



PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE

Facultad de Ingeniería

IIC2613 Inteligencia Artificial

2025-2

Tarea 4

Aprendizaje de Máquina

Fecha de entrega: Jueves 6 de noviembre a las 23:59 hrs

Aspectos generales

Formato y plazo de entrega

El formato de entrega es un archivo con extensión .pdf para la parte 1, y dos archivos con extensión .ipynb para las partes 2 y 3. En estos deben estar tanto las respuestas de código como teóricas. El lugar de entrega es en el repositorio de la tarea, en la branch por defecto, hasta el jueves 6 de noviembre a las 23:59 hrs. Para crear tu repositorio, debes entrar en el enlace del anuncio de la tarea en Canvas. Por último, recuerda que los cupones de atraso son días **reales** (hábiles o no) extra.

Integridad Académica

Este curso se adhiere al Código de Honor establecido por la universidad, el cual tienes el deber de conocer como estudiante. Todo el trabajo hecho en esta tarea debe ser **totalmente individual**. La idea es que te des el tiempo de aprender estos conceptos fundamentales, tanto para el curso, como para tu formación profesional. Las dudas se deben hacer exclusivamente al cuerpo docente a través de las [issues en GitHub](#).

Por otra parte, sabemos que estás utilizando material hecho por otras personas, por lo que es importante reconocerlo de la forma apropiada. Todo lo que obtengas de internet debes citarlo de forma correcta (ya sea en APA, ICONTEC o IEEE). Cualquier falta a la ética y/o a la integridad académica será sancionada con la reprobación del curso y los antecedentes serán entregados a la Dirección de Pregrado.

Comentarios adicionales

El objetivo de esta tarea es que puedan utilizar distintos algoritmos para resolver problemas de clasificación y regresión, aplicándolos en problemas donde pueden ser de gran utilidad. Es fundamental que pongan énfasis en las justificaciones de sus respuestas, cuidando la redacción, ortografía; manteniendo el código ordenado y comentado. Aquellas respuestas que solo presenten resultados o código (sin contexto ni comentarios) no serán consideradas, mientras que tareas desordenadas pueden ser objeto de descuentos.

Recursos para realizar la tarea

Pandas

Esta librería es especialmente útil para manejar conjuntos de datos en forma de tablas. Es una extensión de **NumPy**, por lo que los cálculos matemáticos realizados usando las clases de esta librería son muy eficientes, aunque sus mayores atributos están en la forma en la que se indexan los datos. La clase **DataFrame** permite hacer un manejo muy eficiente de estos.

En general, se denominará **DataFrame** a la tabla de datos cuando estos ya estén cargados, y **Series** a las columnas. Para importar la librería, por convención, utilizamos el diminutivo **pd**:

```
1 import pandas as pd
```

Por convención, un **DataFrame** se denomina **df**. Si queremos tener una idea de lo que contiene, podemos usar el método **head**, que despliega los primeros cinco elementos (filas) del dataset, lo que es bastante rápido y ordenado.

Por otro lado, podemos conocer el número de filas y columnas de un **DataFrame** con el atributo **shape**. También, pueden resultar útiles los métodos **info** y **describe**. El primero devuelve características (número de columnas, cuáles son y número de elementos no nulos), y el segundo proporciona información básica estadística del conjunto de datos.

Por otro lado, si queremos limpiar el **DataFrame** de datos indeseados, deberemos conocerlos primero. Para ello, se puede utilizar el método **isna**. Los elementos nulos aparecen con el denominativo **NaN**, y el método **isna** devolverá un booleano con un valor correspondiente a si se encuentra dicho denominativo o no. De la misma manera, el método **duplicated** nos ayudará a encontrar duplicados.

Navegar por los datos es relativamente intuitivo. Una columna o serie puede ser accedida como si fuera un atributo de valor **name**, por ejemplo, **df.name**, o como si fuera un diccionario con una llave: **df[“name”]**. Para acceder a dos series se utiliza la extensión **df[[“name1”, “name2”]]**. Una lista de todas las series disponibles se obtiene con **df.columns**. Finalmente, para acceder a un elemento dentro de una columna se debe utilizar su índice luego de haber llamado a la serie. Por ejemplo, al elemento **354** de **name** se accede haciendo **df[“name”][354]**.

Una vez hemos podido acceder a ciertas series, o a elementos de ellas, es posible trabajar de manera numérica. Algunas operaciones matemáticas son bastante simples y directas, mientras que otras pueden requerir un poco más de investigación en la documentación de la librería. Por ejemplo:

```
1 # Esta es una operacion simple sobre dos celdas del DataFrame
2 q = df[ name1 ][354] + df[ name2 ][726]
3
4 # Estas son operaciones sobre toda la serie
5 m = df[ name3 ].mean()
6 s = df[ name4 ].sum()
```

Una operación numérica interesante sobre la serie es la de **value_counts**, que retornará la cantidad de elementos de cada clase de una serie. En particular, las clases de esa serie se pueden recuperar con el método **unique**.

Existen accesos más avanzados que se pueden invocar con el método **loc** o el método **groupby**. Estos métodos permiten trabajar con grandes cantidades de datos, hacer consultas y agrupaciones. El método **loc** permite filtrar filas o columnas mediante una etiqueta o un booleano. Un ejemplo puede ser:

```
1 df.loc[df[ name1 ] > 6, [ name2 ]]
```

En este ejemplo se filtrarán todas las filas cuya serie de nombre **name1** tenga un valor **mayor que 6**, pero se recuperarán los valores de la serie **name2** de esas filas.

Por otro lado, el método **groupby** retornará un objeto con varias propiedades interesantes. Típicamente se utiliza sobre valores no numéricos a los cuales se les pueden aplicar otras operaciones o métodos:

```
1 df.groupby([ name1 ]).sum()
```

Si la serie **name1** tiene **5 instancias**, el método mostrará una tabla de **5 filas** para las cuales intentará sumar sus valores en las otras series.

Finalmente, para la manipulación de archivos, existen dos tipos de métodos principales. Si lo que se desea es guardar un DataFrame procesado en un archivo **.csv**, se puede usar el método **to_csv**:

```
1 df.to_csv( archivo_de_destino.csv )
```

Por otro lado, si se quiere cargar un DataFrame desde un archivo **.csv**, se utiliza el método **read_csv**:

```
1 df.read_csv( archivo_por_cargar.csv )
```

Notar que existen métodos similares para otras extensiones de archivo.

Scikit-learn

Adicionalmente, usaremos la librera scikit-learn, la cual provee una gran gama de técnicas de aprendizaje de máquina, tal como clasificadores del tipo **SVM**, árboles de decisión, entre otros.

Es una librería orientada a objetos, y también importaremos algunas funciones útiles. Primero que todo, para esta oportunidad nos interesan los clasificadores, que podemos importar haciendo:

```
1 # Clase que utilizaremos para un clasificador SVM
2 from sklearn.svm import SVC
3
4 # Clase que utilizaremos para un clasificador Bayesiano ingenuo
5 from sklearn.naive_bayes import GaussianNB
6
7 # Clase que utilizaremos para un clasificador de arbol de decision
8 from sklearn.tree import DecisionTreeClassifier
```

A la hora de trabajar con estas clases se debe tener en cuenta lo siguiente:

- Todos los clasificadores pueden ajustar sus hiperparámetros con el método **set_params**.
- El método **fit(X, y)** ejecutará el procedimiento de optimización de parámetros o entrenamiento del clasificador.
- El método **predict(X)** generará un vector **y** con la predicción del clasificador para la matriz **X**.

Google Colab

Si queremos usar estos algoritmos de manera más rápida, podemos usar las famosas GPUs (tarjetas gráficas). Estas tarjetas son procesadores diseñados para realizar tareas gráficas, pero también han demostrado una gran utilidad en el campo del aprendizaje automático por su alto poder de procesamiento en paralelo. Google ha dispuesto una herramienta gratuita para poder aprovechar su uso. Esta corresponde a Google Colab, por lo que recomendamos fuertemente desarrollar la tarea en esta plataforma, que además permite exportar el archivo como un archivo **.ipynb**.

IMPORTANTE: Para poder activar el uso de GPU en Google Colab, deben ir a *Editar → Configuración del notebook → Acelerador de hardware → GPU*.

Open Computer Vision (OpenCV)

La librería de Python **OpenCV (Open Computer Vision)** es una de las más utilizadas para el procesamiento y análisis de imágenes y video.

En esta tarea, el uso de OpenCV está limitado principalmente a partes automatizadas del código base de la sección DCCachipun, salvo en el *bonus*, donde podrás reutilizar fragmentos del código encargado de generar los datasets en formato .csv.

En esta tarea se emplearan las siguientes funciones:

- **Lectura de imágenes:** convierte una imagen almacenada en disco en un arreglo NumPy manipulable por Python.

```
1 image = cv2.imread( nombre_imagen.jpg )
```

- **Conversión de espacio de color:** OpenCV carga las imágenes en formato **BGR**, mientras que librerías como `matplotlib` y `mediapipe` trabajan en formato **RGB**. Por tanto, es necesario realizar la conversión:

```
1 image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
```

- **Copia y manipulación de imágenes:** en algunos casos es necesario duplicar la imagen original para dibujar sobre ella sin modificar los datos base:

```
1 image_copy = image.copy()
```

- **Dibujo y visualización (opcional):** OpenCV permite dibujar sobre imágenes utilizando funciones como:

```
1 cv2.line(image, pt1, pt2, color, thickness)
2 cv2.circle(image, center, radius, color, thickness)
3 cv2.putText(image, text, org, font, fontScale, color, thickness)
```

1. Reflexión (1.5 puntos)

Esta sección de la tarea consiste en una serie de preguntas de reflexión que tendrás que responder basándote en el podcast de la profesora Jocelyn Dunstan "Hablemos de Datos"¹ que está disponible en todas las plataformas de escucha (incluida Spotify). Escucha uno o más capítulos del podcast y escoge un episodio para contestar las siguientes preguntas:



- 1.1. (0.3 ptos.) ¿Por qué escogiste este episodio?
- 1.2. (0.6 ptos.) ¿Cuál crees que es la relevancia del tema tratado en el capítulo dentro del campo de la inteligencia artificial, *machine learning* o ciencia de datos?
- 1.3. (0.6 ptos.) Busca al menos una referencia externa que enriquezca alguna idea mencionada en el capítulo y discute.

¿Qué entregar en esta parte?

En un subdirectorio **reflexion/**:

- Un archivo **respuesta.pdf** con las respuestas a las preguntas.

¹<https://open.spotify.com/show/3QSijQ6hGVYBBv3Rlq5J9O?si=7a89af32ec3148f2>

2. DCCachipun (1.75 puntos)



2.1. Introducción

21 de octubre de 2025. Final mundial de cachipun. Eres tu contra Björn, el campeón sueco. Vas 2-1 al mejor de tres. Solo necesitas ganar una vez más. Siguiente jugada, sacas piedra y Björn saca papel. El marcador queda 2-2. Comienzas a pensar en tu siguiente jugada: “Si el sacó papel antes, entonces va a sacar tijera, eso significa que debo sacar piedra. Pero si yo saco piedra, el sacará papel, ...” y sigues así sin poder elegir. Empiezas a sudar como loco/a y debes decidir pronto. No sabes qué hacer y de pronto todo se vuelve negro. Suena tu alarma y te despiertas en tu cama. Te das cuenta que todo fue un sueño y que debes arreglarte para ir a la clase de Inteligencia Artificial, pero no puedes sacarte la final soñada de la cabeza. Decides convertir este sueño en realidad y ser el próximo campeón mundial de cachipun, para lo que vas a necesitar mucho entrenamiento. Te quedas pensando en esto, pero no sabes cómo hacerlo. Camino a la universidad, escuchando tu podcast favorito de IA, aprendes sobre visión por computador y comentan de un ejemplo sobre la clasificación de gestos de la mano, lo que despierta en ti una idea, tal vez la más revolucionaria. Decides que harás un clasificador de gestos de manos para obtener un programa que te permita entrenar tus habilidades en el cachipun!

2.1.1. Objetivo general

El objetivo general de esta sección de la tarea es desarrollar modelos de aprendizaje automático capaces de reconocer gestos de la mano —específicamente los gestos de *piedra*, *papel* y *tijera*— a partir de imágenes estáticas de manos.

Para lograrlo, se utilizarán los *landmarks* (o coordenadas) resultantes de la librería MediaPipe², los cuales describen la posición tridimensional de 21 puntos característicos de la mano. A partir de estos datos, se entrenarán y evaluarán distintos modelos de clasificación supervisada con el fin de determinar cuál logra un mejor desempeño en la tarea de reconocimiento de gestos. Algunas de las funciones de esta librería **requieren del uso de una GPU**, por lo que te recomendamos revisar la sección de **Google Colab**.

Además, se espera que seas capaz de comprender el flujo completo de trabajo de un sistema de visión por computador basado en *landmarks*, desde la lectura, procesamiento de datos y elección de características, hasta la selección, entrenamiento y evaluación de modelos predictivos.

²<https://ai.google.dev/edge/mediapipe/solutions/guide>

2.1.2. Dataset

El dataset utilizado en esta tarea corresponde a *Rock, Paper, Scissors Dataset*³ disponible en la plataforma *Kaggle*⁴.

Para desarrollar correctamente esta actividad, se te entregará el archivo `hands.zip` (disponible en la sección archivos del curso en Canvas), que contiene el conjunto de prueba y entrenamiento de las imágenes de las manos, junto con un archivo base `.ipynb` (Jupyter notebook) que contiene el código que procesa las imágenes RGB del *dataset* de *Kaggle* a formato CSV, generando así los archivos `test_landmarks.csv`, `train_landmarks.csv`, `val_landmarks.csv`.

Cada entrada de los *datasets* contiene los siguientes datos:

- **Filename:** Corresponde al nombre de la imagen en su carpeta correspondiente de `test`, `train` o `val`.
- **Label:** Corresponde a la categoría de la foto. Puede ser *rock*, *paper* o *scissors*.
- $x_i, \forall i \in \{0, 1, \dots, 62\}$: corresponden a los *landmarks* que genera la librería mediapipe para la imagen de la mano respectiva. Puesto que la mano contiene 21 *landmarks*, y cada *landmark* tiene 3 coordenadas (X,Y,Z), se obtienen $21 \cdot 3 = 63$ puntos por imagen.

2.2. Actividades

2.2.1. *Landmarks* para las manos (0.25 puntos)

Un *landmark* o punto de referencia es un punto de interés clave y localizable en un objeto o imagen. En el contexto de inteligencia artificial, son puntos específicos que ayudan a los algoritmos a comprender la estructura, la pose y las características de un objeto. Para poder generarlos usamos la librería MediaPipe, la cual genera 21 puntos que describen la anatomía de la mano, que tienen coordenadas propias en el espacio (X,Y,Z). Estas coordenadas se encuentran normalizadas respecto al tamaño de la imagen en que se generan, y nos permiten posicionar en la imagen, como si fuese un plano cartesiano, las coordenadas. Por otra parte, podemos pensar en la mano como un grafo no dirigido $G = (V, E)$, donde:

- Los nodos V corresponden a los *landmarks* (los 21 puntos de la mano).
- Las aristas E corresponden a las tuplas definidas en `hand_connections`, que indican qué puntos están conectados.

Formalmente:

$$V = \{0, 1, 2, \dots, 20\}$$

$$E = \{(0, 1), (1, 2), (2, 3), \dots, (19, 20), (0, 17)\}$$

esquemáticamente:

³<https://www.kaggle.com/datasets/glushko/rock-paper-scissors-dataset?resource=download>

⁴<https://www.kaggle.com/>

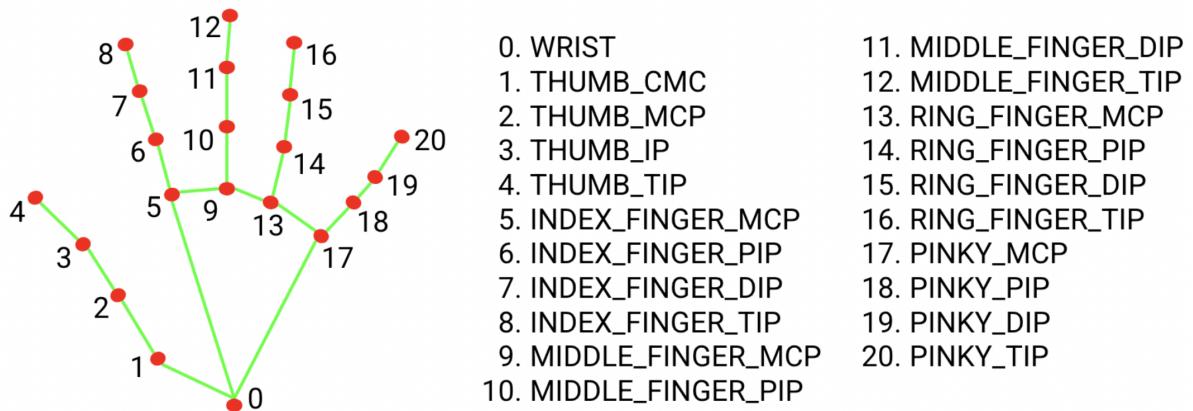


Figura 1: Modelado general de la mano

Para esta parte de la tarea, se entregará la función `imprimir_landmarks` que deberás completar. Esta función permitirá imprimir en pantalla una imagen del conjunto de entrenamiento junto con sus respectivos *landmarks* posicionados sobre la mano, mostrando los números de los landmarks así como las conexiones. Un ejemplo de *output* sería el siguiente:

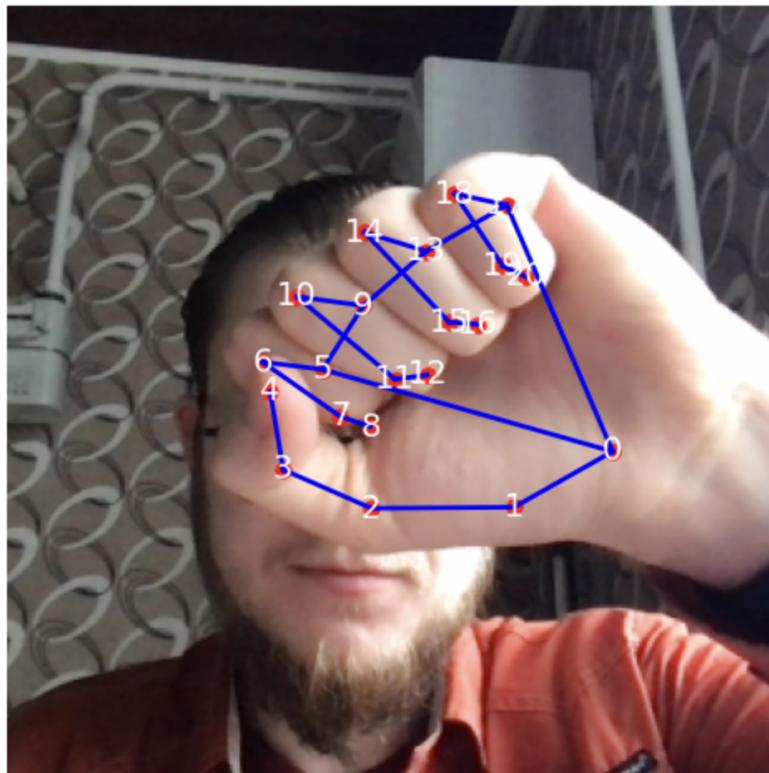


Figura 2: Output con landmarks de la imagen glu_235.png ubicada en la carpeta rock de entrenamiento.

La información completa de como armar las conexiones podrás encontrarla aquí.

2.2.2. Creación de características (0.75 puntos)

A partir de los *landmarks*, fabrica al menos 7 *features* para poder entrenar un modelo de *machine learning*. Para cada una de ellas, explica que hace o a que corresponde, justifica su uso y explica por qué crees que será útil para obtener un buen desempeño en la clasificación de los gestos de las manos. (**Nota: Esta**

prohibido usar los landmarks como *features*. Debes crear tus propias características en base a ellos. En caso contrario, no será corregida esta parte de la tarea).

Luego, elige dos de ellas y realiza un análisis estadístico. ¿Cómo se ve su distribución en el conjunto de datos? ¿Consideras que es una buena característica para separar los datos?

2.2.3. Entrenamiento y evaluación de un modelo (0.75 puntos)

Ahora utilizaremos los *dataset* de entrenamiento, validación y entrenamiento generados previamente para encontrar el mejor modelo de detección de las manos.

Con los datos de entrenamiento, entrena los siguientes algoritmos: *SVM*, *Random Forest* y *Decision Tree*. Para cada uno de ellos deberás escoger 3 sets de hiperparámetros. Luego, utilizando el conjunto de validación evalúa estos modelos con la métrica de *accuracy* y genera su matriz. A continuación selecciona los tres mejores modelos y evalúalos en el set de prueba. Luego comenta sobre los resultados obtenidos y su matriz de confusión. Por último, comenta qué características del modelo que obtuvo el mejor resultado, pueden haberlo favorecido frente a los otros.

2.2.4. Bonus: utilizando nuestra propia mano (0.3 puntos)

En esta última parte, deberás integrar todo lo desarrollado anteriormente para evaluar el desempeño de tu mejor modelo en una imagen completamente nueva.

Para ello, utiliza una fotografía de tu propia mano realizando uno de los tres gestos (*piedra*, *papel* o *tijera*) y realiza las siguientes tareas:

1. Extraer los *landmarks* de la imagen utilizando la misma función empleada en el procesamiento del *dataset* original.
2. Preprocesar las coordenadas de la misma forma en que fueron tratadas los datos de entrenamiento.
3. Utiliza tu mejor modelo entrenado para predecir la clase correspondiente al gesto realizado.
4. Visualiza la imagen junto con los *landmarks* detectados y muestra en pantalla la predicción obtenida.

¿Qué entregar en esta parte?

Para esta actividad se deberá entregar un archivo `DCCachipun.ipynb` en el repositorio **con todas las celdas ejecutadas, de lo contrario no será corregido**. Dicho notebook deberá poder funcionar al momento de la corrección para obtener el puntaje. Aquí debes incluir tanto el código como las respuestas teóricas. Recuerda que debes justificar tus decisiones y explicar tu procedimiento, para cada una de las actividades.

Importante: No debes subir los datos entregados a tu repositorio de github, de lo contrario habrá un penalización importante en tu nota.

3. DCCarreras (2.75 pts.)



3.1. Introducción

Como estudiante del DCC que estas cursando el ramo de Inteligencia Artificial, te sientes muy motivado a aplicar los conocimientos adquiridos durante el curso a un proyecto que combine el análisis de datos y el modelamiento predictivo a un contexto real e interesante, pero no se te ocurre un problema al que aplicarlo. Un buen día, te encuentras conversando con una amiga de ingeniería mecánica que es fanática de los autos, en particular de la Fórmula 1 (F1). Ella te comenta que siempre le ha intrigado la idea de predecir las posiciones finales de los pilotos, y en particular quién va a ganar la próxima carrera porque es un deporte con mucha incertidumbre.

Esta idea nace de la motivación de tu amiga por comprender cómo es que múltiples factores —como el rendimiento en las sesiones de práctica, la clasificación en estas, las condiciones de la pista o las estrategias de carrera— influyen en el resultado final.

Esta conversación enciende en ti una ampolleta, ¡es la idea que tanto estabas buscando para tu proyecto!. Por lo que decides crear un programa para generar predicciones para las próximas carreras utilizando información histórica de la F1 y técnicas de aprendizaje automático (Machine Learning).

Para este proyecto se utilizarán los datos entregados por la **API FastF1**, una herramienta especializada que permite acceder a información detallada de cada fin de semana de Fórmula 1: tiempos de vuelta, posiciones, uso de neumáticos, condiciones climáticas y más. Con esta información, se entrenarán modelos que buscan predecir la variable **Race Position**, es decir, la posición final que obtiene cada piloto en la carrera.

3.2. Contexto F1

La Fórmula 1 (F1) es la categoría más importante del automovilismo mundial, donde pilotos y equipos compiten por el mejor rendimiento en circuitos de todo el planeta. Cada fin de semana de carrera (*race weekend*) se estructura en distintas etapas que combinan aspectos técnicos, estratégicos y de desempeño.

Un fin de semana típico comienza con tres sesiones de práctica libre (*Practice 1, Practice 2 y Practice 3*). Durante estas sesiones, los pilotos se familiarizan con el circuito, prueban distintos ajustes del auto (*setups*) y experimentan con diferentes tipos de neumáticos y niveles de combustible. Los datos obtenidos en estas prácticas ayudan a los ingenieros a optimizar la configuración del monoplaza para el resto del fin de semana.

Posteriormente se realiza la clasificación (*Qualifying*), compuesta por tres fases eliminatorias: Q1, Q2 y Q3. En cada fase, los pilotos intentan marcar su mejor tiempo de vuelta, y los más lentos van quedando fuera progresivamente. El resultado final de la clasificación determina el orden de salida (*grid position*) para la carrera principal.

El evento principal se lleva a cabo el domingo, con la carrera (*Race*). En esta instancia, los pilotos deben completar un número determinado de vueltas, y el reglamento obliga a realizar al menos una parada en *boxes* (*pit stop*), donde se cambia de neumáticos. Cada parada implica una pérdida aproximada de 20 segundos, lo que convierte la estrategia de *pit stop* en un factor decisivo para el resultado final. Factores como la estrategia, el clima o los incidentes en pista pueden alterar significativamente el desenlace de la competencia.

3.3. Datos

Para la realización de este trabajo se entregará el conjunto de datos (*dataset*) `races.csv` que abarca información de las temporadas **2023**, **2024** y **2025** (hasta la fecha actual). El *dataset* contiene variables provenientes de las distintas etapas del fin de semana de carrera, permitiendo analizar patrones de rendimiento entre pilotos y equipos. Este *dataset* fue extraído directamente de la *API* descrita anteriormente. Este archivo lo podrás encontrar en la sección archivos del curso en Canvas.

El conjunto de datos entregado incluye las siguientes columnas principales:

- **Year:** Año correspondiente a la carrera (por ejemplo, 2024).
- **GrandPrix:** Nombre del Gran Premio (carrera) disputado (por ejemplo, *Spanish Grand Prix*).
- **Driver_ID:** Identificador o número del piloto participante.
- **Driver_Abbrev:** Abreviación del piloto participante.
- **Driver_FullName:** Nombre del piloto participante.
- **Race_BestLap:** Mejor tiempo de vuelta registrado por el piloto durante la carrera.
- **Race_NumLaps:** Número total de vueltas completadas en la carrera.
- **Practice_1_BestLap, Practice_2_BestLap, Practice_3_BestLap:** Mejor tiempo de vuelta registrado por el piloto en cada una de las tres sesiones de práctica libre.
- **Practice_1_NumLaps, Practice_2_NumLaps, Practice_3_NumLaps:** Número de vueltas completadas por el piloto en cada sesión de práctica.
- **Qualifying_BestLap:** Mejor tiempo de vuelta durante la clasificación.
- **Qualifying_NumLaps:** Número total de vueltas registradas en la clasificación.
- **Race_GridPosition:** Posición de partida en la parrilla de salida, determinada por la clasificación.
- **Race_Position:** Posición final del piloto en la carrera. Esta es la variable **objetivo del proyecto**, es decir, la que se debe predecir mediante los modelos de aprendizaje automático.
- **Race_Status:** Estado final del piloto en la carrera (por ejemplo, *Finished*, *Retired*, *DNF*, etc.).
- **Race_Points:** Puntos obtenidos según la posición final y las reglas vigentes de la Fórmula 1.
- **Race_BestLapTime:** Tiempo de la mejor vuelta del piloto expresado en formato temporal.
- **Race_NumberOfPitStops:** Número de paradas en *boxes* realizadas durante la carrera.

3.4. Actividades

A continuación se describirán las actividades que debes realizar. Recuerda que toda decisión debe ser justificada, y que cualquier resultado debe ser analizado. Si se entrega código sin explicación, este no será considerado.

3.4.1. Lectura y Estudio de los Datos (0.75 pts.)

Para poder trabajar correctamente con los datos, primero debemos entenderlos en profundidad. En esta sección debes hacer lo siguiente:

- Carga los datos entregados del archivo `races.csv`. Una vez cargada la información, comenta sobre los elementos que contiene y su naturaleza. Para esto, responde al menos a estas preguntas: ¿Cuántos datos hay? ¿Cuántos atributos hay y cuáles son? ¿A qué tipo de dato corresponde cada atributo? ¿Cuál es la columna que queremos predecir? ¿Cómo se ve la distribución de las posiciones finales en las carreras?
- Realiza un análisis exploratorio de los datos. Para entenderlos de mejor manera, es muy útil visualizar los datos con gráficos. Te pediremos que hagas al menos **tres gráficos** que te permitan entender mejor los datos. Por ejemplo, puedes graficar:
 - La distribución de las posiciones finales (`Race_Position`)
 - La relación entre la posición en la parrilla (`Race_GridPosition`) y la posición final
 - El rendimiento de los pilotos a lo largo de las temporadas
 - Cualquier otra visualización que consideres relevante

Comenta sobre lo que observas en cada gráfico y qué información te entrega sobre el problema a resolver.

3.4.2. División de Datos (0.3 pts.)

Antes de realizar cualquier preprocesamiento, es fundamental dividir los datos en conjuntos de entrenamiento y prueba para evitar *data leakage*.

- Divide el *dataset* en conjuntos de entrenamiento y prueba. **Importante:** La división debe respetar las carreras como conjunto, es decir, no debes separar aleatoriamente datos de una misma carrera. Por ejemplo, podrías dejar en entrenamiento todos los datos de las carreras *Austrian Grand Prix* de 2023 y 2025 y en prueba *Austrian Grand Prix* 2024 o similar. Justifica tu elección de las proporciones y explica por qué es importante mantener este orden en este tipo de problemas. Asimismo, discute si es necesario mantener el orden temporal de los datos.
- Investiga qué es el *data leakage* y por qué es importante dividir los datos antes de realizar el preprocesamiento.

3.4.3. Preprocesamiento de Datos (0.6 pts.)

Una vez que has dividido los datos, es necesario prepararlos adecuadamente para que puedan ser utilizados por los modelos de aprendizaje automático. **IMPORTANTE: Todas las decisiones de preprocesamiento deben tomarse basándose únicamente en el conjunto de entrenamiento y luego aplicarse al conjunto de prueba.**

- Identifica y maneja los datos faltantes o anómalos en el *dataset*. Es importante considerar que existen múltiples razones por las cuales puede haber datos faltantes: algunos pilotos pueden no haber participado en todas las sesiones de práctica, pueden haber abandonado la carrera, o incluso hay fines de semana donde el formato no incluye las tres sesiones de práctica libre (por ejemplo, en algunos circuitos se realizan solo dos prácticas o se utilizan formatos de carrera especiales como el *Sprint*). Analiza los datos faltantes en el conjunto de entrenamiento, decide qué hacer con ellos y justifica tu decisión. Luego aplica la misma estrategia al conjunto de prueba.

- Analiza todas las columnas del *dataset* y responde: ¿Son todos los atributos útiles para predecir la posición final? ¿Hay atributos que puedan causar *data leakage* (es decir, que contengan información que no estaría disponible antes de la carrera)? Identifica estos atributos y decide qué hacer con ellos. Justifica tus decisiones.
- Convierte las variables categóricas a numéricas cuando sea necesario. Primero responde: ¿Cuáles son las columnas categóricas que deben ser convertidas? ¿Qué método utilizarás para cada una (por ejemplo, *One-Hot Encoding*, *Label Encoding*, etc.)? Justifica tu elección según las características de cada variable. Recuerda que debes ajustar el codificador con el conjunto de entrenamiento y luego aplicarlo al conjunto de prueba.
- Normaliza o estandariza las características numéricas según consideres necesario. Explica qué técnica utilizaste y por qué es apropiada para este problema. **Recuerda:** calcula los parámetros de normalización/estandarización (media, desviación estándar, mínimo, máximo, etc.) usando **solo el conjunto de entrenamiento**, y luego aplica esos mismos parámetros al conjunto de prueba.

3.4.4. Entrenamiento de Modelos (0.5 pts.)

A continuación debes seleccionar y entrenar modelos predictivos de aprendizaje automático para resolver esta tarea de regresión. Para esto:

- Usa regresión lineal de mínimos cuadrados y otro algoritmo de regresión. Algunos ejemplos pueden ser: *Árboles de Decisión*, *Random Forest*, *Gradient Boosting*, *K-Nearest Neighbors*, entre otros. Justifica por qué elegiste ese algoritmo considerando las características del problema. Explica brevemente como funciona el algoritmo escogido.
- Para el algoritmo seleccionado, identifica sus hiperparámetros más importantes y selecciona al menos un conjunto apropiado de valores para probarlos. Justifica tu elección de hiperparámetros y explica cómo estos pueden afectar el rendimiento del modelo.
- Implementa una estrategia de validación apropiada para seleccionar los mejores hiperparámetros. Puedes usar técnicas como *Grid Search*, *Random Search* o validación cruzada temporal (*Time Series Cross-Validation*). Justifica tu elección de estrategia.
- Entrena los modelos seleccionados con los mejores hiperparámetros encontrados y documenta el proceso de entrenamiento.

3.4.5. Análisis de Resultados (0.6 pts.)

Una vez entrenados los modelos, debes evaluar su desempeño y analizar los resultados obtenidos. Para esto:

- Evalúa los modelos entrenados utilizando al menos las siguientes métricas de regresión:
 - *Mean Absolute Error* (MAE)
 - *Root Mean Squared Error* (RMSE)
 - *R² Score* (Coeficiente de determinación)

Explica brevemente qué indica cada una de estas métricas en el contexto de este problema.

- Investiga y selecciona al menos **una métrica adicional** que sea apropiada para evaluar problemas de predicción de *rankings* o posiciones.

Explica qué métrica elegiste, qué información proporciona que las métricas anteriores no entregan, y por qué es especialmente útil para evaluar predicciones de posiciones en carreras de Fórmula 1. **Recuerda citar las fuentes que utilizaste para investigar esta métrica.**

- Calcula todas las métricas mencionadas (las tres básicas más tu métrica adicional) para cada modelo tanto en el conjunto de entrenamiento como en el conjunto de prueba. Presenta los resultados en una tabla comparativa. ¿Observas señales de sobreajuste (*overfitting*) o subajuste (*underfitting*) en alguno de los modelos? Explica.
- Reflexiona sobre por qué métricas como *accuracy* (usada en clasificación) no son apropiadas para este problema. ¿Qué diferencia fundamental hay entre un problema de clasificación y un problema de regresión de posiciones/*rankings*?
- Crea al menos dos visualizaciones que te permitan analizar el desempeño de los modelos. Por ejemplo:
 - Gráfico de predicciones vs. valores reales
 - Gráfico de residuos (errores de predicción)
 - Comparación del error por carrera o por piloto

Comenta qué información te entregan estas visualizaciones sobre el desempeño de tus modelos.

- Basado en los resultados obtenidos, decide cuál de los modelos es el mejor para este problema en particular. Justifica tu elección considerando tanto las métricas cuantitativas como el análisis cualitativo. Discute sobre los aspectos puntuales del algoritmo ganador que pueden haber ayudado a obtener mejores resultados.
- **Análisis de errores:** Identifica en qué casos tu mejor modelo falla más. ¿Hay ciertos pilotos, equipos o circuitos donde las predicciones son menos precisas? ¿A qué crees que se debe esto? Realiza un análisis crítico de las limitaciones de tu modelo.

3.4.6. Bonus (Hasta 1 punto en total)

Esta sección es opcional pero puede otorgar puntos adicionales. Puedes realizar una o más de las siguientes actividades:

Para el bonus, puedes utilizar información adicional mediante la *API* pública de **FastF1**, o incluso combinarla con otras fuentes de datos de Fórmula 1, con el fin de enriquecer el análisis y mejorar la capacidad predictiva de los modelos.

La librería **FastF1** permite extraer datos detallados de las distintas sesiones de un fin de semana de Fórmula 1. A continuación, se muestra un ejemplo básico de cómo instalar la librería, cargar una sesión y guardar los datos más relevantes:

```

1 # Documentacion: https://docs.fastf1.dev/
2 !pip install fastf1
3
4 import fastf1
5
6 session = fastf1.get_session(year, gp_name, session_type)
7 session.load(laps=True, telemetry=True, weather=True, messages=False)
8
9 session.results.to_csv(f '{folder}/results.csv' , index=False)
10
11 laps = session.laps
12 laps.to_csv(f '{folder}/laps.csv' , index=False)

```

Listing 1: Ejemplo de uso de la API FastF1

Cabe destacar que la *API* ofrece muchos más *endpoints* y opciones, como datos de telemetría, condiciones climáticas, información de neumáticos y mensajes de radio, que pueden ser explorados para ampliar el

análisis y construir modelos más robustos y precisos. Para esto recomendamos revisar la documentación de la *API FastF1*⁵.

- **Enriquecimiento de datos** (Hasta 0.5 puntos dependiendo de la complejidad): Utiliza la *API FastF1* para obtener datos adicionales que no están en el `races.csv` proporcionado. Algunos ejemplos pueden ser:

- Datos de telemetría (velocidades, aceleraciones, uso de frenos, DRS)
- Información detallada de cada vuelta (*lap times*, sectores)
- Condiciones climáticas durante las sesiones
- Estrategias de neumáticos utilizadas
- Cualquier otra información que consideres relevante

Documenta qué datos agregaste, justifica su relevancia, cómo los obtuviste y re-entrena tus modelos incluyendo esta información. Compara los resultados con los obtenidos anteriormente y discute si hubo mejoras.

- **Predicción en tiempo real** (0.2 puntos por entregar. La persona que obtenga la mejor predicción obtendrá 0.3 puntos más, la segunda 0.2 puntos y la tercera 0.1 puntos): El Gran Premio de Brasil se realizará **el domingo 9 de noviembre de 2025**, es decir, después de que se entregue la tarea. Para este bonus debes realizar lo siguiente:

- Utilizar tu mejor modelo entrenado para predecir las posiciones finales de la carrera del Gran Premio de Brasil 2025
- Incluir en tu *notebook* las predicciones para cada piloto
- Después de la carrera (domingo 9 de noviembre), se compararán tus predicciones con los resultados reales

Este bonus te permite demostrar que tu modelo tiene capacidad predictiva en un escenario real y actual. Para entregar este bonus, deberás entregar en este *forms*⁶ una foto en la que se muestre tu código ejecutado junto con la predicción de las posiciones finales de todos los corredores. El plazo máximo para responder este formulario será el **domingo 9 de noviembre a las 14:00**. Nótese que este plazo es posterior a la entrega de la tarea, por lo que la entrega de este es completamente voluntaria y no se descontaran cupones de atraso por aquello.

¿Qué entregar en esta parte?

Para esta actividad se deberá entregar un archivo `DCCarreras.ipynb` en el repositorio **con todas las celdas ejecutadas, de lo contrario no será corregido**. Dicho *notebook* deberá poder funcionar al momento de la corrección para obtener el puntaje. Aquí debes incluir tanto el código como las respuestas teóricas. Recuerda que debes justificar tus decisiones y explicar tu procedimiento, para cada una de las actividades.

Importante: No debes subir los datos entregados a tu repositorio de *github*, de lo contrario habrá un penalización importante en tu nota.

⁵<https://docs.fastf1.dev/>

⁶<https://forms.gle/1ycHt5ZeCMXg5sDbA>