PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
Facultad de Ingeniería
IIC2613 Inteligencia Artificial
2023-2

Tarea 5
Aprendizaje Reforzado y Redes Neuronales

Fecha de entrega: Lunes 11 de diciembre a las 23:59 hrs

Aspectos generales

Formato y plazo de entrega

El formato de entrega son archivos con extensión `.ipynb` con un PDF para las respuestas teóricas y archivos de extensión `.py` para el apartado de aprendizaje reforzado. El lugar de entrega es en el repositorio de la tarea, en la branch por defecto, hasta el lunes 11 de diciembre 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 **no hábiles** 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 redes neuronales convolucionales para llevar a cabo tareas de clasificación sobre conjuntos de datos y el uso del algoritmo Q-Learning para desarrollar una política de comportamiento de un agente. 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.

1. DCCasas

IMPORTANTE: Para esta parte está EXPRESAMENTE PROHIBIDO subir el set de datos utilizado a tu repositorio en GitHub. En caso contrario un descuento será aplicado a tu entrega, por lo que procura subir solamente el notebook y las respuestas de desarrollo.

En esta tarea utilizarán el conjunto de datos *Multimodal House Prices* consistente en datos tabulares y visuales, donde a partir de estos deberán estimar el valor de mercado de las propiedades analizadas. La parte tabular de los datos se encuentra contenida en el archivo `data.csv` y consiste en 12519 registros, donde cada uno considera las siguientes columnas:

- `image_id`: identificador de la imagen correspondiente a la casa.
- `street`: dirección de la casa.
- `city`: ciudad donde se encuentra ubicada la casa.
- `bedrooms`: número de dormitorios.
- `bathrooms`: número de baños.
- `sqft`: superficie de la propiedad en pies cuadrados.
- `price`: precio de la casa.

La parte visual de los datos está conformada por 12519 archivos, presentando cada uno de estos una fotografía del exterior de la propiedad. La resolución, perspectiva y distancia de cada fotografía es potencialmente distinta.

Para facilitar el trabajo, en el código base del notebook se te entrega una celda que contiene todo lo necesario para cargar las imágenes de las propiedades (cada línea de la celda tiene un comentario explicando lo que hace, por si se quiere saber a detalle cómo se están cargando los datos). Lo único que debes hacer es correr esta celda y obtendrás tres arrays:

- **X_mlp**: Contiene las características tabulares de las 12518 propiedades, para utilizar en la MLP.
- **X_cnn**: Contiene las imágenes de las propiedades, procesadas y redimensionadas para quedar de dimensiones 12518 x 311 x 415 x 3. Este array lo debes usar para la parte de CNN.
- **y**: Es el vector que contiene el valor objetivo `price` de cada propiedad en la base de datos. Este vector ya está en formato numérico y listo para ser utilizado.

Tener en cuenta que igual se deberán hacer cambios en los arrays para poder utilizarlos en las redes neuronales.

Nota: Es normal que la celda mencionada se demore 5 minutos o más en correr, y el entrenamiento de las redes también toma su tiempo, por lo que recomendamos fuertemente que ocupen notebooks de Google Colab para realizar la tarea. Si decide utilizar Colab para desarrollar la tarea, se recomienda mantener una copia del conjunto de datos en una unidad de Google Drive, con el fin de no realizar la carga de datos desde sus computadores personales a la nube cada vez que trabajen en su solución.

Importante: Recuerda que debes separar las bases en entrenamiento, testeo y validación. Los porcentajes que elijas para cada set de datos quedan a tu criterio y deben ser justificados.

1.1. Comprendiendo los datos

Antes de armar una red neuronal, por lo general existe un proceso meticuloso de preparación y comprensión de los datos. Esta fase preliminar, conocida como preprocesamiento, juega un papel crucial en la determinación del éxito de nuestros modelos.

- Investiga y describe por qué el preprocesamiento es una etapa crucial para las redes neuronales. Explica cómo los pasos de preprocesamiento pueden afectar el rendimiento de un modelo.
- Si tuviéramos pocas imágenes para entrenar un modelo, ¿qué técnicas podrías usar para enriquecer el conjunto de datos? Investiga sobre el aumento de datos en imágenes.

1.2. Explorando los Multilayer Perceptron (MLP)

- Discute cómo la profundidad y el ancho (número de capas y neuronas por capa) de un MLP afectan su capacidad para aprender patrones complejos. ¿Cuáles son los desafíos relacionados con el aumento de la complejidad del modelo?
- Investiga y compara (a nivel teórico) al menos dos funciones de activación no lineales diferentes utilizadas en MLPs. ¿Cómo afectan estas funciones al tipo de decisiones que puede aprender la red? Considera aspectos como la saturación y la no linealidad.
- Examina las causas y síntomas del overfitting en MLP. ¿Qué técnicas se pueden aplicar para prevenir este problema y cómo afectan el proceso de aprendizaje?
- Más allá del SGD (Stochastic Gradient Descent) y Adam, investiga sobre otro método de optimización utilizado en el entrenamiento de MLP. ¿Cuáles son sus características únicas y en qué situaciones podría ser preferible?
- Identifica y analiza un estudio de caso donde se haya utilizado un MLP para resolver un problema real. ¿Qué características del problema hicieron que un MLP fuera una buena elección y cómo se diseñó la arquitectura de la red para adaptarse a las necesidades específicas del problema?

1.3. Implementación de un Multilayer Perceptron (MLP)

Utilizando únicamente la parte tabular del conjunto de datos y un Multilayer Perceptron (MLP), construya un sistema predictor del precio de cada casa. Para evaluar el rendimiento del modelo, considere la creación de un conjunto de prueba independiente aleatorio, es decir, utilizando una partición aleatoria de la parte tabular de los datos. Se debe cumplir lo siguiente:

- Implementa un MLP con al menos dos capas ocultas. Entrena tu modelo en el conjunto de datos de entrenamiento (solamente `X_mlp`) y realiza ajustes en los hiperparámetros (como la tasa de aprendizaje, número de neuronas, funciones de activación) para mejorar el rendimiento del modelo.
- Evalúa el rendimiento de tu modelo en el conjunto de datos de prueba. Utiliza métricas relevantes para problemas de regresión, como el error cuadrático medio (MSE).
- Realiza al menos dos experimentos variando la arquitectura del MLP o los hiperparámetros. Describe cómo cada cambio afecta el rendimiento del modelo y discute tus hallazgos.
- Basado en tus resultados, ¿cuáles son las características más importantes de un MLP para la predicción precisa en este conjunto de datos? ¿Cómo relacionas esto con la teoría aprendida?
- Describe los desafíos que encontraste al implementar y entrenar el MLP. ¿Cómo los superaste y qué aprendiste en el proceso? ¿en qué otros tipos de problemas crees que un MLP podría ser efectivo?

1.4. Introducción y teoría de las CNN's

Investiga qué es una Red Neuronal Convolucional, y responde de manera breve las siguientes preguntas teóricas. Puedes encontrar información que te será útil en el siguiente **artículo**, el cual está orientado al uso de CNN's en imágenes y uso de la librería *Keras* para tareas de visión.

- ¿Qué es una operación convolucional? ¿Qué es un *kernel*? Utiliza estos conceptos para explicar el rol de las capas convolucionales en una CNN.
- ¿Cuál es el rol de las funciones de activación? ¿Y de las capas de *Max Pooling*?
- Quizás habrás notado que la mayoría de arquitecturas de CNN's utilizan una última capa conocida como *flatten layer*. ¿Cúal es su función? ¿Cuál es el rol de la función *softmax* en ella?
- ¿Cuáles son las ventajas y desventajas de las CNN's frente a las MLP's? ¿Para qué tipo de tareas suele ser útil utilizar CNN's?

1.5. Creando y evaluando una CNN

A continuación, construye una red neuronal convolucional que al igual que las MLP corresponda a un sistema predictor de precios de casas, pero esta vez utiliza la porción visual de los datos. La estructura de la red queda completamente a tu criterio, pero te recomendamos que revises las arquitecturas más populares para que puedas guiarte, como *VGG-16, AlexNet, Le-Net5, ResNet-50, Inception-v1*, etc. **HINT:** Ten en cuenta que la mayoría de estas redes abordan el problema de **clasificación**, mientras que el problema que queremos abordar es de **regresión**, por lo que deberás investigar qué se suele hacer en las arquitecturas para este tipo de problemas.

Para poder evaluar tu modelo podrás elegir implementar alguna de las siguientes métricas. Primero, podrás utilizar el accuracy, el cual se entiende para nuestro set de datos como:

$$accuracy = \frac{|precio\ real - prediccion\ de\ precio|}{precio\ real} \times 100\,\%$$

Otra forma posible de evaluar tu modelo (y que es más común para este tipo de datos) es el error cuadrático medio MSE, el cual definiremos como:

$$MSE = \frac{1}{N}\sum_{t=1}^{N}(precio\ real_t - prediccion\ de\ precio_t)^2$$

Luego de implementar alguna de estas métricas de rendimiento, explica a partir de ellas el rendimiento de tu modelo. Por último, deberás graficar la función de pérdida en entrenamiento y validación, explicar su utilidad y para qué sirve en tu modelo.

Se espera que crees esta arquitectura utilizando la API de alto nivel de *Tensorflow*, Keras.

1.6. Comparación de modelos

A continuación, responde las siguientes preguntas relacionadas a comparar ambos tipos de redes neuronales:

- ¿Cuál de los modelos tiene un mejor *accuracy*?
- ¿Qué rangos de precios tienen un mejor rendimiento? ¿Y cuáles el peor? ¿Depende del modelo?

2. DCCanario

En esta parte ocuparás aprendizaje reforzado (*Reinforcement Learning*) para entrenar una IA que sea capaz de aprender a jugar al clásico juego Flappy Bird por sí misma:

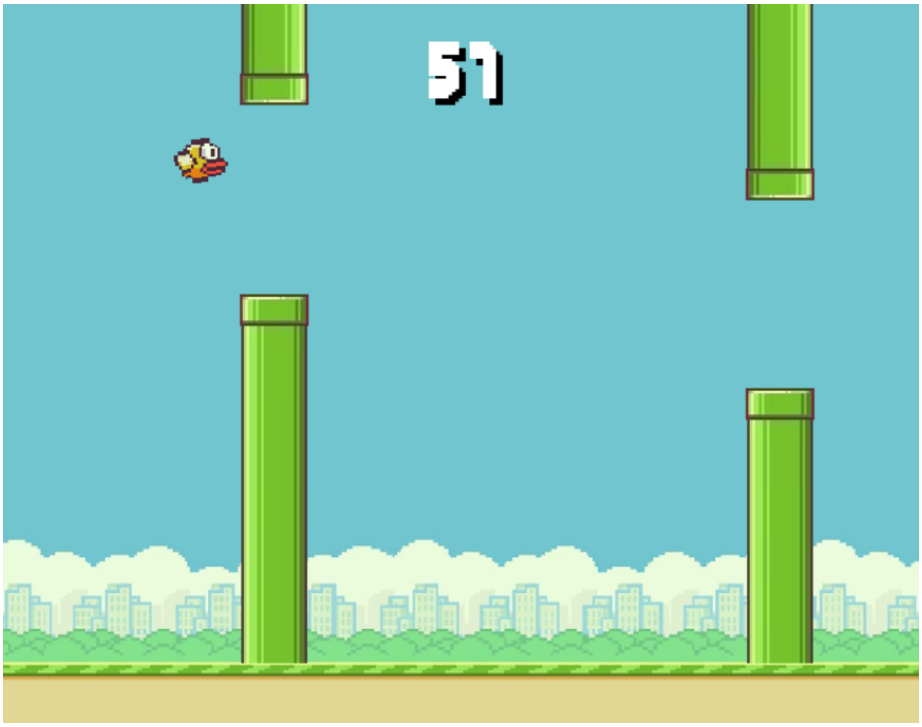

Figura 1: GUI del juego Flappy Bird

En tu repositorio se te da el código del juego Flappy Bird, pero el personaje (un pájaro que salta a lo largo el eje Y) decide si saltar o no de manera aleatoria. Deberás implementar el algoritmo *Q-Learning* para que pueda moverse de forma "inteligente", tratando de lograr el mayor puntaje posible (atravesando el mayor número de tuberías consecutivamente), además de responder las preguntas en un archivo pdf.

Los archivos contenidos en el código base son:

- `FlappyBirdAI.py`: Código con la interfaz gráfica y funcionamiento general del juego. **No modificar**.
- `QAgent.py`: Este archivo contiene el modelo de un agente que se mueve de forma aleatoria. Es aquí donde deberás trabajar y aplicar tus conocimientos de aprendizaje reforzado.

Como ya se mencionó anteriormente, el archivo sobre el cual debes trabajar e implementar *Q-Learning* es `QAgent.py`[1]. Este archivo contiene la clase `Agent` con los siguientes métodos:

- `__init__()`: Este método inicializa los atributos del agente. En el caso del agente aleatorio, solo se inicializan en 0 las partidas jugadas por el agente.
- `get_state(game)`: Este método consulta al juego por el estado actual del agente y lo retorna como un vector o tupla de 5 dimensiones, donde cada una de las entradas representa la siguiente información (en orden):

[1]Este archivo se encuentra muy bien comentado, por lo que se recomienda leerlo como parte del enunciado.

1. **Distancia en el eje X:** Toma un valor entero entre 0 y 50 correspondiente a la distancia entre el jugador y la siguiente tubería en el eje X (corresponde a una subdivisión del espacio posible en 51 partes iguales).

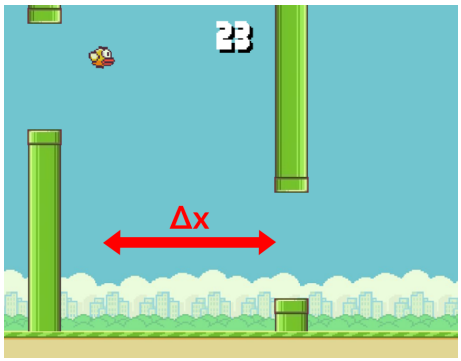

Figura 2: Proximidad entre el jugador y la siguiente tubería

2. **Distancia en el eje Y:** Toma un valor entero entre 0 y 27 que representa la distancia entre el jugador y el centro del agujero de la siguiente tubería en el eje Y, tal como se muestra en la figura:

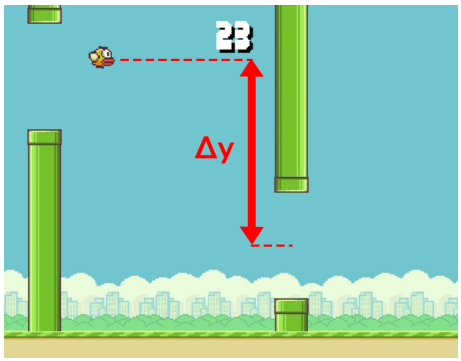

Figura 3: Proximidad del jugador y el agujero siguiente

3. **Sentido de la próxima tubería:** Es un entero correspondiente a la dirección en que se encontrará el próximo agujero (0 si se encuentra encima del jugador y 1 si se encuentra debajo de él).
4. **Distancia en el eje Y de la subsiguiente tubería:** Toma un valor entero entre 0 y 27 que representa la distancia entre el jugador y el centro del agujero de la tubería que vendrá después de la siguiente, tal como se muestra en la figura:

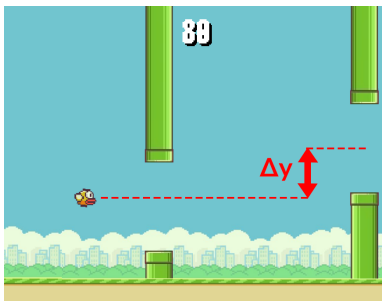

Figura 4: Proximidad entre el jugador y el agujero subsiguiente

5. **Sentido de la próxima tubería subsiguiente:** Es un entero correspondiente a la dirección en que se encontrará el agujero subsiguiente (0 si se encuentra encima del jugador y 1 si se encuentra debajo de él).

- `get_action(state)`: Este método recibe el estado del agente y retorna un entero representando la acción que debe tomar el agente. Las acciones posibles son:
  - `0`: No moverse.
  - `1`: Saltar.

  Nota: En el caso del agente aleatorio, este no utilizará la información que entrega el estado y retornará una acción al azar.

Fuera de esta clase, sólo existe la función `train()`. Esta función es la encargada de entrenar al agente y es la que se llama al correr el archivo de ejecución principal.

Actividad 1: Comprendiendo los hiperparámetros

En esta actividad tendrás que pensar el problema antes de comenzar a programarlo, para ello, responde lo siguiente:

- ¿Cuál es la función del *exploration rate*?
- ¿Cuál es el problema de tener un *exploration rate* mínimo con un valor muy bajo y un *exploration decay rate* con un valor muy alto?
- ¿Qué pasa si el *exploration decay rate* es demasiado bajo?

En qué tipo de situaciones o cosas se podrían reflejar los cambios en estos hiperparámetros, contextualizado en este problema (juego Flappy Bird).

Actividad 2: Implementando Q-Learning

Basándote en lo visto en clases y ayudantías, implementa el algoritmo *Q-Learning*[2] para el agente del juego Flappy Bird, esto es, el pájaro. Para esto deberás realizar los siguientes pasos[3] (recuerda comentar todo lo que hagas en tu código):

1. Inicializar la *Q-Table* del agente en el método `__init__` con sus dimensiones correctas **(0.3 pts.)**.
2. Modificar el método `get_action` para que el agente sea capaz de explorar el estado actual o explotar la mejor acción conocida hasta el momento **(0.6 pts.)**.
3. Actualizar la *Q-Table* una vez que se ejecute la acción seleccionada por el agente **(0.5 pts.)**.
4. En caso de terminar una partida, actualizar el *exploration rate* del agente **(0.4 pts.)**.

[2]Si tienes dudas con *Q-Learning*, puedes revisar este artículo.
[3]Se recomienda revisar el código base, específicamente los `#IMPLEMENTAR` que hay en el código

IMPORTANTE Y OBLIGATORIO

Para facilitar la corrección de tu tarea, te pediremos subir la *Q-Table* de tu agente ya entrenado como un archivo en formato `q_table.csv` sin *headers* (filas/columnas con títulos o nombres).

El archivo .csv debe poseer 7 columnas y 159936 filas. Las primeras 5 columnas corresponden a los posibles valores de las 5 dimensiones de los estados en el orden que se mencionan en este enunciado. Por otra parte, las siguientes 2 columnas corresponden al *Q-Value* de las acciones posibles del agente también en el orden del enunciado (primero, no moverse y segundo, saltar). De esta forma, cada fila representa un posible estado y los valores de las acciones posibles para ese estado. Una fila de este archivo se podría ver así:

$$[41,\ 17,\ 1,\ 9,\ 0,\ 1.9878,\ 99.346]$$

Si todavía tienes dudas al respecto, puedes generar una issue en este enlace.

Actividad 3: Análisis de parámetros del agente

Una vez hayas programado tu modelo y este sea capaz de aprender, juega con los hiper-parámetros y responde las siguientes preguntas:

- ¿Qué rol cumple la tasa de descuento en *Q-Learning*?
- ¿Qué tasa de descuento te dio mejores resultados? ¿Por qué crees que fue así?
- ¿Para qué sirve el *learning rate*? ¿Qué valor te entregó mejores resultados? Comenta los resultados.

Actividad 4: Nueva política de recompensas

Actualmente, el juego otorga recompensas al agente si se acerca hacia el agujero de la tubería, sumando 1 punto, o bien, restando 1 punto si se está alejando de esta. Además, castiga al agente con -1 punto si pierde la partida y lo premia con 50 cada vez que pasa a traves de una tubería.

Para esta actividad deberás diseñar (no es necesario implementar) una política para recompensa distinta a la implementada en el archivo `FlappyBirdAI.py` para poder entrenar al agente y responde:

- ¿Por qué crees que sería una buena forma de recompensar al agente? Argumenta.

Comentarios Finales

Como podrás notar, buena parte de esta tarea involucra respuestas de desarrollo escrito, donde debes transparentar tu razonamiento y explicar las decisiones que tomas. Por este motivo, para que una respuesta se considere correcta, debes tener cuidado de fundamentar apropiadamente lo que digas, aludiendo a referencias confiables y a la documentación de las librerías que utilices. Por ejemplo, si te pedimos explicar un hiperparámetro en particular, o calcular una métrica de desempeño, se espera que demuestres un dominio general de lo que hace dicho hiperparámetro, o de la información que entrega la métrica solicitada. Para esto, es esperable que busques recursos (libros, artículos, documentación oficial, etc.) para fundamentar tus respuestas, donde debes indicar claramente la fuente de tal recurso.

Al mismo tiempo, es posible que al intentar ejecutar operaciones costosas (como entrenar un modelo sobre un conjunto de datos), la ejecución sea más lenta de lo que esperas. Esto es un escenario cotidiano al trabajar con modelos de aprendizaje de máquina, de modo que se espera que seas capaz de lidiar con tales situaciones, y que transparentes y fundamentes las decisiones que tomes en el proceso.

Política de Integridad Académica

Los/as estudiantes de la Escuela de Ingeniería de la Pontificia Universidad Católica de Chile deben mantener un comportamiento acorde a la Declaración de Principios de la Universidad. En particular, se espera que mantengan altos estándares de honestidad académica. Cualquier acto deshonesto o fraude académico está prohibido; los/as estudiantes que incurran en este tipo de acciones se exponen a un Procedimiento Sumario. Es responsabilidad de cada estudiante conocer y respetar el documento sobre Integridad Académica publicado por la Dirección de Docencia de la Escuela de Ingeniería.

Específicamente, para los cursos del Departamento de Ciencia de la Computación, rige obligatoriamente la siguiente política de integridad académica. Todo trabajo presentado por un/a estudiante para los efectos de la evaluación de un curso debe ser hecho **individualmente** por el/la estudiante, **sin apoyo en material de terceros**. Por "trabajo" se entiende en general las interrogaciones escritas, las tareas de programación u otras, los trabajos de laboratorio, los proyectos, el examen, entre otros.

En particular, si un/a estudiante copia un trabajo, o si a un/a estudiante se le prueba que compró o intentó comprar un trabajo, **obtendrá nota final 1.1 en el curso** y se solicitará a la Dirección de Pregrado de la Escuela de Ingeniería que no le permita retirar el curso de la carga académica semestral.

Por "copia" se entiende incluir en el trabajo presentado como propio, partes hechas por otra persona. En caso que corresponda a "copia" a otros estudiantes, la sanción anterior se aplicará a todos los involucrados. En todos los casos, se informará a la Dirección de Pregrado de la Escuela de Ingeniería para que tome sanciones adicionales si lo estima conveniente.

También se entiende por copia extraer contenido sin modificarlo sustancialmente desde fuentes digitales como Wikipedia o mediante el uso de asistentes inteligentes como ChatGPT o Copilot. Se entiende que una modificación sustancial involucra el análisis crítico de la información extraída y en consecuencia todas las modificaciones y mejoras que de este análisis se desprendan. Cualquiera sea el caso, el uso de fuentes bibliográficas, digitales o asistentes debe declararse de forma explícita, y debe indicarse cómo el/la estudiante mejoró la información extraída para cumplir con los objetivos de la actividad evaluativa.

Obviamente, está permitido usar material disponible públicamente, por ejemplo, libros o contenidos tomados de Internet, **siempre y cuando se incluya la referencia correspondiente**.

Lo anterior se entiende como complemento al Reglamento del Estudiante de la Pontificia Universidad Católica de Chile (`https://registrosacademicos.uc.cl/reglamentos/estudiantiles/`). Por ello, es posible pedir a la Universidad la aplicación de sanciones adicionales especificadas en dicho reglamento.

Compromiso del Código de Honor

Este curso suscribe el Código de Honor establecido por la Universidad, el que es vinculante. Todo trabajo evaluado en este curso debe ser propio. En caso que exista colaboración permitida con otros/as estudiantes, el trabajo deberá referenciar y atribuir correctamente dicha contribución a quien corresponda. Como estudiante es un debe conocer el Código de Honor (`https://www.uc.cl/codigo-de-honor/`).