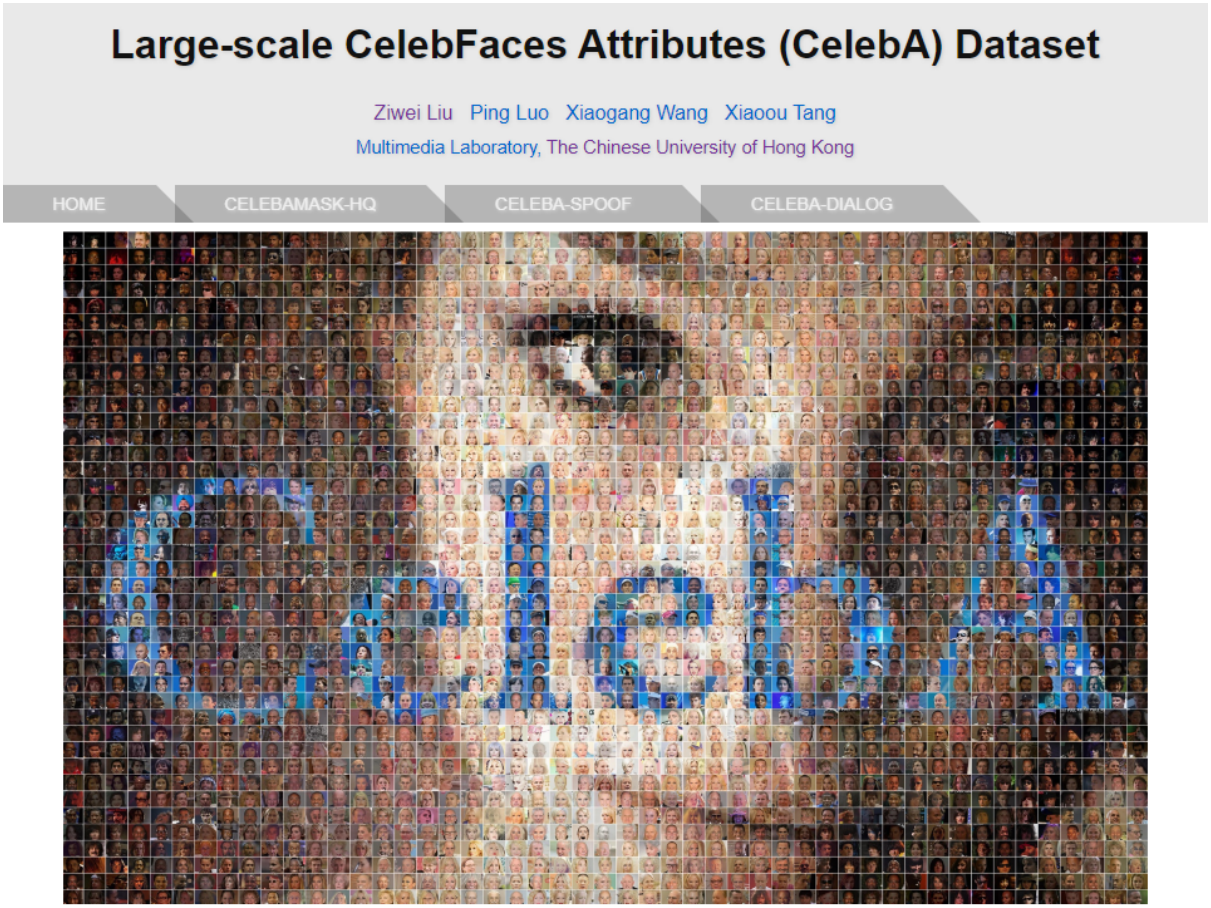
1. Extraiga el conjunto de datos, limpienlo y cárgalo a la plataforma seleccionada donde será utilizado.
2. Suban los scripts de configuración utilizados y una muestra pequeña de los datos cargados a su repositorio de github para poder verificar que el cargado funciona correctamente.
3. Separen sus datos en conjuntos de prueba y de entrenamiento utilizando el esquema *k-fold cross validation*.

---

3. Extracción, limpieza y carga de datos

Los datos que vamos a utilizar es la base de datos CelebFaces Attributes Dataset (CelebA) que es un conjunto de datos de atributos faciales a gran escala con más de 200.000 imágenes

de celebridades. Las imágenes de este conjunto de datos cubren grandes variaciones de pose y desorden de fondo.



Las imágenes ya están pretratadas por lo que no es necesario limpiar los datos y ha sido ampliamente utilizada en el área de investigación en visión computacional, lo que sí definimos fue utilizar la versión de imagenes ‘in the wild’ porque esto es lo que más se aproxima en nuestro problema puesto que también tenemos que desarrollar una solución para recortar y alinear las imágenes de los alumnos en el salón de clases.

Para cargar los datos podemos correr el siguiente script que enviará todos nuestros .jpg que tenemos a nuestra base de datos en MongoDB:

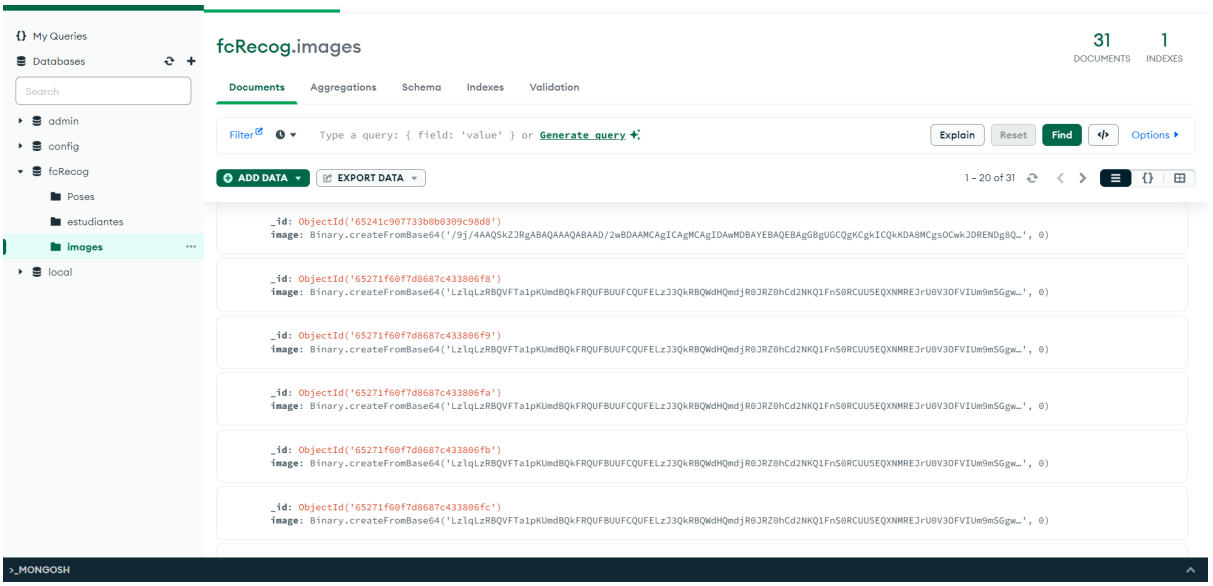
```
async function uploadToServer(imageData) {
    try {
        const studentData = {
            image: imageData,
            nombre_del_alumno: nombreDelAlumno,
            matricula: matricula,
            fecha_de_registro: fechaDeRegistro,
            clase: clase
        };

        const response = await fetch('http://localhost:5000/upload', {
            method: 'POST',
            headers: { 'Content-Type': 'application/json' },
            body: JSON.stringify(studentData)
        });

        const result = await response.json();
        console.log(result.message);
    } catch (error) {
        console.error('Error uploading data:', error);
    }
}
```

4. *Scripts de configuración y datos cargados*

Para subir imágenes a nuestra base de datos en MongoDB utilizamos el siguiente código y a continuación podemos ver cómo es que se ven unas imágenes de prueba subidas a la base de datos. De igual manera cualquier imagen que estén en la carpeta de “images” en el backend pueden ser agregadas solo mandando a llamar la ruta post “/save_all_images” después de ponerlas las imágenes en la carpeta correspondiente:

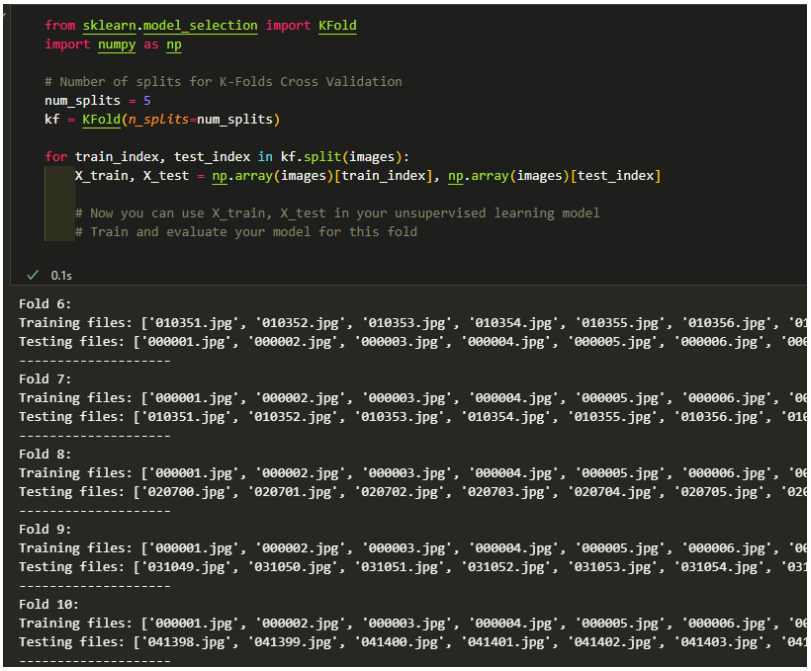


5. *Creación de conjunto de datos Train y Test*

En este análisis, utilizamos técnicas avanzadas de validación cruzada para crear conjuntos de datos de entrenamiento y prueba. La validación cruzada es una metodología crucial en el aprendizaje automático que nos permite evaluar el rendimiento del modelo de manera más precisa. Para llevar a cabo esta técnica, usamos la función KFold de la biblioteca scikit-learn (sklearn.model_selection) para dividir nuestros datos en varios partes o "folds". Cada fold actúa como un conjunto de prueba en un ciclo, mientras que los demás se utilizan para el entrenamiento del modelo.

Metodología:

Utilizamos la función KFold con un valor de 5 para el número de folds. Esto implica que nuestros datos se dividieron en cinco partes iguales. A continuación, presentamos una representación visual de cómo se realizaron las divisiones:



```python
from sklearn.model_selection import KFold
import numpy as np

# Number of splits for K-Folds Cross Validation
num_splits = 5
kf = KFold(n_splits=num_splits)

for train_index, test_index in kf.split(images):
    X_train, X_test = np.array(images)[train_index], np.array(images)[test_index]

    # Now you can use X_train, X_test in your unsupervised learning model
    # Train and evaluate your model for this fold
```

✓ 0.1s

```
Fold 6:
Training files: ['010351.jpg', '010352.jpg', '010353.jpg', '010354.jpg', '010355.jpg', '010356.jpg', '01
Testing files: ['000001.jpg', '000002.jpg', '000003.jpg', '000004.jpg', '000005.jpg', '000006.jpg', '00
--------------------
Fold 7:
Training files: ['000001.jpg', '000002.jpg', '000003.jpg', '000004.jpg', '000005.jpg', '000006.jpg', '00
Testing files: ['010351.jpg', '010352.jpg', '010353.jpg', '010354.jpg', '010355.jpg', '010356.jpg', '01
--------------------
Fold 8:
Training files: ['000001.jpg', '000002.jpg', '000003.jpg', '000004.jpg', '000005.jpg', '000006.jpg', '00
Testing files: ['020700.jpg', '020701.jpg', '020702.jpg', '020703.jpg', '020704.jpg', '020705.jpg', '02
--------------------
Fold 9:
Training files: ['000001.jpg', '000002.jpg', '000003.jpg', '000004.jpg', '000005.jpg', '000006.jpg', '00
Testing files: ['031049.jpg', '031050.jpg', '031051.jpg', '031052.jpg', '031053.jpg', '031054.jpg', '03
--------------------
Fold 10:
Training files: ['000001.jpg', '000002.jpg', '000003.jpg', '000004.jpg', '000005.jpg', '000006.jpg', '00
Testing files: ['041398.jpg', '041399.jpg', '041400.jpg', '041401.jpg', '041402.jpg', '041403.jpg', '041
--------------------
```

Observamos que los archivos se separaron en cada conjunto (fold) de acuerdo con sus nombres correspondientes. Cada fold representa una porción específica de los datos que se utiliza alternativamente para evaluar el modelo mientras se entrena en los otros folds. Este enfoque garantiza una evaluación exhaustiva del modelo en diferentes subconjuntos de datos, lo que mejora la confiabilidad de nuestras conclusiones y decisiones.

6. Determinación de Big Data

Para determinar esto consideramos los siguientes factores:

1. **Volumen de datos**: Debido a que estamos utilizando los bytes de las imágenes quizá se pueda pensar que necesitemos big data, sin embargo concurrimos que el volumen de estas no llegaría más allá de los 200mb, basándonos en la población de un salón de clases y en la manera que estamos transmitiendo las imágenes a través de nuestra API no debería aumentar en manera significativa
2. **Velocidad de datos**: De momento, la generación de datos es bastante eficaz, de nuevo, al guardar las imágenes en base64 estas son transferidas de manera inmediata entre nuestro frontend y base de datos-
3. **Variedad de datos:** Quizá este punto sea el que más nos dio a considerar un modelo de BigData, al transformar los datos de las imágenes estamos manejando diferentes tipos de datos, sin embargo este sería el único caso donde tenemos diferentes tipos de datos, y todavía podríamos hegemonizar estos en nuestro backend, así que de nuevo no lo vemos necesario.
4. **Complejidad:** Al poder hegemonizar los datos en el frontend y el backend, y así como tener control completo de que es lo que se sube a la base de datos, vemos imposible la complejidad innecesaria de datos.

Tras analizar diversos factores críticos para determinar la necesidad de un enfoque de Big Data, llegamos a la conclusión de que, en nuestro contexto, no es necesario adoptar dicho enfoque. A pesar de que manejamos imágenes que potencialmente pueden ocupar grandes volúmenes de almacenamiento, hemos constatado que el volumen total no excede los 200mb, adecuándose a la población de un salón de clases y a la eficiencia de nuestra API. Si bien la variedad de datos, particularmente debido a la transformación de imágenes, podría haber inclinado la balanza hacia un modelo de Big Data, nuestra capacidad para uniformizar estos datos en el frontend y backend minimiza esta necesidad. Además, la velocidad con la que se generan y transmiten los datos es eficiente, y la complejidad de los mismos es manejable, ya que mantenemos un control estricto sobre lo que se envía y almacena en la base de datos. Por lo tanto, con base en estos análisis, determinamos que un enfoque tradicional es adecuado y más rentable para nuestra aplicación.