# Predictor de Votaciones CR

El objetivo de este software es entrenar y evaluar distintos modelos de clasificación. Para lograrlo se hace uso de un conjunto de `votantes` generados con base en los indicadores cantonales del estado de la nación del 2011 y de los escrutinios de votos para presidencia del 2018.

## Instalación

### Requisitos Previos

**Nota**: Si se trabaja con Windows 7 o posterior, se debe contar con la versión `3.5.x` o `3.6.x` de Python de 64-bit.

1. Abrir la consola de comandos con permisos de administrador.
2. Instalar la versión adecuada de Tensorflow, `cpu` o `gpu` mediante alguno de los siguientes comandos según corresponda:

   > pip install --upgrade tensorflow

   > pip install --upgrade tensorflow-gpu

## Manual de Uso

Para ejecutar cualquiera de los modelos es necesario navegar hasta el directorio `tec\ic\ia\p1`. En dicho directorio se debe abrir una terminal o el símbolo del sistema en caso de tratarse de Windows. El usuario debe ejecutar el archivo `g03.py` usando `Python`. Cada uno de los modelos recibe diferentes parámetros, por lo que a continuación se describe un ejemplo de como ejecutar cada uno de ellos.

Antes de eso se describen los parámetros que todos tienen en común:

**--prefijo** es el nombre del archivo resultante que se plasmará en el documento final con las predicciones.

**--poblacion** es el tamaño de la muestra que se va generar para realizar el entrenamiento, la validación y las pruebas.

**--k-segmentos** indica los k segmentos en que se dividirá el set de entrenamiento, para realizar el proceso de cross-validation.

### Ejecución del árbol de decisión

> --arbol --umbral-poda 0.1 --prefijo arboles --poblacion 1000 --porcentaje-pruebas 10 --k-segmentos 10

Explicación de los párametros **--arbol** es el flag para indicar que se debe usar el modelo de clasificación con árboles de decisión. **--umbral-poda** ajusta el umbral de poda a utilizar con el árbol de decisión.

### Ejecución de la regresión logística

> --regresión-logistica --l1 --prefijo reg --poblacion 1000 --porcentaje-pruebas 10 --k-segmentos 10 --regresión-logistica --l2 --prefijo reg --poblacion 1000 --porcentaje-pruebas 10 --k-segmentos 10

**--l1** activa la regularización l1 **--l2** activa la regularización l2 **Nota:** el incluir ambos comandos anteriores, activa la regularización por defecto del clasificador lineal utilizado. **--normalizacion** normalización para preproceso de los datos, valores posibles [os, ss, fs] (que significan overmax standarization, standard standarization, feature rescaling)

### Ejecución  KNN-KD-TREE

> -- knn -k 5 -prefijo arboles --poblacion 1000 --porcentaje-pruebas 10 --k-segmentos 10

**--k** cantidad de vecinos más cercanos

### Ejecución de la red neuronal

> -- red-neuronal --numero-capas 3 --unidades-por-capa 5 --funcion-activacion relu

**--numero-capas** cantidad de capas de las red **--unidades-por-capa** cantidad de capas de las red **--funcion-activacion** función de activación , valores posibles [relu, softmax, softplus] **--normalizacion** normalización para preproceso de los datos, valores posibles [os, ss, fs] (que significan overmax standarization, standard standarization, feature rescaling)

# Reportes de Métodos Implementados

## Clasificación basada en modelos lineales: Regresión Logística

Este tipo modelo se utiliza para predecir el valor de una variable, cuando dicho valor puede ser uno de un conjunto limitados de opciones. Dicha predicción es realizada con base a un conjunto de atributos de los cuales se espera la variable es dependiente.

Para el presente proyecto se toma como variable dependiente ya sea **el voto para primera ronda** o **el voto para segunda ronda** de una persona según un listado de indicadores cantonales correspondientes a la misma, así como algunos otros atributos personales en su mayoría cualitativos.

## Implementación del modelo

Para la implementación del modelo de regresión logística se hace uso de la biblioteca de *Tensorflow 1.8.0*. En concreto se utiliza un estimador para clasificación lineal, en el que el voto emitido por una persona en una ronda específica está dado por los indicadores del cantón en el que emitió dicho voto. Por lo que se asume que vive en el mismo cantón en el que emite el voto. Los atributos tomados en cuenta para cada predicción son los siguientes:

#### Ronda 1 y Ronda 2

Atributos personales, son atributos categóricos en su mayoría (a excepción del promedio de años de estudio y la edad), lo cual significa que su valor está definido por un conjunto limitado de posibilidades (en este caso `True` o `False` representados por `1` y `0` respectivamente).

> Edad, residencia en zona urbana, género, dependencia económica, estado de su vivienda, condición de hacinamiento, alfabetización, promedio de años que estudió, si se encuentra en educación superior, si trabaja, posee seguro, si es extranjero, discapacidad.

Atributos compartidos entre las personas del mismo cantón, que en su mayoría son atributos numéricos (a excepción del cantón en sí). Sus valores correspondientes son continuos y varían altamente.

> Cantón, población total, superficie, densidad poblacional, viviendas individuales ocupadas, promedio de ocupantes, porcentaje de viviendas con jefatura femenina y porcentaje de jefatura compartida.

**Ronda 2 usando resultados de Ronda 1**

Para la tercera predicción realizada se utiliza el voto emitido por cada persona en **ronda 1** como atributo adicional para predecir su voto en **ronda 2**. Por lo tanto se utilizan todos los atributos anteriores de igual forma.

## Normalización de datos

Se implementó la normalización de los datos ante la existencia de grandes diferencias de rangos para diferentes atributos. Como ejemplo:

- Edad máxima: 65. Edad mínima: 18. (Años)
- Población máxima: ~300 000. Población mínima: ~25 000. (Habitantes)

Para que estas diferencias no sesguen la construcción del modelo, se implementaron diferentes métodos de normalización ya conocidos. La forma de escoger estas para el procesamiento de los datos se encuentra detallada en el manual, como apéndice del documento.

1. El más simple es la división de cada valor por el máximo en su mismo atributo. Denominado *scaling over the max value*. Esta define nuevos valores entre `0` y `1`.
   1.1 Definida por: $x' = \frac{x}{max(x)}$
2. Se encuentra disponible también la normalización estándar. Esta normalización introduce valores negativos `x < 0`, dentro de los datos, además que no define valores entre `0` y `1`.
   2.1 Definida por: $x' = \frac{x-\mu}{\sigma}$
3. Finalmente la re-escala de atributos. Esta normalización no es recomendada si los valores son muy cercanos, pues acentúa las diferencias entre los mismos.
   3.1 Definida por: $x' = \frac{x-min(x)}{max(x)-min(x)}$

## Configuración del modelo

**Taza de aprendizaje**

Uno de los hiperparámetros configurados para la regresión logística del proyecto es el *learning rate* o taza de aprendizaje. El valor escogido con base en pruebas realizadas se definió como `0.03`, con base en las siguientes observaciones.

1. Valores en décimas mostraban efectos negativos en la precisión de los modelos. En ocasiones reduciéndola hasta por 10% o 15%.
2. Valores en milésimas no mostraban cambios significativos en relación a los cambios vistos mediante centésimas.
3. Entre las centésimas, se escoge `0.03` de forma arbitraria, pues los valores cercanos parecen dar resultados no distinguibles ante el ojo no entrenado de los estudiantes.

## Medidas de rendimiento

**Precisión**

La medida principal para cuantificar el desempeño de los modelos implementados fue la precisión. Refiriéndose precisión, a la cantidad de aciertos dividido entre el total de predicciones realizadas.

La precisión tomada en cuenta es la obtenida de la predicción del conjunto de datos para validación, una vez entrenado el modelo. El error de entrenamiento suele tomarse en cuenta únicamente para determinar cuando el modelo ha convergido.

Luego del proceso de K-Fold Cross Validation, se utiliza la precisión de cada modelo en su validación para determinar cual es el más óptimo, para así generar la predicción de todo el conjunto de datos para generar el archivo de salida solicitado en la especificación del proyecto.

## Clasificación basada en redes neuronales

Las redes neuronales tienen gran cantidad de usos, como el reconocimiento de patrones, clasificación de elementos, aproximación de funciones, entre otros más.

Para el presente proyecto se utilizan redes neuronales para predecir un el voto de una persona para primera o segunda ronda de elecciones presidenciales. Esto con base en algunos atributos determinados que pueden tener relación a dicha decisión.

## Implementación del modelo

Para la implementación del modelo de regresión logística se hace uso de la biblioteca de *Tensorflow 1.8.0*. En concreto se utiliza el módulo de Keras. Dicho módulo proporciona diferentes componentes entre estimadores, métricas y muchas otras funcionalidades para el desarrollo de modelos. Al igual que con la regresión logística, se busca predecir el voto de elecciones presidenciales según los atributos personales de un votante, así como algunos relacionados al cantón donde vota. Por lo anterior, se asume que vive en el mismo cantón en el que emite el voto. Los atributos tomados en cuenta para cada predicción son los siguientes:

**Ronda 1 y Ronda 2**

Los atributos categóricos, lo cual significa que su valor está definido por un conjunto limitado de posibilidades (en este caso `True` o `False` representados por `1` y `0` respectivamente), pasaron por un proceso de conversión para su uso apropiado.

Inicialmente estos atributos poseen valores representados con cadenas de texto, pero durante el preprocesamiento, son modificados para seguir las opciones definidas anteriormente. Dicha modificación se realiza pues cada *neurona* y en general la red neuronal, suele operar mediante cálculos matemáticos exclusivamente, lo cual evidentemente requiere de números y no cadenas de texto. Los atributos utilizados son:

> Edad, residencia en zona urbana, género, dependencia económica, estado de su vivienda, condición de hacinamiento, alfabetización, promedio de años que estudió, si se encuentra en educación superior, si trabaja, posee seguro, si es extranjero, discapacidad, población total, superficie, densidad poblacional, viviendas individuales ocupadas, promedio de ocupantes, porcentaje de viviendas con jefatura femenina y porcentaje de jefatura compartida.

**Ronda 2 usando resultados de Ronda 1**

Para la tercera predicción realizada se utiliza el voto emitido por cada persona en **ronda 1** como atributo adicional para predecir su voto en **ronda 2**. Por lo tanto se utilizan todos los atributos anteriores de igual forma.

### Normalización de datos

Se implementó la normalización de los datos ante la existencia de grandes diferencias de rangos para diferentes atributos. Como ejemplo:

- Edad máxima: 65. Edad mínima: 18. (Años)
- Población máxima: ~300 000. Población mínima: ~25 000. (Habitantes)

Para que estas diferencias no sesguen la construcción del modelo, se implementaron diferentes métodos de normalización ya conocidos. La forma de escoger estas para el procesamiento de los datos se encuentra detallada en el manual, como apéndice del documento.

1. El más simple es la división de cada valor por el máximo en su mismo atributo. Denominado *scaling over the max value*. Esta define nuevos valores entre `0` y `1`.
   1.1 Definida por: $x' = \frac{x}{max(x)}$
2. Se encuentra disponible también la normalización estándar. Esta normalización introduce valores negativos `x < 0`, dentro de los datos, además que no define valores entre `0` y `1`.
   2.1 Definida por: $x' = \frac{x-\mu}{\sigma}$
3. Finalmente la re-escala de atributos. Esta normalización no es recomendada si los valores son muy cercanos, pues acentúa las diferencias entre los mismos.
   3.1 Definida por: $x' = \frac{x-min(x)}{max(x)-min(x)}$

## Configuración del modelo

### Taza de aprendizaje

Uno de los hiperparámetros configurados para la red neuronal al igual que para la regresión logística, es el *learning rate* o taza de aprendizaje. El valor escogido con base en pruebas realizadas se definió como `0.03`, con base en las siguientes observaciones.

1. Valores en décimas mostraban efectos negativos en la precisión de los modelos. En ocasiones reduciéndola hasta por 10% o 15%.
2. Valores en milésimas no mostraban cambios significativos en relación a los cambios vistos mediante centésimas.
3. Entre las centésimas, se escoge `0.03` de forma arbitraria, pues los valores cercanos parecen dar resultados no distinguibles ante el ojo no entrenado de los estudiantes.

### Epochs

Como definición general, un *epoch* es una iteración sobre **todos** los ejemplos de una muestra de datos durante el entrenamiento. Las redes neuronales suelen requerir de múltiples *"pasadas"* o iteraciones sobre los datos durante el entrenamiento, antes de que el modelo pueda converger. Luego de las pruebas realizadas se determinó que se cumplen 50 epochs para que la mayoría de modelos alcancen una convergencia suficiente según el criterio de los estudiantes.

Cabe destacar que para algunos modelos se necesitan más de 50 epochs, en ocasiones 70 epochs suelen ser suficientes para que la red alcance el pico de su desempeño, pero las condiciones para que dichos escenarios se den de forma consistente no están claras. Para la predicción de **segunda ronda**, utilizando como atributo el voto de **primera ronda**, 20 epochs suelen ser suficientes para que el modelo alcance un estado cercano a su pico. Por lo tanto, por motivos principalmente de tiempo de ejecución se definen los epochs en una cantidad media. Según lo observado se concluye que la cantidad de epochs depende en gran medida de la aleatoriedad del simulador de datos de votantes.

## Medidas de rendimiento

#### Precisión

La medida principal para cuantificar el desempeño de los modelos implementados fue la precisión. Refiriéndose precisión, a la cantidad de aciertos dividido entre el total de predicciones realizadas. Al igual que con la regresión logística.

La precisión tomada en cuenta es la obtenida de la predicción del conjunto de datos para validación, una vez entrenado el modelo. El error de entrenamiento suele tomarse en cuenta únicamente para determinar cuando el modelo ha convergido.

Luego del proceso de K-Fold Cross Validation, se utiliza la precisión de cada modelo en su validación para determinar cual es el más óptimo, para así generar la predicción de todo el conjunto de datos para generar el archivo de salida solicitado en la especificación del proyecto.

#### Pérdida

Si la precisión no es suficiente para separar un modelo de otro, pues en ocasiones más de una red maneja la misma precisión posterior a la validación, se utiliza entonces el criterio de la pérdida durante el entrenamiento. Particularme se utiliza una función de entropía cruzada, disponible entre los elementos que brinda Tensorflow.

Así entonces se definen los desempates entre redes neuronales de igual precisión durante la validación. Una vez escogido el modelo de red que maximiza la precisión y minimiza la pérdida, se utiliza el mismo para la predicción del conjunto de datos de prueba aparte que se reservó.

## Resultados

Dado que cada ejecución del programa genera una simulación de muestra de datos aleatoria, los resultados obtenidos no pueden reproducirse ni siquiera al copiar el comando utilizado para dicha ejecución.

### 1

#### Parámetros de ejecución

```
python g03.py --red-neuronal --prefijo red_N --poblacion 1000 --numero-capas 4 --unidades-por-capa 6 --funcion-activacion relu
```

Enlace al archivo de salida.

#### Salida de consola

> Duración de generar_muestra_pais: 1.127063512802124
>
> Prediciendo: r1
>
> ...
>
> Prediciendo: r2
>
> Precisión de cada subset:
>
> Subset 0: 0.5833333333333334
>
> Subset 1: 0.6555555555555556
>
> Subset 2: 0.5833333333333334

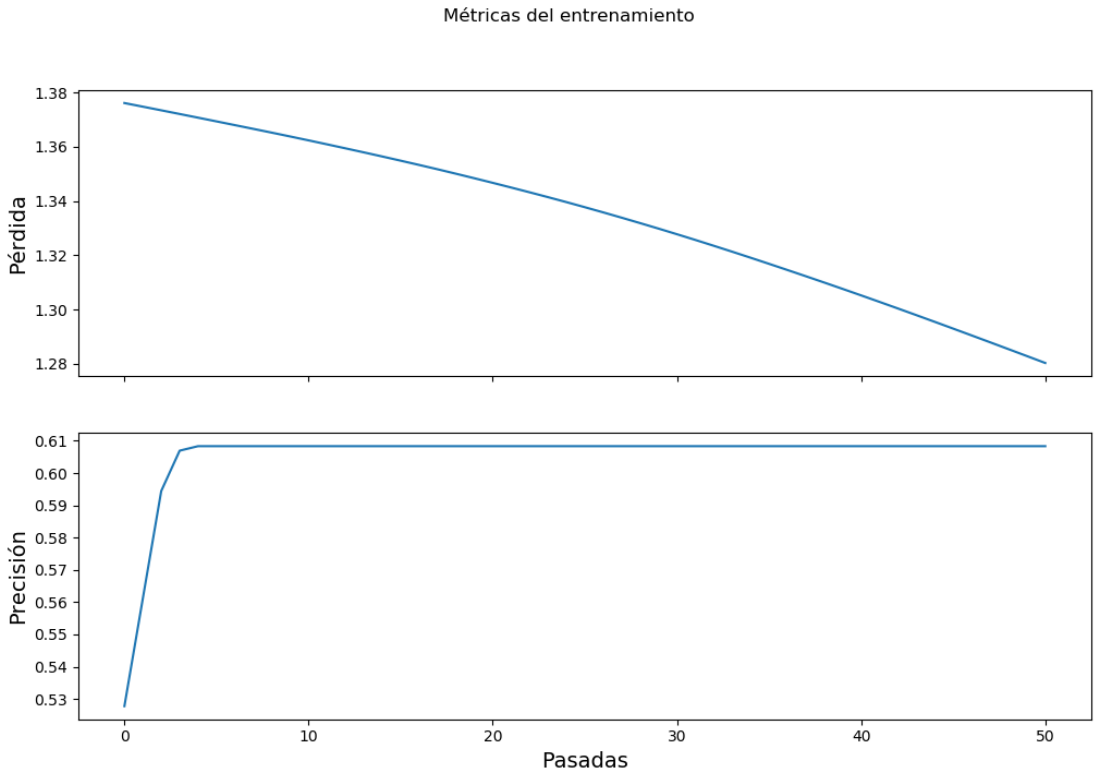
> Subset 3: 0.6
>
> Subset 4: 0.5611111111111111
>
> Pérdida de cada subset:
>
> Subset 0: 1.28
>
> Subset 1: 1.317
>
> ...
>
> Precisión para el set de pruebas aparte: 0.61
>
> Prediciendo: r2_with_r1
>
> Precisión de cada subset:
>
> Subset 0: 0.9833333333333333
>
> Subset 1: 0.9333333333333333
>
> Subset 2: 1.0
>
> Subset 3: 0.9833333333333333
>
> Subset 4: 0.9833333333333333
>
> Pérdida de cada subset:
>
> ... Precisión para el set de pruebas aparte: 0.99

Enlace al archivo con la salida completa.

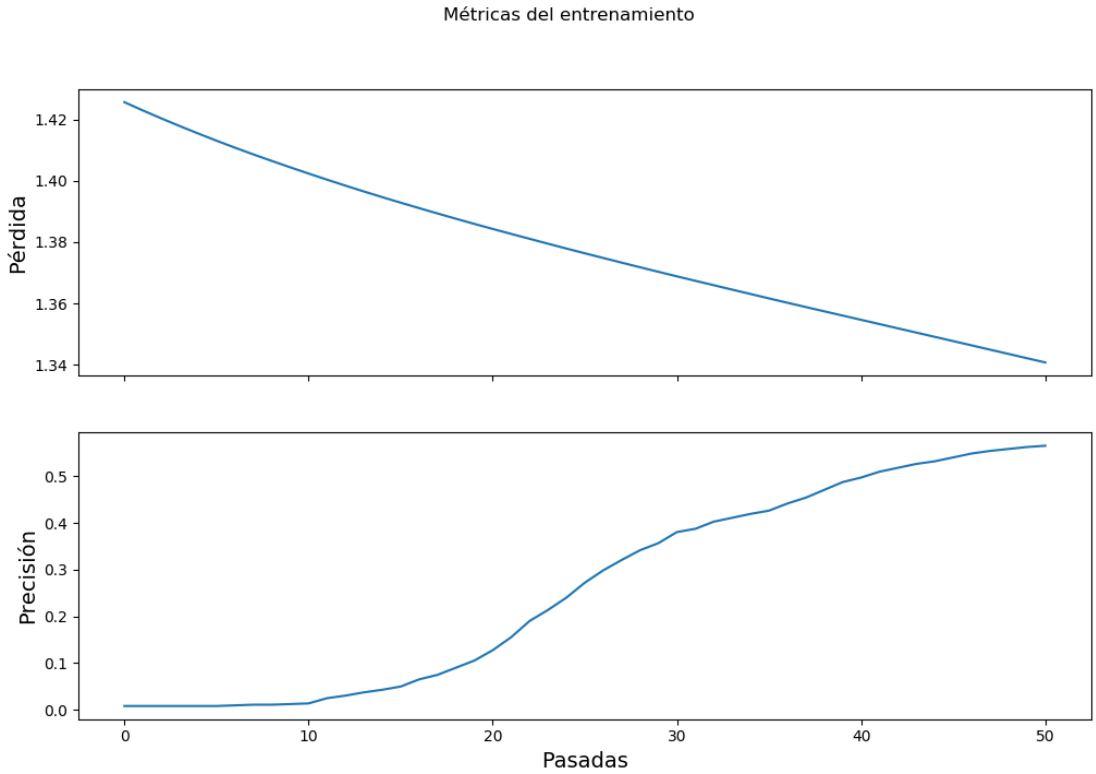
Algunas gráficas que muestran la optimización de la red durante el entrenamiento.

**Prediciendo Ronda 2**

Subset 0:

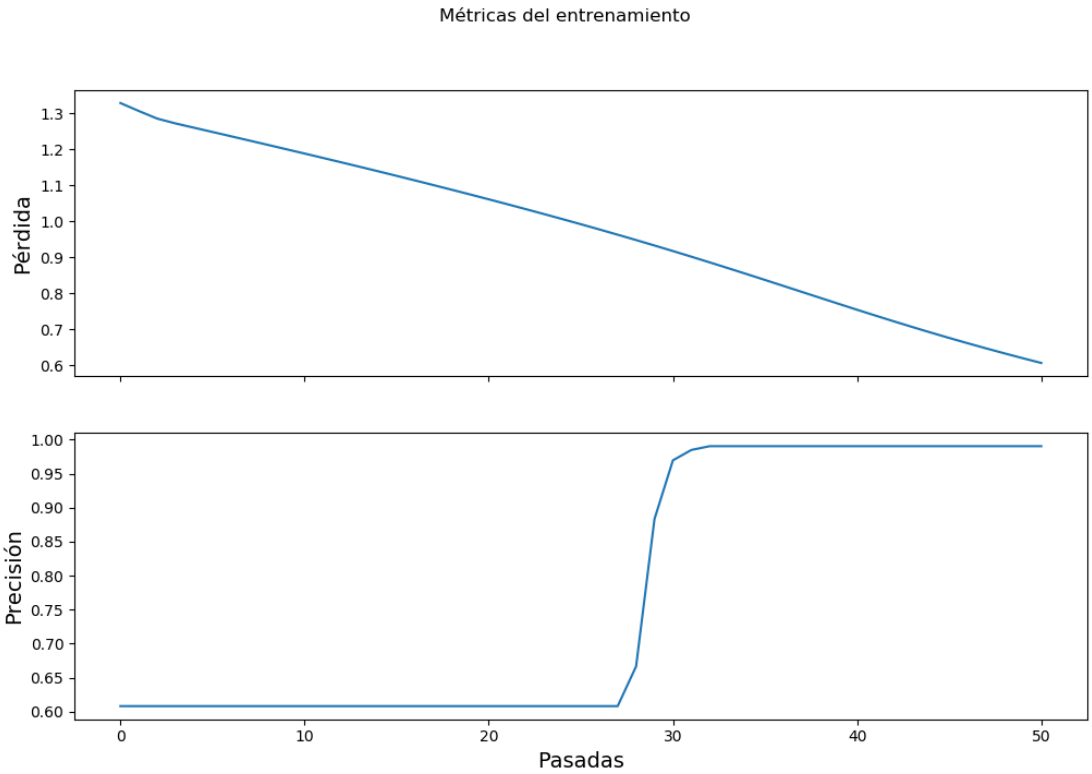



Subset 3:

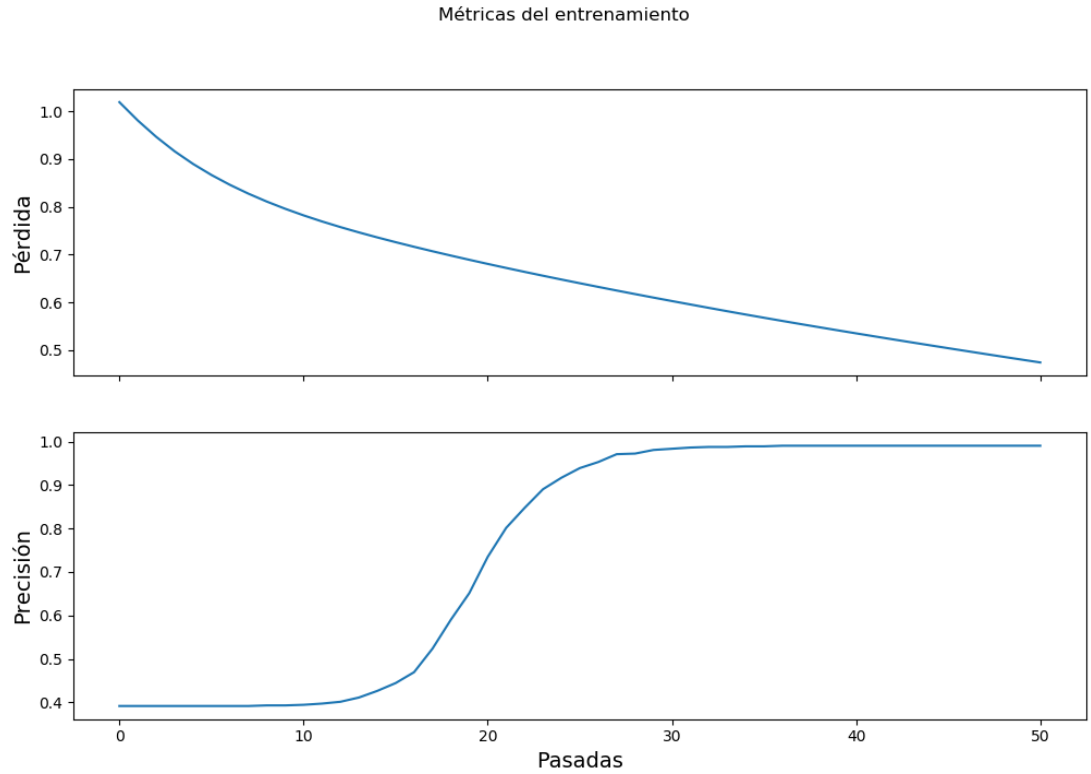



**Prediciendo Ronda 2 con Ronda 1 como atributo**

Subset 0:




Subset 3:




De lo anterior se puede apreciar, que como se esperaba, el agregar el voto de **ronda 1** como atributo al modelo, incrementa en gran medida el desempeño del mismo.

## 2

**Parámetros de ejecución**

```
python g03.py --red-neuronal --prefijo red_N --poblacion 1000 --numero-capas 8 --unidades-por-capa 12 --funcion-activacion softmax
```

Enlace al archivo de salida.

**Salida de consola**

> ...
>
> Prediciendo: r1
>
> Precisión de cada subset:
>
> Subset 0: 0.6722222222222223
>
> Subset 1: 0.0
>
> Subset 2: 0.39444444444444443
>
> Subset 3: 0.0
>
> Subset 4: 0.3333333333333333
>
> ...
>
> Precisión para el set de pruebas aparte: 0.65
>
> Prediciendo: r2
>
> Precisión de cada subset:
>
> Subset 0: 0.6722222222222223
>
> Subset 1: 0.4444444444444444
>
> Subset 2: 0.6
>
> Subset 3: 0.5833333333333334
>
> Subset 4: 0.6444444444444445
>
> ...
>
> Precisión para el set de pruebas aparte: 0.65
>
> Prediciendo: r2_with_r1
>
> Precisión de cada subset:
>
> Subset 0: 0.6722222222222223
>
> Subset 1: 0.5333333333333333
>
> Subset 2: 0.6
>
> Subset 3: 0.5833333333333334
>
> Subset 4: 0.6444444444444445

> ...
>
> Precisión para el set de pruebas aparte: 0.65

[Enlace al archivo con la salida completa.]

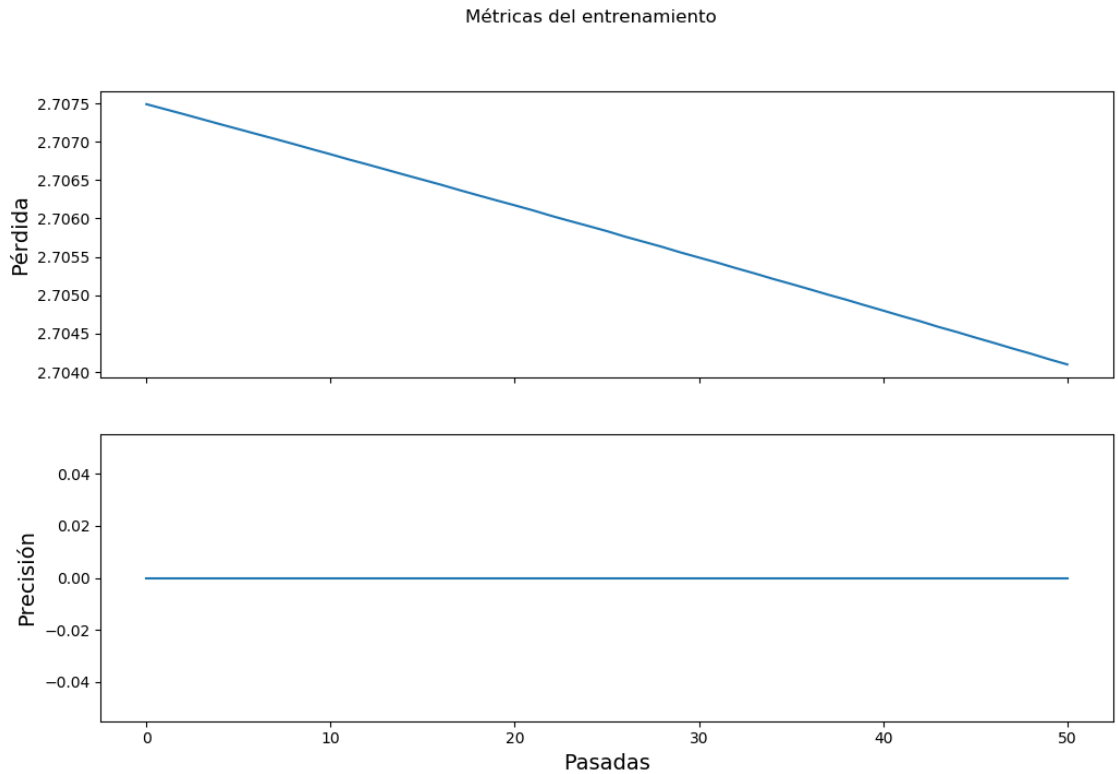
Tómese en cuenta que con respecto a las primeras pruebas realizadas, se duplicó la cantidad de capas, así como unidades por capa, y al mismo tiempo se cambió la función de activación para las unidades. Estos cambios en los parámetros tuvieron efecto en la optimización de forma muy notoria, según la experiencia, el cambio de función de activación **relu** hacia **softmax** provoca los modelos usualmente alcancen su pico de desempeño de forma abrupta. En ocasiones esto significa también que si el pico del modelo es cercano a cero por ciento de precisión, el modelo se mantiene estancado sin mejorar.

Por otra parte, la influencia de la cantidad de capas y unidades por capa no queda clara hasta el momento. Por lo que se incluye una tercera prueba.

Para consultar las gráficas de cada subset: Gráficas

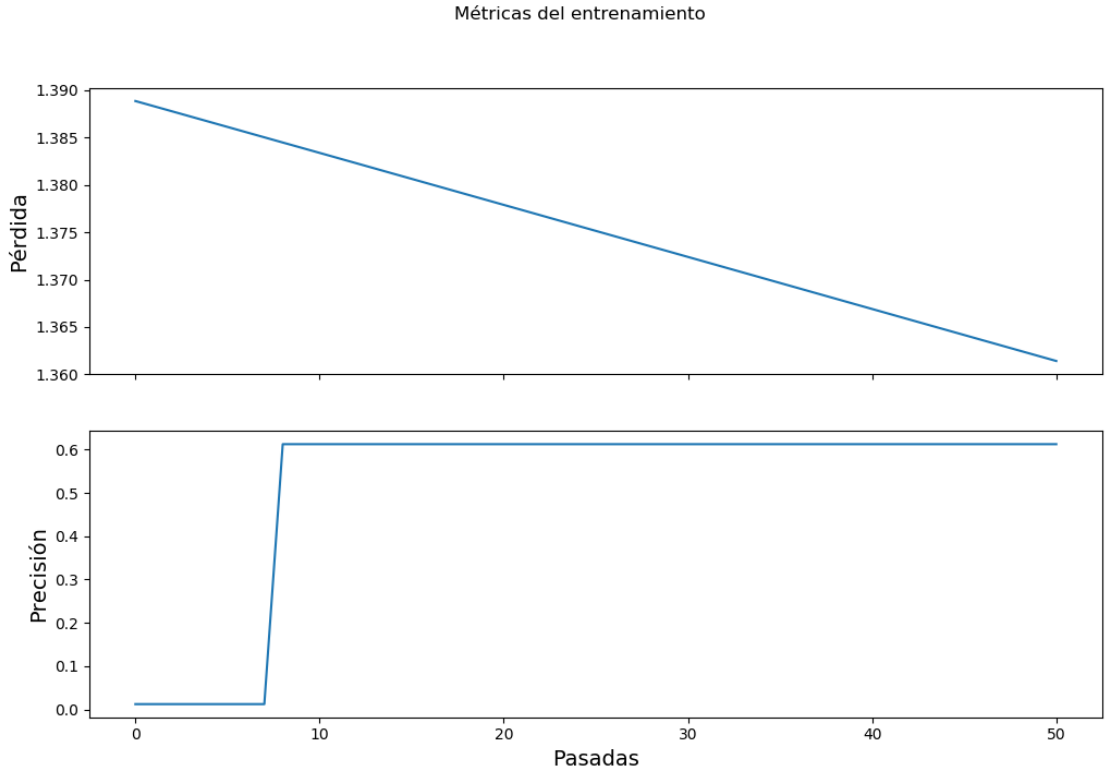
**Prediciendo Ronda 1**

Subset 3:



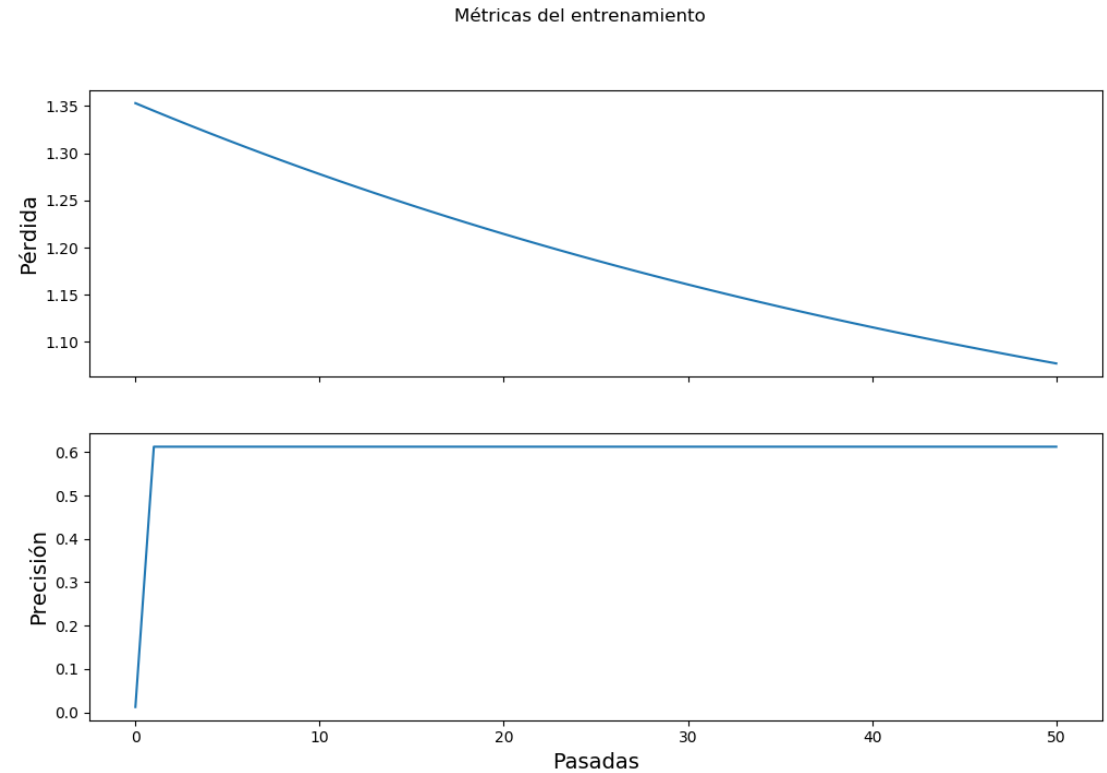

**Prediciendo Ronda 2**

Subset 3:




**Prediciendo Ronda 2 con Ronda 1 como atributo**

Subset 3:




Las gráficas anteriores muestran la mejora en la minimización de la cantidad de epochs necesarios para que el modelo converja. Esto probablemente está relacionado a los atributos utilizados por cada modelo, así como a la progresiva disminución de etiquetas disponibles.

## 3

Esta tercera ejecución tiene como objetivo poderse comparar con la ejecución 1. Esto pues comparten la misma función de activación **relu**, pero la cantidad de capas y de unidades por capa se aumentó en 10 con respecto a dicha prueba 1.

**Parámetros de ejecución**

```
python g03.py --red-neuronal --prefijo red_N --poblacion 1000 --numero-capas 14 --unidades-por-capa 16 --funcion-activacion relu
```

Enlace al archivo de salida.

**Salida de consola**

> Prediciendo: r1
>
> Precisión de cada subset:
>
> Subset 0: 0.43
>
> Subset 1: 0.5966666666666667
>
> Subset 2: 0.56
>
> ...
>
> Precisión para el set de pruebas aparte: 0.54
>
> ...
>
> Prediciendo: r2
>
> Precisión de cada subset:
>
> Subset 0: 0.5766666666666667
>
> Subset 1: 0.5966666666666667
>
> Subset 2: 0.55
>
> ...
>
> Precisión para el set de pruebas aparte: 0.54
>
> ...
>
> Prediciendo: r2_with_r1
>
> ...
>
> Precisión de cada subset:
>
> Subset 0: 0.5766666666666667
>
> Subset 1: 0.9833333333333333
>
> Subset 2: 0.56
>
> ...

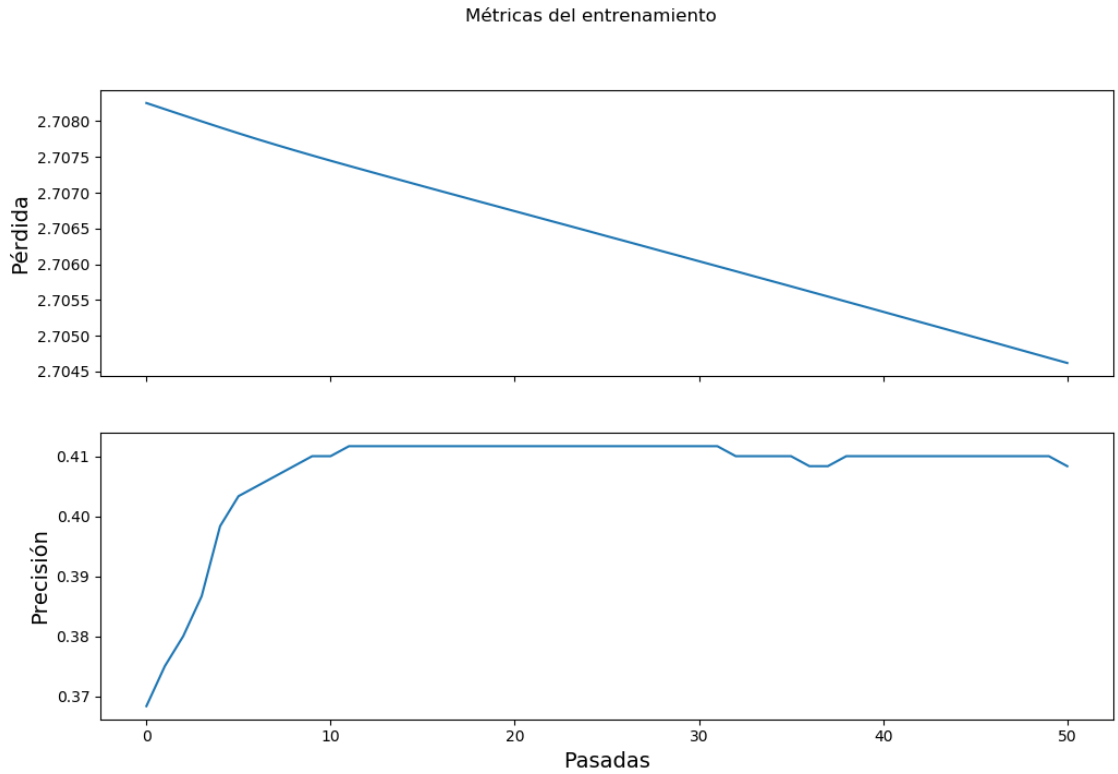
Enlace al archivo con la salida completa.

Con base en las salidas de consola, parece ser que no existe mucha diferencia al cambiar tan drásticamente la estructura de los modelos. Pero al observar las gráficas se nota una evidente distorsión en la convergencia y estabilidad de cada modelo.

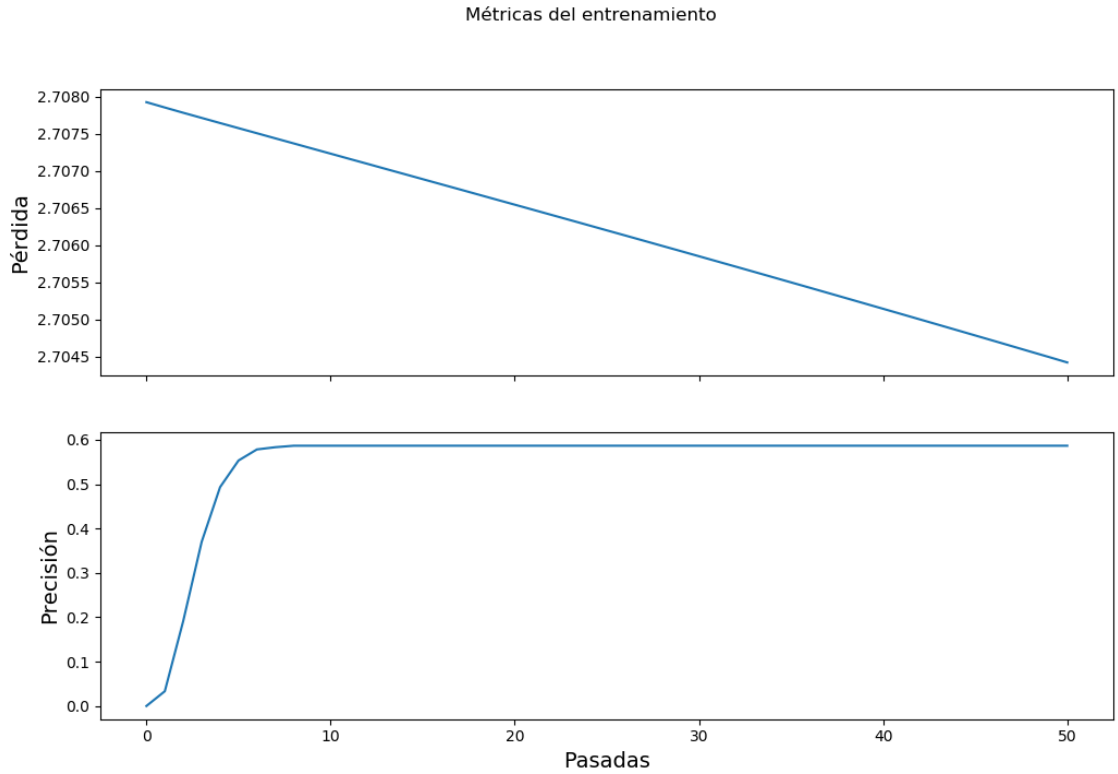
Para consultar las gráficas de cada subset: Gráficas

**Prediciendo Ronda 1**

Subset 0:

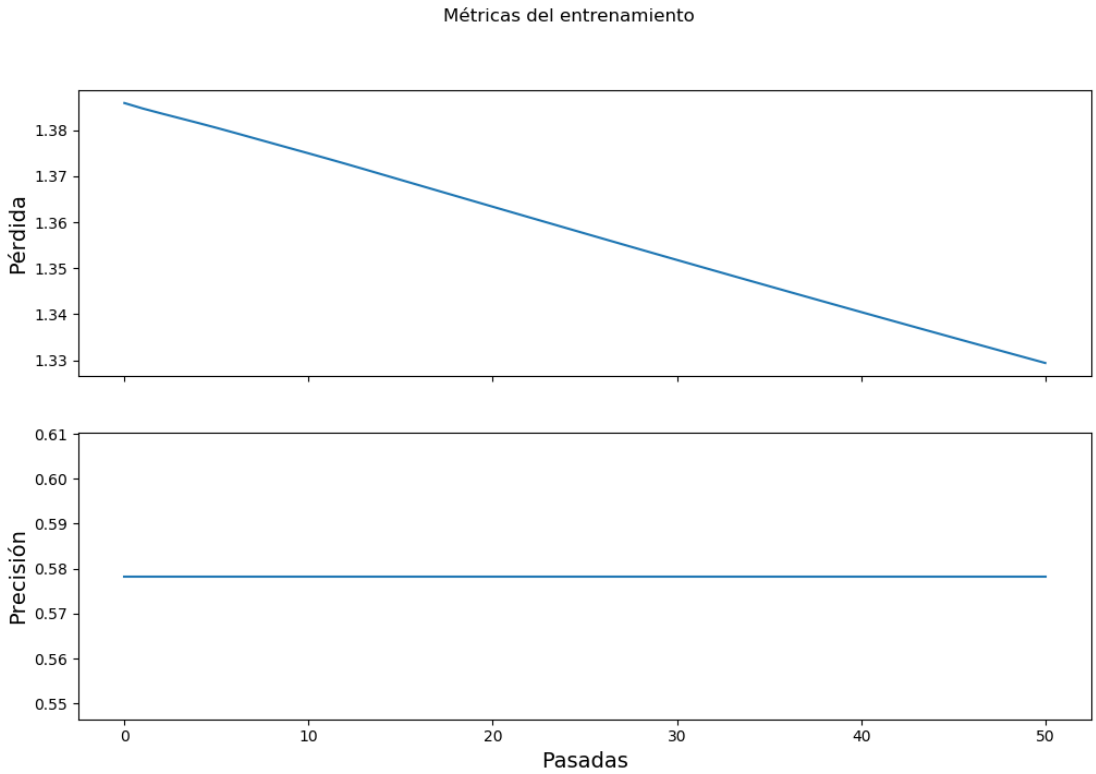



Subset 2:

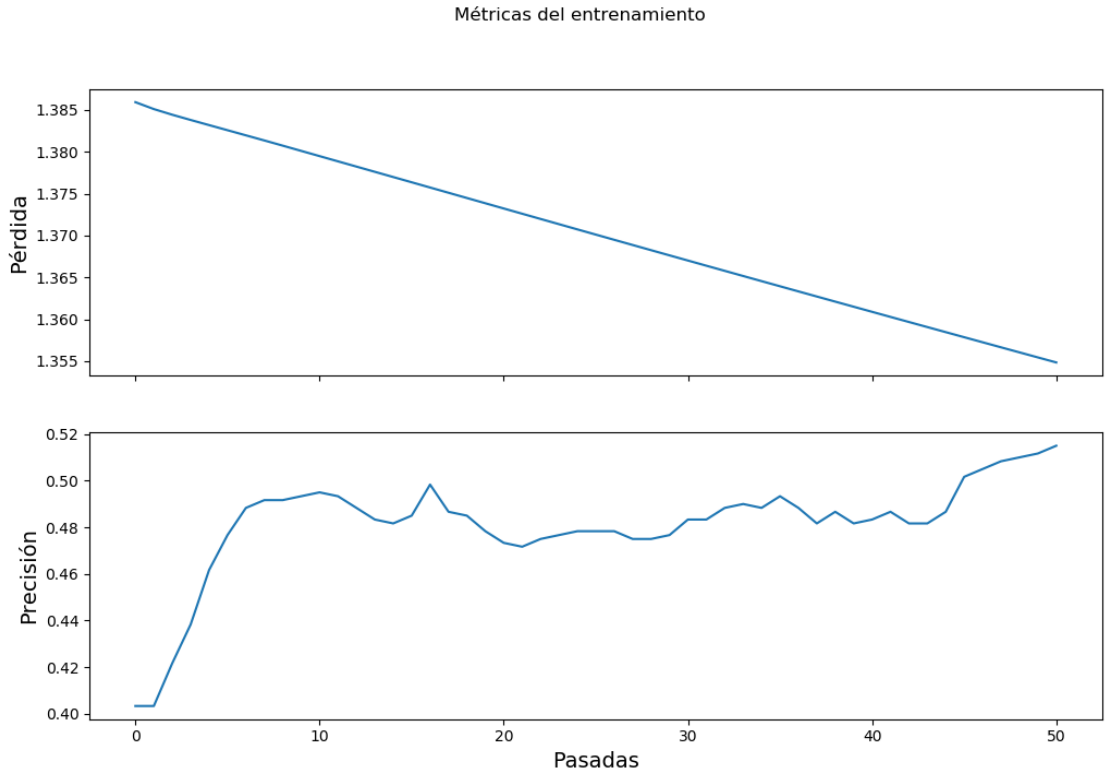



**Prediciendo Ronda 2**

Subset 0:

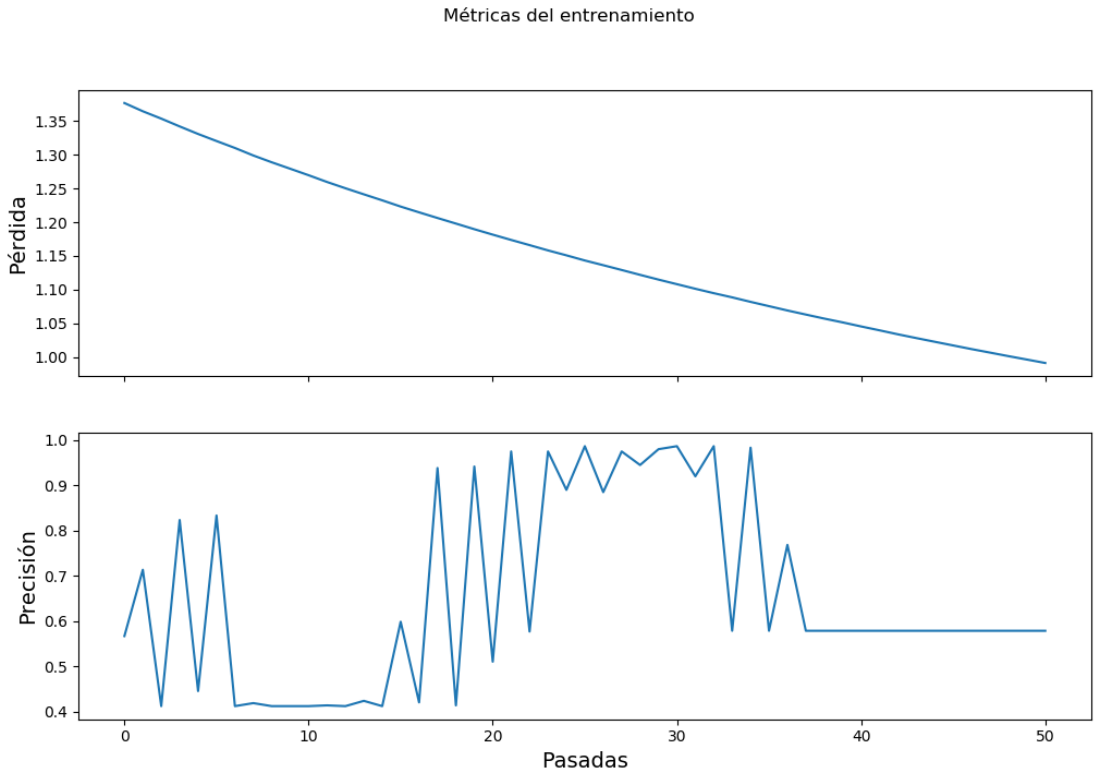



Subset 2:

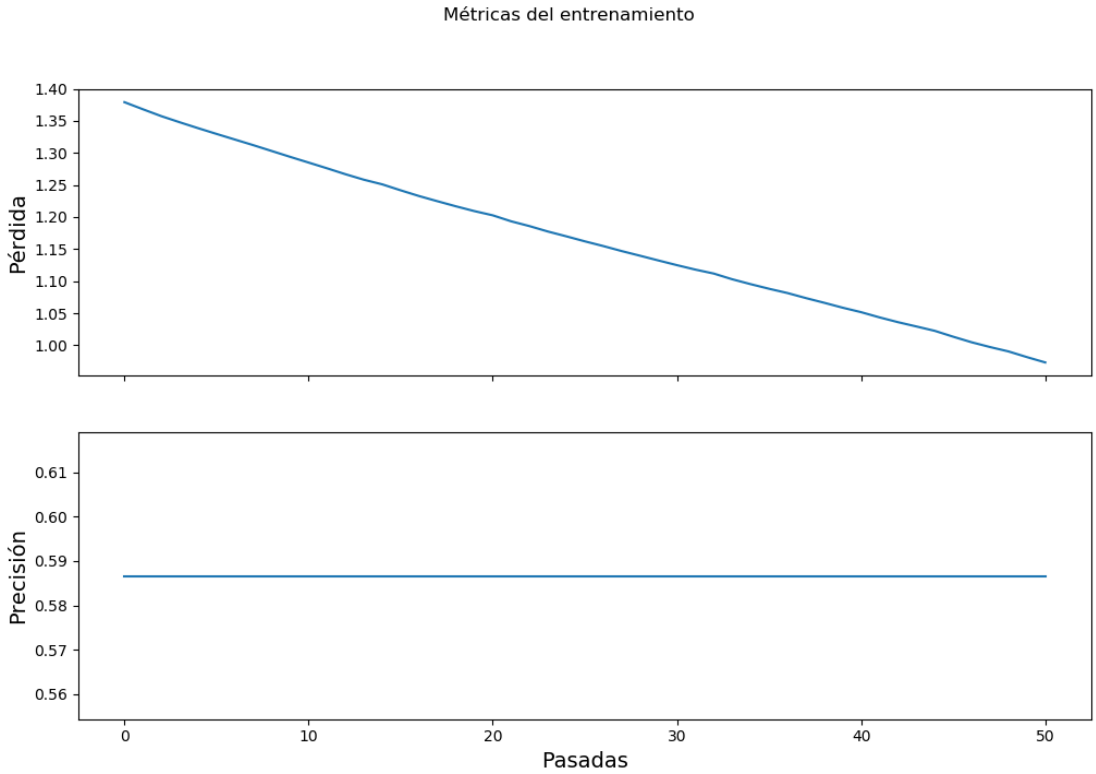



**Prediciendo Ronda 2 con Ronda 1 como atributo**

Subset 0:




Subset 2:




Considerando las gráficas anteriores, dado que la función de activación es la misma que en la ejecución 1, se concluye que el aumento considerable de la cantidad de capas, así como la cantidad de unidades por capa, afecta la estabilidad del proceso de convergencia de los modelos, pues no se logra identificar un patrón claro como en otras ocasiones. Esto podría relacionarse con el principio de que los modelos excesivamente complejos tienden a no ser los óptimos.

# Clasificación basada árboles de decisión

Antes de abordar el tema del modelo basado en árboles de decisión, es fundamental tener una perspectiva clara de lo que implica dicho modelo. Un árbol de decisión toma como entrada un objeto o una situación descrita a través de un conjunto de atributos y devuelve una **decisión**: el valor previsto de la salida dada la entrada.

- Los atributos de entrada pueden ser discretos o continuos.
- El valor de la salida puede ser a su vez discreto o continuo

Además, se debe tener claro que aprender una función de valores discretos se denomina clasificación y aprender una función continua se denomina regresión. En este caso las clasificaciones no son booleanas, ya que dicha clasificación puede tomar una de las 13 etiquetas que representan a los partidos políticos. Algunas de las características mencionadas en el libro de IA, A Modern Approach son las que se mencionan a continuación:

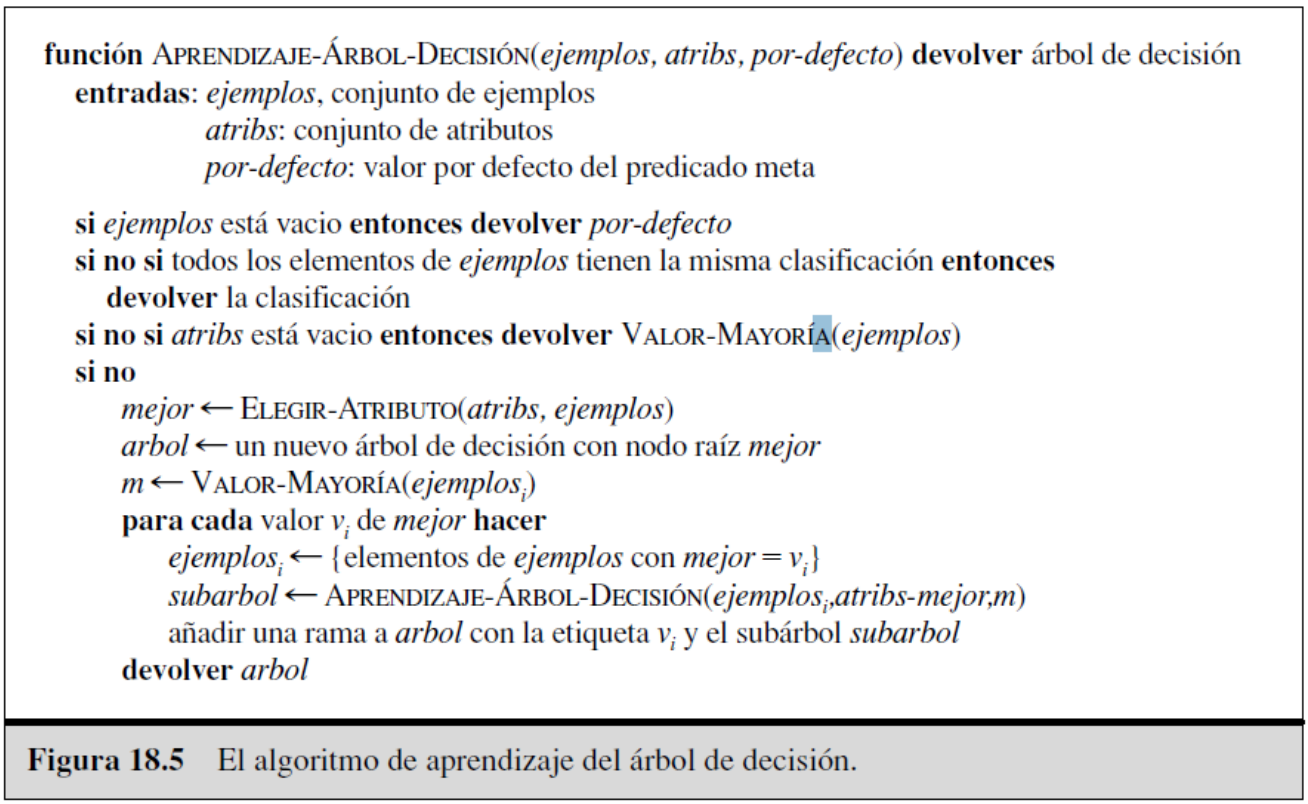
- Un árbol de decisión desarrolla una secuencia de test para poder alcanzar una decisión.
- Cada nodo interno del árbol corresponde con un test sobre el valor de una de las propiedades, y las ramas que salen del nodo están etiquetadas con los posibles valores de dicha propiedad.
- Cada nodo hoja del árbol representa el valor que ha de ser devuelto si dicho nodo hoja es alcanzado.

---

## Algoritmo de aprendizaje

Ahora dejando de lado la teoría, se procede a enfocar la atención en la sección del algoritmo que se debe emplear, este se muestra en la siguiente imagen, tomada del libro IA, A Modern Approach.

## Algoritmo de aprendizaje del árbol de decisión



**función** APRENDIZAJE-ÁRBOL-DECISIÓN(*ejemplos*, *atribs*, *por-defecto*) **devolver** árbol de decisión
  **entradas**: *ejemplos*, conjunto de ejemplos
    *atribs*: conjunto de atributos
    *por-defecto*: valor por defecto del predicado meta

  **si** *ejemplos* está vacio **entonces devolver** *por-defecto*
  **si no si** todos los elementos de *ejemplos* tienen la misma clasificación **entonces**
    **devolver** la clasificación
  **si no si** *atribs* está vacio **entonces devolver** VALOR-MAYORÍA(*ejemplos*)
  **si no**
    *mejor* ← ELEGIR-ATRIBUTO(*atribs*, *ejemplos*)
    *arbol* ← un nuevo árbol de decisión con nodo raíz *mejor*
    *m* ← VALOR-MAYORÍA($ejemplos_i$)
    **para cada** valor $v_i$ de *mejor* **hacer**
      $ejemplos_i$ ← {elementos de *ejemplos* con *mejor* = $v_i$}
      *subarbol* ← APRENDIZAJE-ÁRBOL-DECISIÓN($ejemplos_i$,*atribs-mejor*,*m*)
      añadir una rama a *arbol* con la etiqueta $v_i$ y el subárbol *subarbol*
    **devolver** *arbol*

**Figura 18.5** El algoritmo de aprendizaje del árbol de decisión.

---

## Fórmulas empleadas

Otro aspecto importante que se debe conocer, es la generación de cálculos matemáticos que nos ayudarán a calcular la ganancia de información que tiene un determinado atributo, y con ello poder decidir sobre cuál de todos hacer una determinada **bifurcación** en el proceso de entrenamiento del modelo de árbol de decisión.

## Entropía

El primer cálculo a tener en cuenta es la entropía, que indica el grado de incertidumbre que posee un cierto atributo. En este caso se implementa para determinar tanto la incertidumbre del set de datos o muestras, y también para el atributo específico sobre el que se quiere bifurcar. A continuación se muestra dicha fórmula:

$$Entropia(s) = /sum_{i=1}^{n} - p_i log_2 p_i$$

Donde:

- **S**: es una colección de objetos
- **Pi** : es la probabilidad de los posibles valores
- **i**: las posibles respuestas de los objetos

## Ganancia de Información

Por otra parte, se encuentra la formula de ganancia, la cual hace uso del **Resto** y la **Entropía**. Esta consiste básicamente en restar la entropía del set de datos con el resto obtenido del atributo A, obteniendo así el valor de ganancia de dicho atributo. A continuación la fórmula:

$$Ganancia(A) = I(/fracpp + n, /fracnp + n) - Resto(A)$$

## Resto

Por último, se encuentra la fórmula Resto, traducida del inglés **Remainder**.

$$Resto(A) = /sum_{i=1}^{v} /fracp_i + n_i p + nI(/fracp_i p_i + n_i, /fracn_i p_i + n_i)$$

---

## Pasos llevados a cabo

Respecto a la clasificación basada en DTs (Decision Trees), la misma se resume en una serie de pasos que se mencionan a continuación:

- Generar el conjunto de muestras, que en este caso se obtienen del simulador de votantes que se empleó en el proyecto corto 1 (PC1). Esto por medio de los import:

> from tec.ic.ia.pc1.g03 import generar_muestra_pais from tec.ic.ia.pc1.g03 import generar_muestra_provincia

- Luego de ello, por medio de los parámetros recibos, se deben ajustar las muestras de entrenamiento, validación y pruebas. Es decir, crear los subconjuntos a partir de las muestras generadas con el simulador.
- Por último, solamente quedará por ejecutar todo el proceso de entrenamiento, validación y prueba, para cada predicción a generar.

**Nota:** Dichas predicciones son las solicitadas en el proyecto, y que deben ser generadas por cada modelo, plasmando el resultado de cada predicción en un archivo final con exensión **.csv** Estas predicciones se mencionan a continuación:

- Predicción de Ronda #1
- Predicción de Ronda #2
- Predicción de Ronda #2 con Ronda#1
- Y por último, una etiqueta la cual no representa una predicción, pero es una clasificación para indicar si la muestra fue tomada para entrenamiento o en el proceso de pruebas.

---

## Resultados obtenidos

#### Ejecución de ejemplo #1

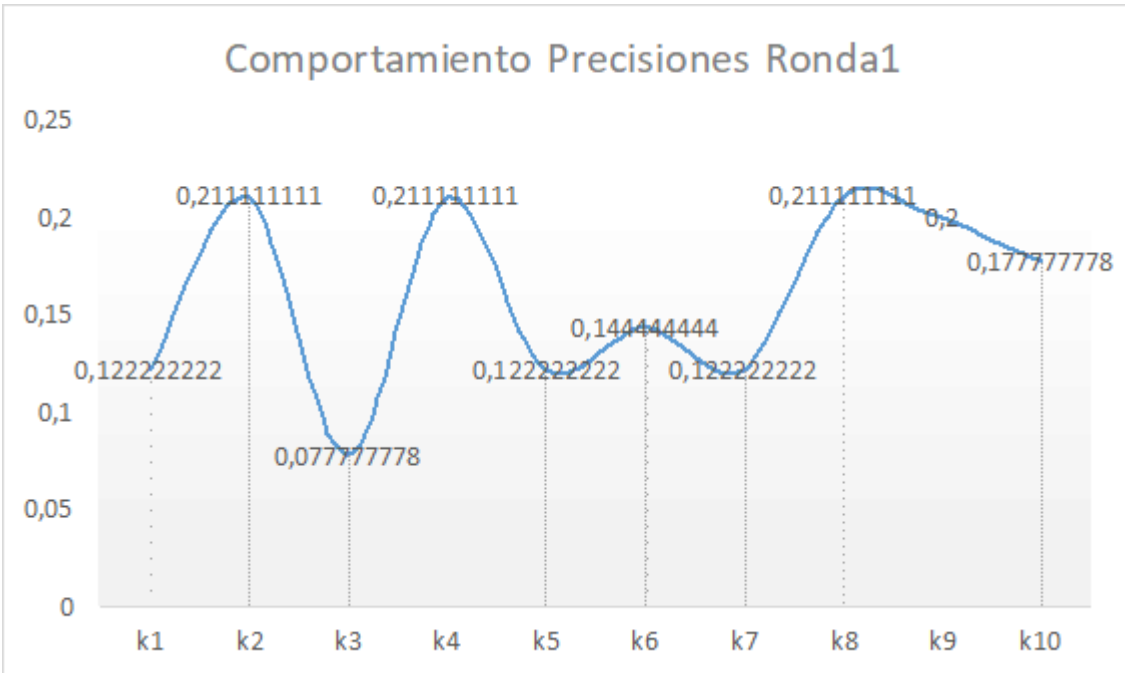
Conjunto de parámetros utilizados en la ejecución:

> --arbol --umbral-poda 0.2 --prefijo arbol_ex1 --poblacion 1000 --porcentaje-pruebas 10 --k-segmentos 10

**Nota:** Se debe tener claro que se utilizará el árbol con mejor precisión según el resultado arrojado por medio del proceso de cross-validation que se lleva a cabo en el entrenamiento.

#### Precisiones obtenidas en la Ronda#1

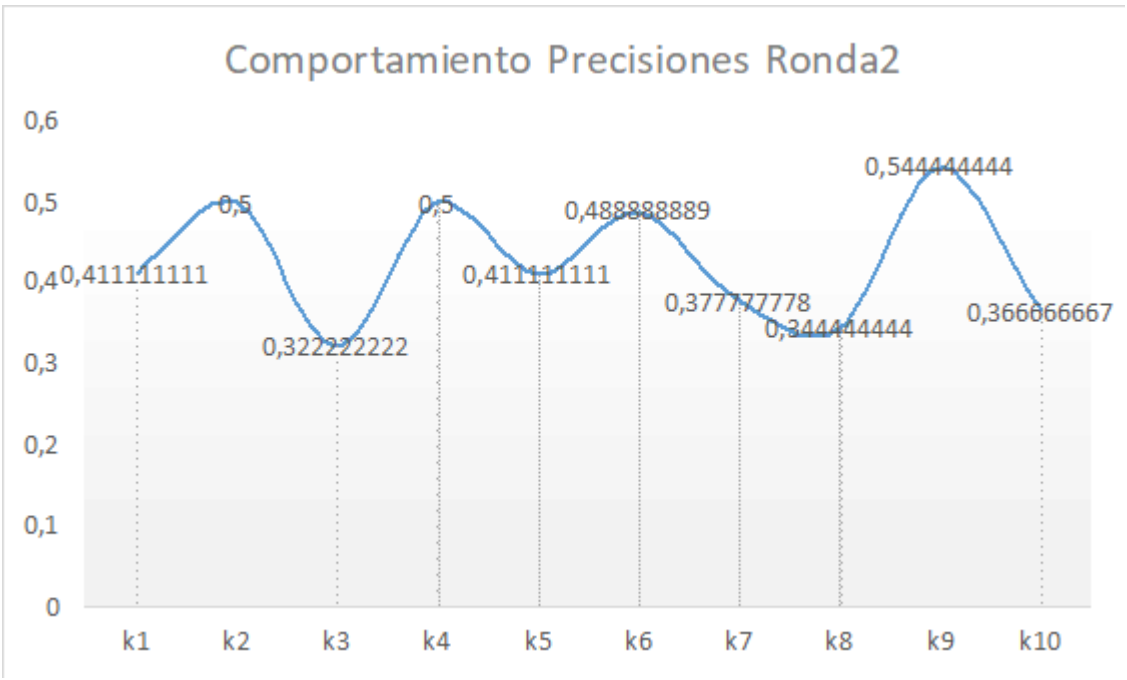
- **Mejor Precisión:** 21.11%
- **Peor Precisión:** 12.22%
- Gráfico de comportamiento:
  Lo que indica es el valor de precisión que se obtuvo por cada grupo K, en el cross validation (K-Fold)




#### Precisiones obtenidas en la Ronda#2

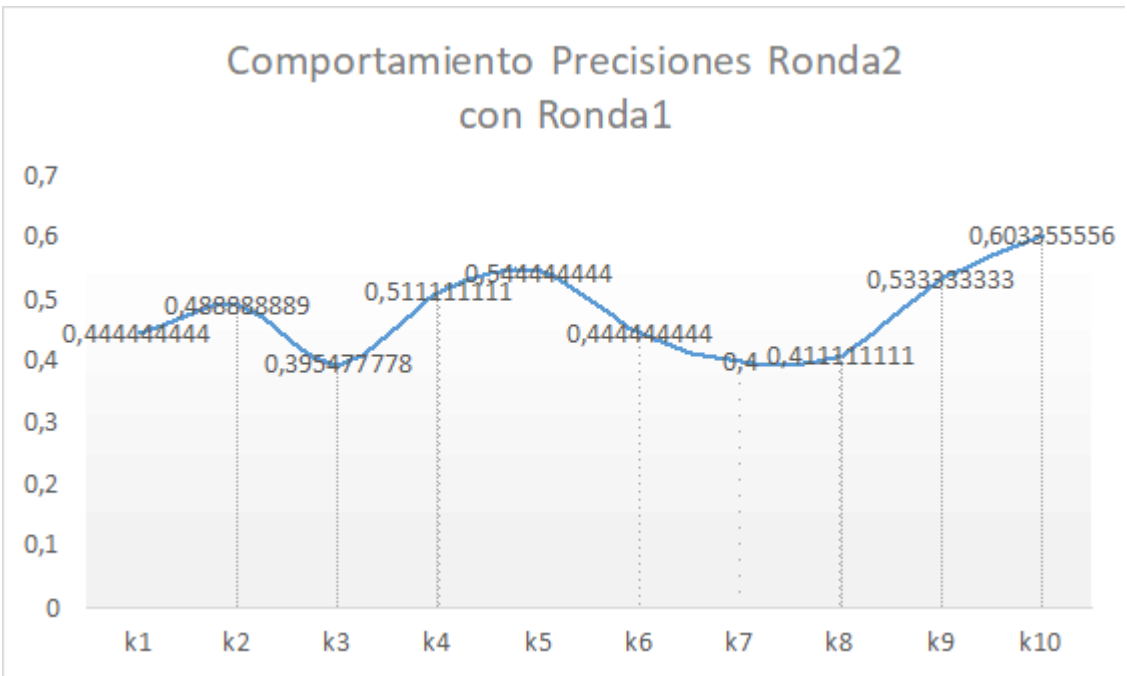
- **Mejor Precisión:** 54.44%

- **Peor Precisión:** 32.22%
- Gráfico de comportamiento:
  Lo que indica es el valor de precisión que se obtuvo por cada grupo K, en el cross validation (K-Fold)




**Precisiones obtenidas en la Ronda#2 con Ronda#1**

- **Mejor Precisión:** 60.33%
- **Peor Precisión:** 39.54%
- Gráfico de comportamiento:
  Lo que indica es el valor de precisión que se obtuvo por cada grupo K, en el cross validation (K-Fold)



**Ejecución de ejemplo #2**

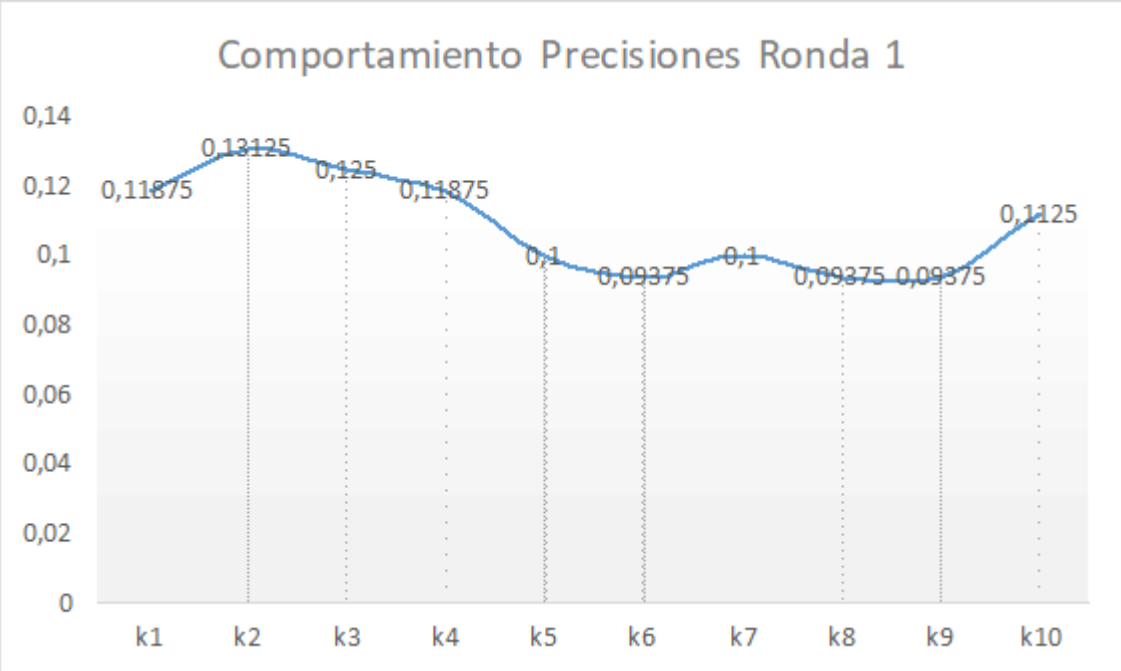
Conjunto de parámetros utilizados en la ejecución:

> --arbol --umbral-poda 0.1 --prefijo arbol_ex2 --poblacion 2000 --porcentaje-pruebas 20 --k-segmentos 10

**Nota:** Se debe tener claro que se utilizará el árbol con mejor precisión según el resultado arrojado por medio del proceso de cross-validation que se lleva a cabo en el entrenamiento.

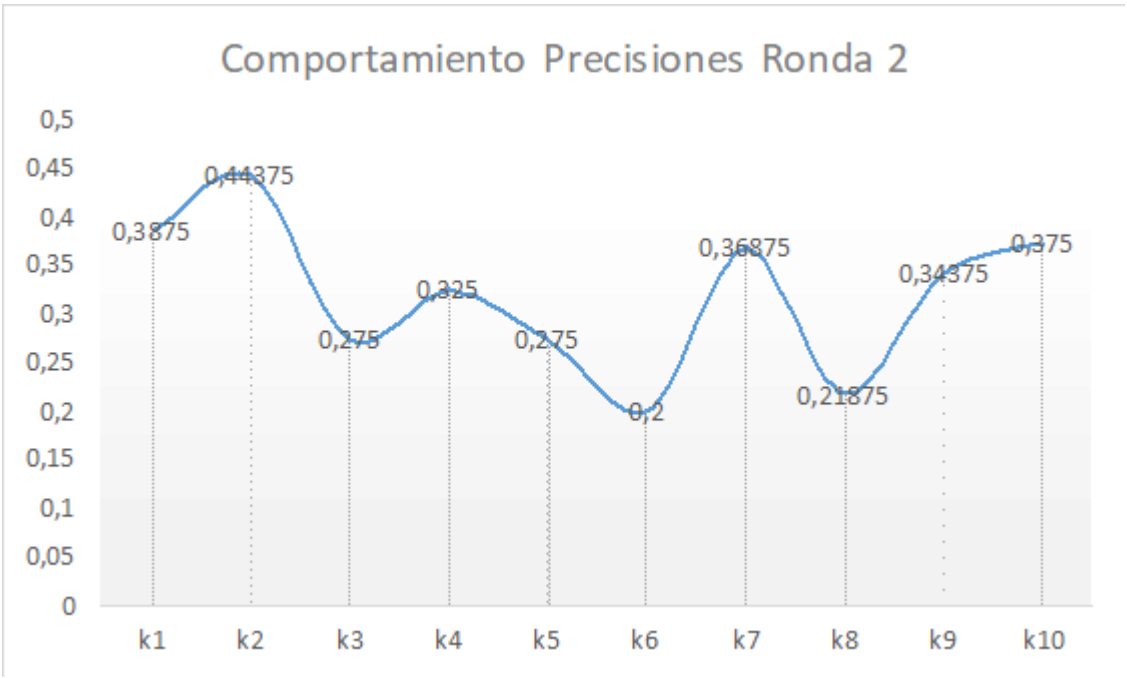
**Precisiones obtenidas en la Ronda#1**

- **Mejor Precisión:** 13,12%
- **Peor Precisión:** 9,37%
- Gráfico de comportamiento:
  Lo que indica es el valor de precisión que se obtuvo por cada grupo K, en el cross validation (K-Fold)



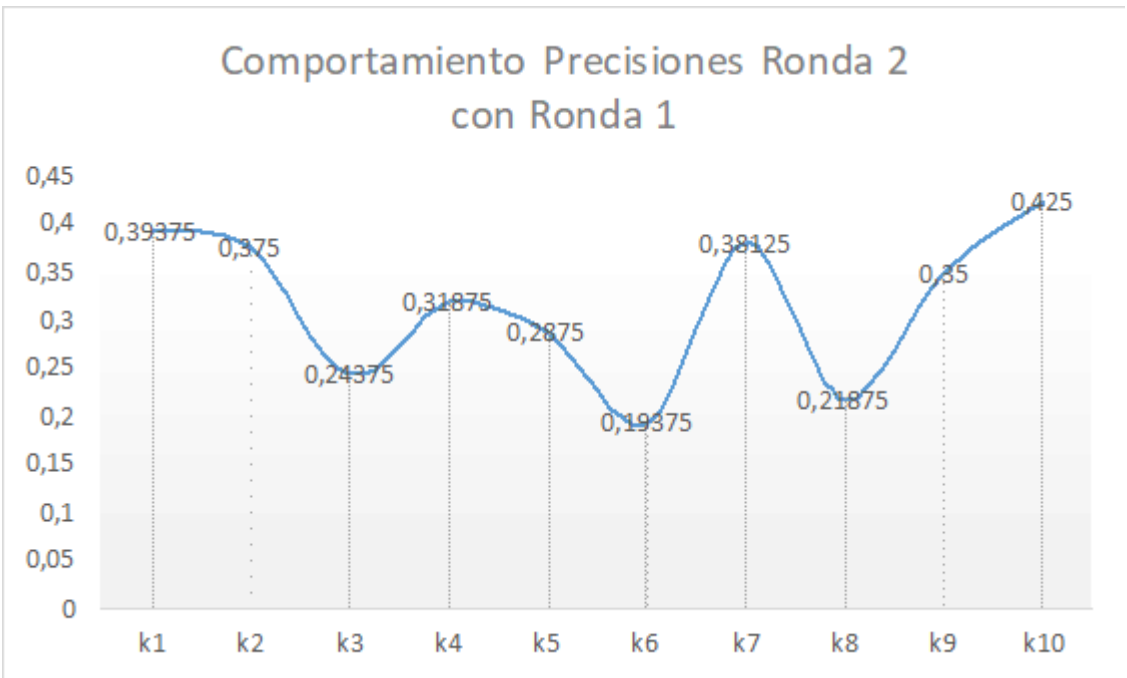

**Precisiones obtenidas en la Ronda#2**

- **Mejor Precisión:** 44,37%
- **Peor Precisión:** 20%
- Gráfico de comportamiento:
  Lo que indica es el valor de precisión que se obtuvo por cada grupo K, en el cross validation (K-Fold)




**Precisiones obtenidas en la Ronda#2 con Ronda#1**

- **Mejor Precisión:** 42,5%
- **Peor Precisión:** 19,37%
- Gráfico de comportamiento:
  Lo que indica es el valor de precisión que se obtuvo por cada grupo K, en el cross validation (K-Fold)



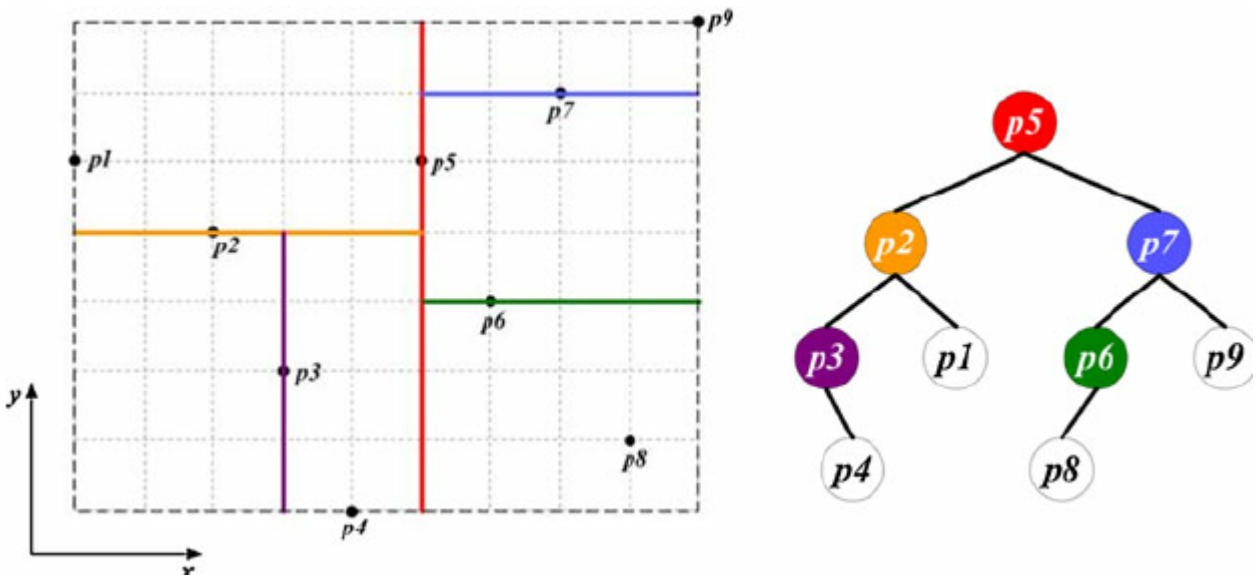

## Clasificación basada en KNN con Kd-trees

### Teoría

La definición de la función $NN(k, x_i)$ es sencilla; dado un conjunto de $N$ muestras y una consulta $x_q$, se debe retornar la muestra $n_i$ que resulte en la menor distancia con $x_q$. Esto tiene un costo computacional de $O(N)$ por lo que no resulta eficiente ante grandes cantidades de datos .

Una solución a esto es disminuir el tiempo de consulta por medio de la estructura `k-d tree` . Con esta estructura se puede reducir el espacio de búsqueda a la mitad cada vez que se realiza una iteración. En cada una de estas iteraciones se selecciona un atributo mediante algún criterio, como la varianza, o por medio de alguna secuencia definida.

En el siguiente ejemplo se puede apreciar el procedimiento utilizado para contruir un `k-d tree` con solamente dos dimensiones.




Primeramente, se dividen las muestras utilizando el atributo $x$ , posteriormente se utiliza el atributo $y$ para los dos subconjuntos resultantes ( , ) recursivamente se aplica el procedimiento hasta que solamente se conserve un par $(x, y)$ en los últimos nodos .

## Implementación

El `k-d tree` ha sido implementado mediante diccionarios de `Python`. Esto debido a que era conveniente para depurar, puesto que era fácilmente visualizable mediante la función `pprint`.

Debido a la necesidad calcular la distancia entre dos vectores, era indispensable que los valores de todas las columnas fueran númericos, por lo que se realiza una etapa de pre-procesamiento que consiste en dos partes.

La primera se encarga de convertir los atributos categóricos en un representacion númerica. Para ello se usa el algoritmo `One Hot Encoding`, el cual crea una columna adicional con valores binarios por cada categoría posible que pueda tener un atributo en particular. Este fue implementado mediante la función `get_dummies` de la librería `pandas`. Un ejemplo de como funciona el algoritmo se ve reflejado en las siguientes tablas:

| Canton | Edad | Sexo |
|---|---|---|
| Paraíso | 20 | M |
| Cervantes | 30 | F |
| Cervantes | 40 | F |

De la tabla anterior, la primera columna debe se convertida a una representación númerica, mientras que la 3 debe ser binaria, por lo que la tabla, luego de este procedimiento, quedaría de la siguiente manera:

| CantonParaiso | CantonCervantes | Edad | Sexo |
|---|---|---|---|
| 1 | 0 | 20 | 1 |
| 0 | 1 | 30 | 0 |
| 0 | 1 | 40 | 0 |

La segunda etapa consiste en normalizar la matriz, para ello se utiliza la función `MinMaxScaler` de la librería `sklearn`. Esta normalización permite que todos los atributos puedan competir entre sí usando la misma escala.

Una vez que se ha realizado el pre-procesamiento la matriz, se puede realizar el entrenamiento del modelo. El procedimiento es el mismo que el descrito anteriormente, sin embargo, la elección del atributo que se usa para realizar las bifurcaciones no está dado al azar, este es elegido calculando la varianza de cada columna de la matriz de muestras. El que obtiene una mayor varianza refleja que es capaz de dividir los datos de una manera más consistente, por lo que es utilizado. Además, se restringe que el atributo que ha sido utilizado en un nodo pueda ser utilizado en sus nodos hijos. Esto porque podría causar que en todas las bifurcaciones se seleccione repetidas veces el mismo atributo.

Adicionalmente, se define un máximo en cuanto a la profundidad que puede tener el `k-d tree`. Como es de evidente, al definir una profundidad el tamaño de las hojas se ve incrementado. Esto se hace para poder disminuir la duración de la creación del `k-d tree`, sin embargo se ve afectado el tiempo promedio para predecir.

Después de usar `k-fold-cross-validation` se obtiene el árbol que cuenta con la mayor precisión. Las pruebas finales son realizadas con este árbol. Para predecir una etiqueta a partir de este, se tiene que definir un `k`, que representa la cantidad de vecinos más cercanos que se deben de tomar en cuenta para tomar una decisión. Al hacer $k = 1$ se está ignorando otros vecinos que pueden afectar significativamente la decisión final, por lo que siempre resulta siempre conveniente encontrar el `k` más óptimo para un modelo en particular.

Al realizar la búsqueda para una consulta $x_q$, se utiliza el nodo raíz, a partir de este se compara el valor del atributo que se utilizó para bifurcar el árbol con el atributo en la misma posición en $x_q$. Si el atributo de la consulta en esta posición es mayor, se limita el área de búsqueda hacia el nodo de la derecha, de lo contrario se limita hacia la izquierda. En algunas ocasiones este valor es igual, por lo que la decisión debe ser tomada calculando la distancia entre la consulta y los nodos hijos del nodo actual. Al llegar a una hoja, se obtienen las distancias entre $x_q$ y los vectores contenidos en los nodos visitados, incluyendo también los

vectores que residen en las hojas. Se toman los `k` más cercanos y se realiza una 'votación' para decidir cual es la etiqueta correspondiente a la consulta.

## Resultados obtenidos

Se ha ejecutado el modelo utilizando distintos valores de $k$, y con una cantidad de muestras $n$. Cada entrenamiento del modelo ha sido realizado utilizando un $10\%$ como porcentaje reservado para pruebas. Se ha probado con valores de $k$ impares para que no existan empates al momento de contar los "votos".

**NOTA:** Para enterder las tablas que se presentan a continuación se debe tomar en cuenta que cada celda representa el porcentaje de precisión obtenido por el modelo para la `ronda 1`, `ronda 2 sin ronda 1` y `ronda 2 con ronda 1` en ese mismo orden.

A continuación, un resumen de los promedios de precisión obtenidos mediante el proceso de `k-fold-cross-validation`.

| n/k | 1 | 5 | 11 | 21 | 41 |
|---|---|---|---|---|---|
| 100 | 24, 54, 54 | 22, 49, 56 | 19, 50, 45 | 22, 62, 63 | 18, 63, 58 |
| 500 | 18, 56, 50 | 23, 56, 53 | 22, 61, 56 | 22, 57, 53 | 20, 53, 52 |
| 1.000 | 20, 48, 53 | 22, 54, 53 | 20, 53, 55 | 23, 53, 54 | 21, 57, 58 |
| 2.000 | 16, 55, 55 | 20, 56, 57 | 22, 56, 53 | 21, 57, 55 | 20, 55, 51 |
| 4.000 | 19, 52, 52 | 19, 53, 52 | 19, 53, 56 | 23, 52, 53 | 20, 58, 55 |
| 10.000 | 19, 51, 52 | 19, 53, 54 | 22, 52, 54 | 20, 53, 56 | 22, 57, 58 |
| 100.000 | 18, 50, 51 | 18, 55, 55 | 21, 58, 57 | 21, 59, 58 | 22, 57, 58 |

De forma general, se puede concluir que el porcentaje promedio de predicciones correctas es sumamente pobre comparado con las redes neuranales, siendo más similar a un árbol de decisión.

La siguiente tabla resume las mejores precisiones obtenidas mediante el mismo proceso de `k-fold-cross-validation`.

| n/k | 1 | 5 | 11 | 21 | 41 |
|---|---|---|---|---|---|
| 100 | 50, 67, 61 | 28, 67, 61 | 33, 61, 67 | 39, 67, 67 | 22, 78, 78 |
| 500 | 24, 63, 58 | 24, 60, 64 | 26, 66, 66 | 34, 63, 54 | 28, 60, 59 |
| 1.000 | 26, 55, 56 | 23, 60, 59 | 25, 55, 59 | 32, 62, 63 | 23, 63, 62 |
| 2.000 | 19, 61, 61 | 21, 61, 61 | 24, 60, 59 | 24, 61, 63 | 22, 59, 60 |
| 4.000 | 20, 57, 56 | 20, 57, 54 | 21, 59, 61 | 25, 60, 59 | 22, 61, 59 |
| 10.000 | 20, 53, 53 | 22, 56, 56 | 24, 57, 56 | 23, 61, 62 | 24, 61, 61 |
| 100.000 | 18, 52, 55 | 19, 57, 57 | 23, 58, 59 | 22, 60, 60 | 28, 59, 59 |

Finalmente, se presentan por medio de la siguiente tabla, las precisiones obtenidas para el conjunto reservado para pruebas.

| n/k | 1 | 5 | 11 | 21 | 41 |
|---|---|---|---|---|---|
| 100 | 30, 60, 40 | 20, 60, 60 | 20, 80, 80 | 10, 80, 80 | 30, 70, 70 |
| 500 | 26, 56, 64 | 22, 50, 52 | 28, 58, 58 | 14, 64, 07 | 22, 60, 60 |
| 1.000 | 12, 52, 52 | 17, 58, 52 | 19, 54, 60 | 23, 64, 67 | 18, 68, 61 |
| 2.000 | 20, 64, 57 | 24, 55, 62 | 26, 61, 60 | 22, 59, 57 | 20, 59, 60 |
| 4.000 | 19, 55, 53 | 20, 59, 55 | 23, 59, 59 | 23, 58, 64 | 24, 64, 63 |
| 10.000 | 19, 54, 54 | 20, 54, 53 | 24, 57, 59 | 24, 62, 63 | 23, 59, 57 |
| 100.000 | 19, 53, 55 | 18, 56, 57 | 23, 58, 59 | 22, 60, 60 | 23, 60, 60 |

# Acerca de

Integrantes del proyecto:

| Nombre | Carné |
|---|---|
| Brandon Dinarte Chavarría | 2015088894 |
| Armando López Cordero | 2015125414 |
| Julian Salinas Rojas | 2015114132 |

Estudianteas de Ingeniería en Computación, en el Instituto Tecnológico de Costa Rica.