PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
Facultad de Ingeniería
IIC2613 Inteligencia Artificial
2024-2

Tarea 4
Aprendizaje de Máquina

Fecha de entrega: Lunes 4 de Noviembre a las 23:59 hrs

Aspectos generales

Formato y plazo de entrega

El formato de entrega para la primera sección de la tarea (DCChampiñones) es **únicamente** un archivo con extensión `.ipynb` para tanto las respuestas de código como las teóricas. Para la segunda sección **COMPLETAR** .El lugar de entrega es en el repositorio de la tarea, en la branch por defecto, hasta el lunes 4 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, entendemos y motivamos el uso de material hecho por otras personas y por asistentes basados en IA, por lo que es importante reconocerlo de la forma apropiada. Todos los recursos externos 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. Asimismo, si utilizas información generada por inteligencia artificial, debes documentarlo según se detalla en el programa del curso.

Comentarios adicionales

El objetivo de esta tarea es que puedan utilizar distintas técnicas de aprendizaje de máquina, aplicándolas 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.

1. Clasificador de DCChampiñones venenosos (2 pts.)

1.1. Introducción

Figura 1: Alumnos investigando para el emprendimiento "DCChampiñones"

Un grupo de alumnos del curso "Inteligencia Artificial" comenzó a planear sus próximas vacaciones de verano. Sin embargo, se encontraron con un obstáculo importante: no hay dinero. Para poder recuadar fondos, decidieron lanzar un emprendimiento llamado "DCChampiñones", en el que venderán hongos que recolectarán cerca de sus casas.

Todo marchaba bien, hasta que surgió una inquietud crucial: ¿**cómo sabemos si un hongo es venenoso**, si en el curso no se enseña acerca de hongos? Afortunadamente, recordaron uno de los últimos conceptos que aprendieron en el curso: *machine learning*. ¿Podrán los modelos de *machine learning* predecir con alta precisión si un hongo es venenoso o no?

Deciden que solo hay una manera de averiguarlo, y esta es poniéndolo a prueba. Por lo tanto, para esta parte de la tarea, se les proporcionará un conjunto de datos con de hongos, el cual contiene diferentes atributos de ellos. Luego, mediante técnicas de preprocesamiento de los datos, entrenamiento de modelos y evaluación de ellos, deberán determinar la efectividad del *machine learning* para predecir si un hongo es venenoso o no.

1.2. Recursos para realizar la tarea

Usaremos principalmente 2 librerías para esta tarea, las cuales serán explicadas a continuación, sumado al uso de **Google Colab**.

1.2.1. Pandas

Esta librería[1] 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.

[1] https://pandas.pydata.org/

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**:

```
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**. Mientras el primero devuelve características (número de columnas, cuáles son y el número de elementos no nulos), 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 **isnull**. Los elementos nulos aparecen con el denominativo **NaN**, y el método **isnull** 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:

```
#esta es una operacion simple sobre dos celdas del DataFrame
q = df["name1"][354] + df["name2"][726]

#estas son operaciones sobre toda la serie
m = df["name3"].mean()
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:

```
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:

```
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, si lo que se desea es guardar un DataFrame procesado, se puede usar el método **to_csv**:

```
df.to_csv("nombre del archivo csv")
```

1.2.2. Scikit-learn

Adicionalmente, usaremos la librera scikit-learn [2], la cual provee una gran gama de técnicas de aprendizaje de máquina, tal como clasificadores del tipo **SVM**.
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:

```
#Clase que utilizaremos para un clasificador SVM
from sklearn.svm import SVC

#Clase que utilizaremos para un clasificador Bayesiano ingenuo
from sklearn.naive_bayes import GaussianNB

#Clase que utilizaremos para un clasificador de arbol de decision
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 hiperpárametros 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`.

1.2.3. Google Colab

Si queremos usar estos algoritmos de manera más rápida podemos usar las famosas tarjetas GPUs. 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 [3], por lo que recomendamos fuertemente desarrollar la tarea en esta plataforma, que además permite exportar el archivo como un archivo **.ipynb**, aunque no es requisito.

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

1.3. Descripción del Dataset

Se te entregará el archivo `mushroom.csv`, el cual fue extraído del siguiente link[4], que corresponde a un conjunto de datos tabulares que incluye varias características de hongos, como color, forma, tamaño, y otros atributos relevantes. Cada registro en el dataset indica si el hongo es venenoso o no. Este archivo estará disponible en tu repositorio personal.

1.4. Actividades a realizar:

Recuerda que toda decisión debe ser justificada, y que cualquier resultado debe ser comentado. Si solo se entrega código sin explicación, este no será considerado.

1.4.1. Preprocesamiento del Dataset (1 pto.)

Para poder trabajar correctamente con los datos, debemos asegurarnos de que estén en el formato que se necesita. Para esto, lo primero que debemos hacer es hacer una limpieza y transformación de algunos de ellos para que queden listos para nuestros modelos. En esta sección debes hacer lo siguiente:

[2] https://scikit-learn.org/stable/
[3] https://colab.research.google.com/
[4] https://www.kaggle.com/datasets/prishasawhney/mushroom-dataset

- Lee la información contenida en el archivo `mushroom.csv`. Una vez que la información esté cargada, 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 clases a predecir?
  Para entenderlos de mejor manera, es muy útil visualizar los datos con gráficos. Para esto te pediremos que hagas al menos un gráfico para la distribución de las clases.

- A continuación, debes dividir el dataset en conjuntos de entrenamiento y prueba. Justifica tu elección de las proporciones.

- Luego debes identificar datos faltantes o anómalos y decidir que hacer con ellos, cuidando de solo basarse en la información del conjunto de entrenamiento. Asimismo, responde a la pregunta: ¿Son todos los atributos útiles para esta tarea? Justifica, y decide que hacer con ellos. Recuerda que puedes normalizar o estandarizar las características según sea necesario.

- Para terminar el preprocesamiento debes convertir variables categóricas a numéricas. Primero responde a la siguiente pregunta: ¿Por qué debemos hacer esta conversión? ¿Cuáles son las columnas que debemos convertir? Luego decide que método utilizaras para cada una y justifica.

1.4.2. Implementación de Algoritmos de *Machine Learning* (0.3 pts.)

A continuación debes elegir al menos tres modelos predictivos de aprendizaje de máquinas vistos en clases y entrenar dichos modelos para resolver esta tarea. Recuerda que cada uno de estos modelos tiene sus respectivos hiperparámetros, por lo que debes seleccionar un conjunto de ellos para cada uno y justificar debidamente la elección.

1.4.3. Evaluación de Modelos (0.7 pts.)

A continuación debes evaluar los modelos entrenados en la sección anterior. Para esto debes utilizar las siguientes métricas (se recomienda utilizar las métricas ya implementadas en las librerías):

- *Accuracy*
- *Precision*
- *Recall*
- *F1-score*

Explica brevemente que indica cada una de estas métricas en general. Luego, calcula estas métricas y muestra la matriz de confusión para cada modelo.
Responde a la siguiente pregunta: ¿Si queremos evitar clasificar mal los hongos venenosos en qué debemos fijarnos? Discute sobre las implicaciones de los errores de tipo 1 y tipo 2 en el contexto de este problema (considera el contexto inicial). Por último, basado en lo anterior decide cual de los tres modelos es el que obtiene los mejores resultados para este problema en particular y explica porque este puede haber funcionado mejor en este caso. Para esto habla sobre los aspectos puntuales del algoritmo que pueden haber ayudado.

1.5. Entregables

Un notebook de Jupyter (**.ipynb**) que contenga las respuestas a las secciones anteriores. **Debe ser entregado con todas sus celdas de código ejecutadas, de lo contrario no será corregido.** Recuerda que toda decisión debe ser justificada, y que cualquier resultado debe ser comentado. Asimismo, recuerda comentar el código explicando que hace este.

2. DCCompresor: Compresión de texto usando aprendizaje supervisado (4 pts.)

Introducción

Figura 2: Los compresores intentando comprimir el mapa

Un audaz equipo de viajeros ciberespaciales, conocidos como "Los Compresores", decidió embarcarse en la búsqueda del mayor tesoro del ciberespacio: el legendario botín de datos perdidos en la Galaxia ciberespacial. Armados con su ingenio y un antiguo mapa digital que revelaba la ubicación del tesoro, comenzaron su travesía. Sin embargo, pronto enfrentaron un problema inesperado: el mapa del tesoro, ocupaba demasiados megabytes, y la velocidad de su conexión intergaláctica era muy limitada. Necesitaban enviar el mapa a su nave nodriza, pero tal como estaba, la transferencia tomaría días.

Frustrados, se dieron cuenta de que necesitaban de un experto en programación como tu para poder completar su misión. Gracias a tu amplio conocimiento y habilidades en programación te das cuenta que la clave para continuar con su misión es comprimir el mapa de la manera más eficiente posible. Recordaste que los esquemas de codificación tradicionales, como ASCII o EASCII, usan una cantidad fija de bits para representar cada carácter, lo cual no es óptimo para textos reales que contienen muchas letras y palabras que se repiten. ¿Por qué dedicar 8 bits a cada carácter cuando algunas letras, sílabas y frases son mucho más frecuentes que otras?

Fue entonces cuando recordaste un concepto crucial: la codificación basada en frecuencia. Si logras diseñar un esquema donde los caracteres más comunes tengan representaciones más cortas, podrás reducir el tamaño del archivo. Pero, ¿Y si, además de usar las frecuencias de los caracteres, usas técnicas de Machine Learning para predecir qué caracteres o palabras son más probables en base al contexto? De esta manera, el codificador podrá adaptarse a diferentes partes del mapa y ser más eficiente, para lograr enviar el mapa a tiempo.

Decidiste que es hora de poner a prueba tus habilidades y desarrollar un codificador adaptativo que pueda comprimir y codificar su mapa del tesoro sin perder información, utilizando métodos como el codificador de Huffman o el codificador aritmético. ¿Lograrán "Los Compresores" completar su misión?

2.1. Codificación basada en frecuencia

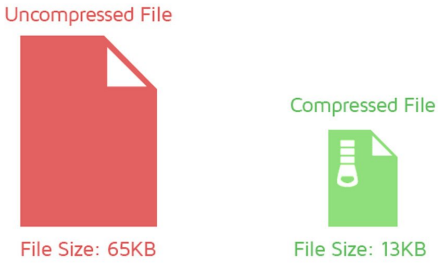

Figura 3: Tamaño de archivos con y sin compresión

Los esquemas tradicionales de codificación de texto utilizan una cantidad fija de bits para representar todos los caracteres. Por ejemplo, en el caso de la codificación EASCII, se utiliza 1 byte (8 bits) para representar cualquiera de los 256 caracteres que soporta. A pesar de que esta simplicidad trae grandes beneficios, los archivos de texto que obedecen a este estilo de codificación tienden a ser más pesados de lo necesario, debido a que contienen una gran cantidad de redundancia en la codificación.

Una forma de remediar esto, es utilizar esquemas de codificación basados en frecuencia. Estos se basan en el hecho de que, de manera natural, algunos caracteres ocurren con mayor frecuencia que otros en los lenguajes humanos. De esta manera, aquellos con más probabilidad de aparecer reciben una codificación que requiere menos bits que las de aquellos caracteres que ocurren menos frecuentemente.

Existe variados algoritmos para realizar codificación basada en frecuencia. Dos de los más utilizados son la **codificación de Huffman**, o la **codificación aritmética**.

2.2. Codificación adaptativa basada en contexto

Siguiendo la misma lógica de la codificación basada en frecuencia, es posible extender estos esquemas al tomar en consideración el hecho que, dependiendo del contexto, hay algunos caracteres que es más probable que ocurran que otros. Por ejemplo, si desde un archivo de texto se lee "*Inteligencia Arti*", es altamente probable que el siguiente carácter a leer desde el archivo sea la letra "*f*" o "*s*". Análogamente, es muy poco probable que el siguiente carácter sea una "*i*" o una "*z*", a pesar de que esta letra aparece regularmente en las palabras, y que un esquema no adaptativo le asignaría una alta probabilidad. Así, un algoritmo de codificación adaptativa usaría todo este conocimiento para optimizar la cantidad de bits necesarios para codificar el siguiente carácter.

Tomando todo lo anterior en consideración, en esta tarea **deberá implementar un esquema de codificación (y decodificación) de texto adaptativo**, de tal manera que sea capaz de comprimir texto simple y luego sea capaz de descomprimirlo, sin que exista pérdida de información. **Todo esto tiene un puntaje de 3.6 puntos.**

Específicamente, el esquema debe utilizar un modelo de lenguaje que realice las predicciones del siguiente carácter en base a una ventana de caracteres previos (*Hint*: considere un token especial de inicio como contexto inicial), las cuales deben ser posteriormente entregadas a un codificador basado en frecuencia en forma de una tabla de frecuencia/probabilidades, para que este realice la codificación. Al repetir este proceso, actualizando el contexto y codificando carácter a carácter, se puede comprimir texto de largo arbitrario.

Similarmente, para descomprimir, dado un contexto ya descomprimido (*Hint*: al igual que antes, considere

un token especial de inicio como contexto inicial), el modelo de lenguaje debe generar las probabilidades del siguiente carácter, que luego son entregadas al codificador para ahora decodificar el siguiente carácter. Al repetir este proceso, actualizando el contexto y decodificando carácter a carácter, se puede descomprimir texto de largo arbitrario.

2.3. Conjunto de datos

Para entrenar sus modelos, puede usar cualquier set de datos de texto disponible, tales como **20 Newsgroups**, o el **Reuters News dataset**. Se recomienda que el conjunto de datos sea lo más extenso posible, con el fin de capturar la mayor cantidad de patrones.

2.4. Modelos de aprendizaje supervisado

En la tarea puede utilizar cualquier modelo de aprendizaje de los vistos hasta el momento en clases. No se permite el uso de modelos más avanzados, o de modelos previamente entrenados. Con el fin de facilitar su labor, se recomienda utiliza la biblioteca *scikit-learn*, que provee implementaciones eficientes de una gran cantidad de algoritmos de aprendizaje supervisado. **Es muy importante que el modelo elegido sea determinista.**

2.5. Preguntas a responder

A continuación, debes responder **una de las siguientes preguntas (0,4 puntos)** . Si respondes más de una, se considerará como puntaje bonus (0.2 por respuesta extra). Sólo se considerarán para bono aquellas respuestas que estén correctamente fundamentadas, ya sea con código o datos.
Las respuestas deben estar en el archivo `dccompresor_train.py`.

- **Modelo políglota**: ¿cómo se comporta la razón de compresión del sistema en textos escritos en idiomas distintos al que fue entrenado?
- **Escalamiento**: ¿cómo se ve afectada la razón de compresión promedio del sistema, al aumentar o disminuir el tamaño del conjunto de entrenamiento y/o al número de épocas de entrenamiento?
- **Sobreajuste**: ¿cuán sensible es el sistema al sobreajuste generado por la complejidad del modelo? Pruebe la mayor cantidad de arquitecturas posible para responder esta pregunta.
- **Variación de los elementos**: ¿Mejora el rendimiento si además de los caracteres, se consideran bigramas, trigramas, etc, en la codificación?

2.6. Algunas consideraciones adicionales

Considere los siguientes aspectos durante el desarrollo de su tarea:

Entrenamiento del modelo de lenguaje

Esta parte es la principal de la tarea. El proceso de entrenamiento debe mostrar claramente el conjunto de datos utilizado, la codificación de entrada y salida, poniendo particular énfasis en como se forma el contexto para predecir la distribución de probabilidades del siguiente carácter.

No se asignará puntaje en base al rendimiento obtenido en la predicción del siguiente carácter, ya que esta parte netamente evalúa si el alumno adquirió las competencias para entrenar un modelo de lenguaje de manera adecuada.

Codificador basado en frecuencia

En esta parte se debe incorporar un codificador como Huffman o Aritmético. Este puede ser programado por usted, o puede ser importado de alguna biblioteca, en dicho caso, debes comentar debidamente cual.

Conexión entre modelo de lenguaje y codificador

En esta parte se debe hacer coincidir el formato de salida del modelo de lenguaje con la entrada del codificador. Es importante que lo hecho tenga sentido, por ejemplo, si la salida del modelo de lenguaje es la probabilidad del carácter más probable, y la entrada para el codificador es una distribución de probabilidades, entonces la conexión es incorrecta, por mucho que se complete el resto de las probabilidades a mano.

Está permitido hacer post-procesamiento de la salida del modelo de lenguaje, para que este sea compatible con el codificador, siempre y cuando esto tenga sentido. Por ejemplo, es correcto usar normalizaciones, traslaciones, truncado, etc, pero tal como se indicó anteriormente, el llenado a mano de probabilidades, de manera arbitraria no tiene puntaje.

2.6.1. Compresión/descompresión

La solución debe cumplir dos cosas: i) el sistema debe reducir, en promedio, el tamaño del texto de entrada al realizar la compresión, es decir, no es necesario que todos los textos de entrada sean comprimidos, y ii) la compresión y posterior descompresión deja al texto en el mismo estado inicial (no hay pérdida de información). Se recomienda utilizar archivos para cuantificar correctamente la reducción del tamaño, es decir, la compresión y luego descompresión de estos. No considere dentro del tamaño al espacio necesario para almacenar el modelo de lenguaje, solo el contenido de los archivos.

2.7. Bonus

Mejor rendimiento del curso

Para las 3 tareas que presenten la mejor razón de compresión en los archivos de prueba se otorgaran 5 décimas de bonus. Para las siguientes 5 tareas (es decir, del cuarto al octavo) se otorgaran 3 décimas. tareas que presenten mejor razón de compresión en los archivos de prueba. Es necesario que el texto al ser codificado y decodificado no pierda información.

2.8. Entregables y evaluación

La tarea debe ser realizada en Python 3.5 o superior. Su entrega debe ser realizada en el directorio `DCCompresor` del repositorio entregado. Este debe incluir, al menos, dos archivos. El primero, llamado `dccompresor.py`, debe implementar el compresor/descompresor de archivos de texto. El segundo archivo, llamado `dccompresor_train.ipynb`, debe contener un Jupyter Notebook donde se implemente y describa el proceso de entrenamiento y construcción del compresor/descompresor. Recuerden que se pueden guardar los pesos finales de un modelo entrenado en un archivo, el cual pueden entregar también si es necesario para que el código de `dccompresor.py` funcione correctamente.

Con respecto a la ejecución del compresor/descompresor, este debe permitir su ejecución a través de la línea de comandos, utilizando parámetros para especificar su modo de ejecución. Específicamente, el sistema deber ser llamado utilizando el siguiente esquema:

- `python dccompresor -c <src> <dst>`: comprime el archivo de texto de nombre `<src>`, generando el archivo comprimido de nombre `<dst>`.
- `python dccompresor -d <src> <dst>`: descomprime el archivo comprimido de nombre `<src>`, generando el archivo de texto de nombre `<dst>`.

Es muy importante que se cumpla esta convención, de lo contrario habrá un descuento considerable.

El archivo a comprimir debe ser un .txt que incluye el texto a codificar con el siguiente formato (esto es un ejemplo del formato):

```
1 well folks, my mac plus finally gave up the ghost this weekend after\nstarting life as a 512
    k way back in 1985.
```

Y el archivo comprimido con el siguiente formato en un archivo .txt.

```
0b111/0b10/0b101/.../0b11/0b1011/0b1001/0b1011
```

Todas las tareas serán evaluadas utilizando el mismo conjunto de archivos de prueba, todos codificados en ASCII.

La entrega de la tarea debe realizarse en la carpeta `DCCompresor` del repositorio en GitHub que será asignado a cada uno. Para fines de corrección, se revisará la última versión subida al repositorio.