Proyecto 1: Predicción de votos.

Introducción.

Este es un proyecto del curso Inteligencia Artificial; es desarrollado por los estudiantes: María Fernanda Alvarado, Freyser Jímenez y Minor Sancho durante el primer semestre de 2018 en el Instituto Tecnológico de Costa Rica.
El proyecto consiste en predecir los votos tanto de la primera como la segunda ronda de las elecciones vividas este 2018, para el cuál se desarrollará un programa primario que podrá entrenar distintos modelos de clasificación de votantes y generará una serie de archivos de salida analizando el rendimiento de cada modelo.

Justificación.

El objetivo primario del proyecto será enfrentar a los estudiantes con una situación cercana a un proyecto de clasificación real donde existe una fuente de datos cruda, debe procesarse los mismos, compara algoritmos y reportar los resultados de una manera formal.

Especificación.

Descripción general de los entregables.

A continuación se listan los elementos que se considerarán en los entregables para efectos de evaluación que los estudiantes deberán

completar como parte del proyecto:

- Simulador de votantes aumentado con información de segunda ronda (10 puntos).
- Programa principal para iniciar entrenamiento de un modelo:
    - Clasificación basada en modelos lineales (15 puntos).
    - Clasificación basada en redes neuronales (15 puntos).
    - Clasificación basada en árboles de decisión (25 puntos).
    - Clasificación basada en KNN con Kd-trees (25 puntos).
    - Clasificación basada en SVM (25 puntos opcionales).
    - Operación básica y detalles ingeniería de software según especificación (5 puntos, si al menos un método es funcional).
- Informe:
    - Manual de instalación y uso (5 puntos).
    - Reporte por cada método implementado (puntaje contenido en cada rubro anterior).
- Presentación de resultados a la clase (Proyecto corto #2).

Simulador aumentado.

La primera tarea realizada tendrá que ver con modificar el simulador de datos ya creado. La interfaz de las funciones se mantendrá igual pero estudiantes deberán retornar una lista de listas con una columna más: la etiqueta que representa por quién se votó en segunda ronda. Se espera que el simulador sea llamado internamente por el programa principal del proyecto. Es decir, no será aceptable que primer haya que correr el simulador para generar conjuntos de datos y después llamar por separado

al programa de clasificación. Se espera que importar las funciones se pueda hacer de la misma manera:

```
from tec.ic.ia.pc1.g01 import generar_muestra_pais, generar_muestra_provincia
```

Modelos lineales.

El programa principal deberá realizar el entrenamiento y evaluación de un modelo de regresión logística cuando se le pase el parámetro *–regresion-logistica* por la línea de comandos. Se espera que los estudiantes utilicen tensorflow directamente para realizar la creación del modelo.

Análisis de resultados.

Deberán evaluar diferentes niveles de regularización (tanto L1 como L2) provisto por tensorflow a la hora de crear los modelos. Deberá exponerse dos banderas para especificar el valor por línea de comandos: *–l1* y *–l2* (L minúscula).

Redes neuronales.

De manera similar, se deberá entrenar un modelo con redes neuronales. En este caso los estudiantes deberán utilizar keras para tales efectos, cuando seprovea la bandera *–red-neuronal*.

Análisis de resultados.

Evaluar diferentes estructuras (al menos 3) y funciones de activación (al menos 2). Exponer banderas llamadas *–numero-capas*, *–unidades-por-capa* y *–funcion-activacion* para configurar cada uno de los 3. Los estudiantes podrán documentar en el manual cuáles valores son válidos para la bandera de funcion-activacion.

Árboles de decisión.

Los estudiantes deberán implementar su propio árbol de decisión incluyendo un paso de poda. Para ello el programa recibirá la bandera *–arbol* indicando que este es el tipo de modelo a entrenar.

Análisis de resultados.

Evaluar diferentes criterios de poda, utilizando la bandera *–umbral-poda* para especificar la ganancia de información mínima requerida para realizar una partición. Es importante recordar que la poda se aplica hasta que el árbol completo haya sido construido.

KNN.

La bandera *–knn* indicará que se debe entrenar un modelo de nearest neighbors. Se utilizará la bandera *–k<numero>* para indicar el número de vecinos a considerar. La estructura de datos interna a utilizar para agrupar los votantes deberá ser un kd-tree, implementado por los estudiantes.

Análisis de resultados.

Evaluar diferentes valores de k.

SVM

Como se mencionó anteriormente, este apartado es opcional. El objetivo es que estudiantes
utilicen alguna biblioteca para clasificación con SVM. Se sugiere utilizar scikit pero se deja a discreción de los estudiantes. Se señalará al programa con la bandera
–svm. Dado que la escogencia de labiblioteca es abierta se le solicita a los estudiantes documentar en la sección de manual de usuario cuáles banderas adicionales deben agregarse.

Análisis de resultados.

Al ser la escogencia abierta no hay un parámetro particular dado en esta sección, sin embargo, los estudiantes deberán escoger algún(os) parámetro(s) expuestos por la biblioteca seleccionada y generar análisis basado en ellos. Los parámetros seleccionados deberán ser expuestos como banderas.

Requerimientos básicos y operativos programa principal.

La entrada del programa será una de las banderas mencionadas anteriormente,una bandera llamada *–prefijo* que se utilizará para poner al frente del nombre de todos los archivos necesarios durante una corrida y

un tamaño de población a generar *(–poblacion<numero>)*. Además,se utilizará una bandera llamada *–porcentaje-pruebas<porcentaje>*. Esto quiere decir que, por ejemplo, tengamos *–poblacion 10000* y *–porcentaje-pruebas 20*, 2000 votantes deberán reservarse para la validación final.
La salida por línea de comandos para cada modelo deberá ser el error de entrenamiento (utilizando cross validation) y pruebas (utilizando un set aparte). Además deberá generar un archivo de salida CSV, que contiene todos los datos originalmente generados por el simulador y cuatro columnas adicionales:

- **es_entrenamiento:** que deberá ser verdadera para una corrida particular se utilizó en el proceso de cross validation.
- **prediccion_r1:** predicción del partido político por el que se votó en primera ronda.
- **prediccion_r2:** predicción del partido político por el que se votó en segunda ronda. No incluye la columna de voto real en primera ronda.
- **prediccion_r2_con_r1:** predicción del partido político por el que se en segunda ronda. Si incluye el voto de primera ronda como atributo para entrenar el modelo.
  Todo el código deberá ser administrado en github.com en un repositorio privado. Nótese que existen cuentas gratuitas para estudiantes.

Informe.

Los estudiantes deberán realizar un reporte técnico cuya audiencia será miembros la escuela que NO están cursando Inteligencia Artificial (ya sea

estudiantes o profesores). Deberán describir todos los aspectos técnicos de la realización (iniciando con el simulador de votantes) hasta un análisis de resultados.

Para cada uno de los modelos se espera que se haga un análisis de resultados comentando qué detalles se tomaron en cuenta para la configuración de cada modelo (e.g. cantidad de capas o unidades en redes neuronales), y las medidas de rendimiento detrás de cada modelo.

El puntaje del análisis cada sección será parte del puntaje generaldel modelo. La descripción del simulador será contemplada en el puntaje del mismo.

Además del análisis se pide generar un apéndice en el que sea un manual de para instalar y correr el proyecto. Se puede asumir que la computadora posee Python instalado únicamente (e.g. debe explicarse cómo instalar tensorflow con pip).

El informe deberá realizarse utilizando markdown en un repositorio público de github.com, perteneciente a los estudiantes. Deberá enviarse una copia en PDF profesor a través de TEC Digital.

Presentación de resultado.

Una vez concluido el proyecto se calendarizarán presentaciones donde los estudiantes mostrarán sus resultados a la Escuela. Los detalles se darán posteriormente,sin embargo, es importante que a lo largo del proceso de este proyecto los estudiantes entiendan que su trabajo podrá ser analizado afuera de la audiencia del curso.

Detalles de Entrega y Revisión.

El proyecto deberá realizarse en los mismos grupos para el Proyecto Corto #1.
Se debe enviar todos los entregables a través de TEC Digital a más tardar el 4 de mayo del año en curso, antes de las 11:59PM.
El profesor se reserva derecho a asignar una nota de cero si los requerimientos no son cumplidos de forma tal que impida ejecutar las funciones principales.
A manera de resumen los entregables serán:

- Código fuente con prefijo modular de Python *tec.ic.ac.p1.g01.* Este deberá estar almacenado en un repositorio privado de github. Se deberá enviar también un archivo comprimido a través de TEC digital.
- Los archivos CSV utilizados por el simulador de votantes.
- Pruebas unitarias. En particular hacerse énfasis en las secciones implementadas por los estudiantes (KNN y árboles de decisión).
- Informe, en formato PDF (a TEC Digital) y Markdown en github.com

Consideraciones Python

- Se utilizará Python 3 (no 2.7).
- Se utilizará pip para instalar el módulo enviado. Se debe respetar la estructura de directorios apropiada para que la instalación sea exitosa.
- Se debe utilizar pytest para pruebas unitarias.
- Seguir recomendaciones en http://docs.python-guide.org/en/latest/

- Seguir PEP 20. Una guía rápida con ejemplos está disponible en: http://artifex.org/~hblanks/talks/2011/pep20_by_example.pdf
- Utilizar PEP 8 como guía de estilo: http://pep8.org/

Análisis técnico.

En esta sección se explican cada uno de los modelos desarrollados, además se detalla como fuerón implementados, las herramientas utilizadas y los conceptos básicos para cada modelo.

Simulador de votos aumentado.

Datos utilizados.

Para la creación del generador de votantes fue necesario realizar un mapeo entre las juntas receptores y cantones, esto con el fin de poder crear un conjunto de datos de partida para poder entrenar el mismo.
Actas de escrutinio: Contiene la cantidad de votos emitidos por cada mesa.
http://www.tse.go.cr/elecciones2018/actas_escrutinio.htm
Mapeo juntas a cantones: Contiene la información de cada cantón y sus mesas respectivas.
http://www.tse.go.cr/pdf/nacional2018/JRV.pdf
Indicadores cantonales: Contiene diferentes indicadores para cada cantón.
https://www.estadonacion.or.cr/files/biblioteca_virtual/otras_publicaciones/I
cantonales_Censos2000y2011.xlsx

Después de realizar el mapeo de los datos previamente mencionados, se utilizó un .CSV para almacenar los mismos (el proceso de mapeo debe ser

hecho de manera manual). La estructura del archivo es la siguiente:

PROVINCIA, CANTON, ACCESIBILIDAD SIN EXCLUSION, ACCION CIUDADANA, ALIANZA DEMOCRATA CRISTIANA, DE LOS TRABAJADORES, FRENTE AMPLIO, INTEGRACION NACIONAL LIBERACION NACIONAL, MOVIMIENTO LIBERTARIO, NUEVA GENERACION, RENOVACION COSTARRICENSE, REPUBLICANO SOCIAL CRISTIANO, RESTAURACION NACIONAL, UNIDAD SOCIAL CRISTIANA, VOTOS NULOS, VOTOS BLANCOS, Población total, Superficie (km2), Densidad de población, Porcentaje de población urbana, Relación hombres-mujeres, Relación de dependencia demográfica, Viviendas individuales ocupadas, Promedio de ocupantes, Porcentaje de viviendas en buen estado, Porcentaje de viviendas hacinadas, Porcentaje de alfabetismo, 10 a 24 años, 25 y más años, Escolaridad promedio, 25 a 49 años, 50 o más años, Porcentaje de asistencia a la educación regular, Menor de 5 años, 5 a 17 años, 18 a 24 años, 25 y más años, Personas fuera de la fuerza de trabajo (15 años y más) , Tasa neta de participación, Hombres, Mujeres, Porcentaje de población ocupada no asegurada, Porcentaje de población nacida en el extranjero, Porcentaje de población con discapacidad, Porcentaje de población no asegurada, Porcentaje de hogares con jefatura femenina, Porcentaje de hogares con jefatura compartida.

Ejemplo archivo.

Lectura de archivo

Una vez finalizado el proceso de creación del archivo de votos reales, es

necesario leer el mismo y almacenar la información en una variable (nuestro caso una variable global de tipo lista). Estos datos son almacenados en sub-listas por provincia, esto con el fin de tener un mejor manejo de los datos por provincia.

Generador votos país primera ronda.

Para la generación de votos por país de la primera ronda se utilizó un algoritmo probabilístico que permite hacer un **“random”** para la elección del voto sin perder la distribución real de los votos realizados en la primera ronda.

El funcionamiento del algoritmo es baste sencillo, la idea principal es crear una lista de índices según los votos reales y luego utilizar la función random para obtener un valor en el rango de índices, una vez asignado el voto, el siguiente paso por hacer es asignar la provincia. Para la asignación de la provincia se tomó en cuenta la probabilidad de que un voto especifico haya sido emitido en **“X”** provincia por lo que se tiene que calcular cada vez que un voto es creado.

Después de tener el voto y la provincia lo más importante es asignar un cantón (el cantón debe pertenecer a la provincia). Este elemento es uno de los más importantes pues de este atributo dependen los demás atributos relacionados a los índices cantonales. Para poder asignar el cantón se realiza el mismo procedimiento que asignar provincia.

Otros atributos que se deben asignar son el sexo y la edad, para la asignación del sexo se respecto a la distribución del país (49% hombres, 51% mujeres). Para la asignación de la edad se utilizó un random entre 18 y 81 años de edad.

El resultado de correr el generador es una lista de sub-listas, donde cada

sub-lista contiene 36 atributos y la última columna es el voto simulado.

Generador votos país segunda ronda.

Para la generación de votos de la segunda ronda se mantienen todos los datos generados por **“generador votos país primera ronda”**, el algoritmo utilizado para la simulación del segundo voto es el mismo que se describió con anterioridad, basado en la probabilidad de los votos se crea una lista de índices y después se utiliza un random para saber por cuál partido se emitió el voto.

Generador votos provincia primera ronda.

Para la generación de votos por provincia de la primera ronda se utilizó un algoritmo probabilístico que permite hacer un “random” para la elección del voto sin perder la distribución real de los votos realizados en la primera ronda.
El funcionamiento del algoritmo es baste sencillo, la idea principal es crear una lista de índices según los votos reales y luego utilizar la función random para obtener un valor en el rango de índices, una vez asignado el voto, el siguiente paso por hacer es asignar la provincia. Para la asignación de la provincia se tomó en cuenta la probabilidad de que un voto especifico haya sido emitido en “X” provincia por lo que se tiene que calcular cada vez que un voto es creado.
Después de tener el voto y la provincia lo más importante es asignar un cantón (el cantón debe pertenecer a la provincia). Este elemento es uno de los más importantes pues de este atributo dependen los demás atributos relacionados a los índices cantonales. Para poder asignar el cantón se realiza el mismo procedimiento que asignar provincia.

Otros atributos que se deben asignar son el sexo y la edad, para la asignación del sexo se respecto a la distribución del país (49% hombres, 51% mujeres). Para la asignación de la edad se utilizó un random entre 18 y 81 años de edad.
El resultado de correr el generador es una lista de sub-listas, donde cada sub-lista contiene 36 atributos y la última columna es el voto simulado.

Generador votos provincia segunda ronda.

Para la generación de votos de la segunda ronda se mantienen todos los datos generados por **“generador votos provincia primera ronda”**, el algoritmo utilizado para la simulación del segundo voto es el mismo que se describió con anterioridad, basado en la probabilidad de los votos se crea una lista de índices y después se utiliza un random para saber por cuál partido se emitió el voto.

Funciones.

Generar muestra por provincia para la primer ronda electoral.

```
def generar_muestra_provincia(n, provincia):
    global votos_provincia, partidos_politicos
    partidos = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
    leerArchivoCSV("datos.csv")
    cantones = cargar_datos_cantones(provincia)
    votos_t = calcular_cantidad_voto_por_canton(cantones)
    promedio_voto = calcular_probabilidad_voto_por_partido_ca
nton(
        cantones, votos_t)
```

```
    sumar_todos_por_provincia(provincia)
    votos = votos_provincia[1]
    # print(votos_provincia[1])
    votantes_simulados = []
    for i in range(1, n + 1):
        numero = random.randrange(votos_provincia[2])
        edad = random.randrange(18, 82, 2)
        if (numero <= suma_partes_lista([votos[0]])):
            partidos[0] += 1
            lista_promedio_cantones = escoger_canton_segun_vo
to(
                partidos_politicos[0], promedio_voto, n)
            canton = random.randrange(len(lista_promedio_cant
ones))
            indicadores = obtener_inidicadores_canton(
                lista_promedio_cantones[canton], cantones)
            sexo = asignar_sexo(indicadores[23], indicadores[
24])
            tempo = [[lista_promedio_cantones[canton]] + indi
cadores + [sexo] + [edad]+ [partidos_politicos[0]]]
            votantes_simulados += tempo
            tempo = []
        elif (numero > suma_partes_lista([votos[0]]) and nume
ro <= suma_partes_lista(votos[:2])):
            partidos[1] += 1
            lista_promedio_cantones = escoger_canton_segun_vo
to(
                partidos_politicos[1], promedio_voto, n)
```

```
            canton = random.randrange(len(lista_promedio_cant
ones))
            indicadores = obtener_inidicadores_canton(
                lista_promedio_cantones[canton], cantones)
            sexo = asignar_sexo(indicadores[23], indicadores[
24])
            tempo = [
                [lista_promedio_cantones[canton]] + indicador
es + [sexo] + [edad]+ [partidos_politicos[1]]]
            votantes_simulados += tempo
            tempo = []
        elif (numero > suma_partes_lista(votos[:2]) and numer
o <= suma_partes_lista(votos[:3])):
            partidos[2] += 1
            lista_promedio_cantones = escoger_canton_segun_vo
to(
                partidos_politicos[2], promedio_voto, n)
            canton = random.randrange(len(lista_promedio_cant
ones))
            indicadores = obtener_inidicadores_canton(
                lista_promedio_cantones[canton], cantones)
            sexo = asignar_sexo(indicadores[23], indicadores[
24])
            tempo = [
                [lista_promedio_cantones[canton]] + indicador
es + [sexo] + [edad]+ [partidos_politicos[2]]]
            votantes_simulados += tempo
            tempo = []
```

```
        elif (numero > suma_partes_lista(votos[:3]) and numer
o <= suma_partes_lista(votos[:4])):
            partidos[3] += 1
            lista_promedio_cantones = escoger_canton_segun_vo
to(
                partidos_politicos[3], promedio_voto, n)
            canton = random.randrange(len(lista_promedio_cant
ones))
            indicadores = obtener_inidicadores_canton(
                lista_promedio_cantones[canton], cantones)
            sexo = asignar_sexo(indicadores[23], indicadores[
24])
            tempo = [
                [lista_promedio_cantones[canton]] + indicador
es + [sexo] + [edad]+ [partidos_politicos[3]]]
            votantes_simulados += tempo
            tempo = []
        elif (numero > suma_partes_lista(votos[:4]) and numer
o <= suma_partes_lista(votos[:5])):
            partidos[4] += 1
            lista_promedio_cantones = escoger_canton_segun_vo
to(
                partidos_politicos[4], promedio_voto, n)
            canton = random.randrange(len(lista_promedio_cant
ones))
            indicadores = obtener_inidicadores_canton(
                lista_promedio_cantones[canton], cantones)
            sexo = asignar_sexo(indicadores[23], indicadores[
```

```
24])
            tempo = [
                [lista_promedio_cantones[canton]] + indicador
es + [sexo] + [edad] + [partidos_politicos[4]]]
            votantes_simulados += tempo
            tempo = []
        elif (numero > suma_partes_lista(votos[:5]) and numer
o <= suma_partes_lista(votos[:6])):
            partidos[5] += 1
            lista_promedio_cantones = escoger_canton_segun_vo
to(
                partidos_politicos[5], promedio_voto, n)
            canton = random.randrange(len(lista_promedio_cant
ones))
            indicadores = obtener_inidicadores_canton(
                lista_promedio_cantones[canton], cantones)
            sexo = asignar_sexo(indicadores[23], indicadores[
24])
            tempo = [
                [lista_promedio_cantones[canton]] + indicador
es + [sexo] + [edad]+ [partidos_politicos[5]]]
            votantes_simulados += tempo
            tempo = []
        elif (numero > suma_partes_lista(votos[:6]) and numer
o <= suma_partes_lista(votos[:7])):
            partidos[6] += 1
            lista_promedio_cantones = escoger_canton_segun_vo
to(
```

```
                partidos_politicos[6], promedio_voto, n)
            canton = random.randrange(len(lista_promedio_cant
ones))
            indicadores = obtener_inidicadores_canton(
                lista_promedio_cantones[canton], cantones)
            sexo = asignar_sexo(indicadores[23], indicadores[
24])
            tempo = [
                [lista_promedio_cantones[canton]] + indicador
es + [sexo] + [edad]+ [partidos_politicos[6]]]
            votantes_simulados += tempo
            tempo = []
        elif (numero > suma_partes_lista(votos[:7]) and numer
o <= suma_partes_lista(votos[:8])):
            partidos[7] += 1
            lista_promedio_cantones = escoger_canton_segun_vo
to(
                partidos_politicos[7], promedio_voto, n)
            canton = random.randrange(len(lista_promedio_cant
ones))
            indicadores = obtener_inidicadores_canton(
                lista_promedio_cantones[canton], cantones)
            sexo = asignar_sexo(indicadores[23], indicadores[
24])
            tempo = [
                [lista_promedio_cantones[canton]] + indicador
es + [sexo] + [edad]+ [partidos_politicos[7]]]
            votantes_simulados += tempo
```

```
            tempo = []
        elif (numero > suma_partes_lista(votos[:8]) and numer
o <= suma_partes_lista(votos[:9])):
            partidos[8] += 1
            lista_promedio_cantones = escoger_canton_segun_vo
to(
                partidos_politicos[8], promedio_voto, n)
            canton = random.randrange(len(lista_promedio_cant
ones))
            indicadores = obtener_inidicadores_canton(
                lista_promedio_cantones[canton], cantones)
            sexo = asignar_sexo(indicadores[23], indicadores[
24])
            tempo = [
                [lista_promedio_cantones[canton]] + indicador
es + [sexo] + [edad]+ [partidos_politicos[8]]]
            votantes_simulados += tempo
            tempo = []
        elif (numero > suma_partes_lista(votos[:9]) and numer
o <= suma_partes_lista(votos[:10])):
            partidos[9] += 1
            lista_promedio_cantones = escoger_canton_segun_vo
to(
                partidos_politicos[9], promedio_voto, n)
            canton = random.randrange(len(lista_promedio_cant
ones))
            indicadores = obtener_inidicadores_canton(
                lista_promedio_cantones[canton], cantones)
```

```
            sexo = asignar_sexo(indicadores[23], indicadores[
24])
            tempo = [
                [lista_promedio_cantones[canton]] + indicador
es + [sexo] + [edad]+ [partidos_politicos[9]]]
            votantes_simulados += tempo
            tempo = []
        elif (numero > suma_partes_lista(votos[:10]) and nume
ro <= suma_partes_lista(votos[:11])):
            partidos[10] += 1
            lista_promedio_cantones = escoger_canton_segun_vo
to(
                partidos_politicos[10], promedio_voto, n)
            canton = random.randrange(len(lista_promedio_cant
ones))
            indicadores = obtener_inidicadores_canton(
                lista_promedio_cantones[canton], cantones)
            sexo = asignar_sexo(indicadores[23], indicadores[
24])
            tempo = [
                [lista_promedio_cantones[canton]] + indicador
es + [sexo] + [edad]+ [partidos_politicos[10]]]
            votantes_simulados += tempo
            tempo = []
        elif (numero > suma_partes_lista(votos[:11]) and nume
ro <= suma_partes_lista(votos[:12])):
            partidos[11] += 1
            lista_promedio_cantones = escoger_canton_segun_vo
```

```
to(
                partidos_politicos[11], promedio_voto, n)
            canton = random.randrange(len(lista_promedio_cant
ones))
            indicadores = obtener_inidicadores_canton(
                lista_promedio_cantones[canton], cantones)
            sexo = asignar_sexo(indicadores[23], indicadores[
24])
            tempo = [
                [lista_promedio_cantones[canton]] + indicador
es + [sexo] + [edad]+ [partidos_politicos[11]]]
            votantes_simulados += tempo
            tempo = []
        elif (numero > suma_partes_lista(votos[:12]) and nume
ro <= suma_partes_lista(votos[:13])):
            partidos[12] += 1
            lista_promedio_cantones = escoger_canton_segun_vo
to(
                partidos_politicos[12], promedio_voto, n)
            canton = random.randrange(len(lista_promedio_cant
ones))
            indicadores = obtener_inidicadores_canton(
                lista_promedio_cantones[canton], cantones)
            sexo = asignar_sexo(indicadores[23], indicadores[
24])
            tempo = [
                [lista_promedio_cantones[canton]] + indicador
es + [sexo] + [edad]+ [partidos_politicos[12]]]
```

```
            votantes_simulados += tempo
            tempo = []
        elif (numero > suma_partes_lista(votos[:13]) and nume
ro <= suma_partes_lista(votos[:14])):
            partidos[13] += 1
            lista_promedio_cantones = escoger_canton_segun_vo
to(
                partidos_politicos[13], promedio_voto, n)
            canton = random.randrange(len(lista_promedio_cant
ones))
            indicadores = obtener_inidicadores_canton(
                lista_promedio_cantones[canton], cantones)
            sexo = asignar_sexo(indicadores[23], indicadores[
24])
            tempo = [[lista_promedio_cantones[canton]] + indi
cadores + [sexo] + [edad]+ [partidos_politicos[13]]]
            votantes_simulados += tempo
            tempo = []
        elif (numero > suma_partes_lista(votos[:14]) and nume
ro <= suma_partes_lista(votos)):
            partidos[14] += 1
            lista_promedio_cantones = escoger_canton_segun_vo
to(
                partidos_politicos[14], promedio_voto, n)
            canton = random.randrange(len(lista_promedio_cant
ones))
            indicadores = obtener_inidicadores_canton(
                lista_promedio_cantones[canton], cantones)
```

```
            sexo = asignar_sexo(indicadores[23], indicadores[
24])
            tempo = [[lista_promedio_cantones[canton]] + indi
cadores + [sexo] + [edad]+ [partidos_politicos[14]]]
            votantes_simulados += tempo
            tempo = []
    # print(partidos)
    return votantes_simulados
```

Generar muestra por país para la primer ronda electoral.

```
def generar_muestra_pais(numero):
    global total
    if len(total) == 0:
        leerArchivoCSV("datos.csv")

    promedio_votos_x_provincia = promedio_votos_por_provincia
()
    lista_voto_provincia = lista_probalidades_voto_provincia(
        promedio_votos_x_provincia, numero)
    votantes = []
    for i in range(numero):
        edad = random.randrange(18, 82, 2)
        numero = random.randrange(len(lista_voto_provincia))
        cantones = cargar_datos_cantones(lista_voto_provincia
[numero][0])
        votos_t = calcular_cantidad_voto_por_canton(cantones)
```

```
        promedio_voto = calcular_probabilidad_voto_por_partid
o_canton(
            cantones, votos_t)
        lista_promedio_cantones = escoger_canton_segun_voto(
            lista_voto_provincia[numero][1], promedio_voto, n
umero)

        canton = random.randrange(len(lista_promedio_cantones
))

        if isinstance(lista_promedio_cantones[canton], float)
:
            canton_x= seleccionar_canton_random(
                    lista_voto_provincia[numero][0])
            indicadores = obtener_inidicadores_canton(canton_
x
                , cantones)

            sexo = asignar_sexo(indicadores[23], indicadores[
24])
            tempo = [
                    [lista_voto_provincia[numero][0]] +
                     [canton_x] +
                     indicadores + [sexo] + [edad] + [lista_v
oto_provincia[numero][1]]]

##            if tempo[0][0]=="SAN JOSE" :
##                print(tempo[0][0],tempo[0][1])
```

```
##                 tempo[0] += [voto_segunda_ronda_canton(temp
o[0][0],tempo[0][1])]
##

        else:
            indicadores = obtener_inidicadores_canton(
                lista_promedio_cantones[canton], cantones)

            sexo = asignar_sexo(indicadores[23], indicadores[
24])
            tempo = [
                    [lista_voto_provincia[numero][0]] +
                     [lista_promedio_cantones[canton]] +
                     indicadores + [sexo] + [edad] + [lista_v
oto_provincia[numero][1]]]


##             if tempo[0][0]=="SAN JOSE"  :
##                 print(tempo[0][0],tempo[0][1])
##                 tempo[0] += [voto_segunda_ronda_canton(temp
o[0][0],tempo[0][1])]

        votantes += tempo

    return votantes
```

Generar muestra por país para la segunda ronda

electoral.

```
def generar_muestra_pais_con_segunda(numero):
    global total,segundaRonda
    if len(total) == 0:
        leerArchivoCSV("datos.csv")
    promedio_votos_x_provincia = promedio_votos_por_provincia
()
    lista_voto_provincia = lista_probalidades_voto_provincia(
        promedio_votos_x_provincia, numero)
    votantes = []
    for i in range(numero):
        edad = random.randrange(18, 82, 2)
        numero = random.randrange(len(lista_voto_provincia))
        cantones = cargar_datos_cantones(lista_voto_provincia
[numero][0])
        votos_t = calcular_cantidad_voto_por_canton(cantones)
        promedio_voto = calcular_probabilidad_voto_por_partid
o_canton(
            cantones, votos_t)
        lista_promedio_cantones = escoger_canton_segun_voto(
            lista_voto_provincia[numero][1], promedio_voto, n
umero)

        canton = random.randrange(len(lista_promedio_cantones
))

        if isinstance(lista_promedio_cantones[canton], float)
```

```
:
            canton_x= seleccionar_canton_random(
                    lista_voto_provincia[numero][0])
            indicadores = obtener_inidicadores_canton(canton_
x
                , cantones)

            sexo = asignar_sexo(indicadores[23], indicadores[
24])
            voto_segunda= voto_segunda_ronda_canton(lista_vot
o_provincia[numero][0],canton_x)
            tempo = [
                    [lista_voto_provincia[numero][0]] +
                     [canton_x] +
                     indicadores + [sexo] + [edad] + [lista_v
oto_provincia[numero][1]]+[voto_segunda]]

        else:
            indicadores = obtener_inidicadores_canton(
                lista_promedio_cantones[canton], cantones)
            sexo = asignar_sexo(indicadores[23], indicadores[
24])
            voto_segunda= voto_segunda_ronda_canton(lista_vot
o_provincia[numero][0],lista_promedio_cantones[canton])
            tempo = [
                    [lista_voto_provincia[numero][0]] +
                     [lista_promedio_cantones[canton]] +
                     indicadores + [sexo] + [edad] + [lista_v
```

```
oto_provincia[numero][1]]+[voto_segunda]]


        votantes += tempo

    return votantes
```

Generar muestra por provincia para la segunda ronda electoral.

```
def generar_muestra_provincia2(n, provincia):
    global votos_provincia, partidos_politicos
    partidos = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
    leerArchivoCSV("datos.csv")
    cantones = cargar_datos_cantones(provincia)
    votos_t = calcular_cantidad_voto_por_canton(cantones)
    promedio_voto = calcular_probabilidad_voto_por_partido_ca
nton(
        cantones, votos_t)
    sumar_todos_por_provincia(provincia)
    votos = votos_provincia[1]
    # print(votos_provincia[1])
    votantes_simulados = []
    for i in range(1, n + 1):
        numero = random.randrange(votos_provincia[2])
        edad = random.randrange(18, 82, 2)
        if (numero <= suma_partes_lista([votos[0]])):
            partidos[0] += 1
```

```
            lista_promedio_cantones = escoger_canton_segun_vo
to(
                partidos_politicos[0], promedio_voto, n)
            canton = random.randrange(len(lista_promedio_cant
ones))
            indicadores = obtener_inidicadores_canton(
                lista_promedio_cantones[canton], cantones)
            sexo = asignar_sexo(indicadores[23], indicadores[
24])
            voto_segunda= voto_segunda_ronda_canton("",lista_
promedio_cantones[canton])
            tempo = [[lista_promedio_cantones[canton]] + indi
cadores + [sexo] + [edad]+ [partidos_politicos[0]]+[voto_segu
nda]]
            votantes_simulados += tempo
            tempo = []
        elif (numero > suma_partes_lista([votos[0]]) and nume
ro <= suma_partes_lista(votos[:2])):
            partidos[1] += 1
            lista_promedio_cantones = escoger_canton_segun_vo
to(
                partidos_politicos[1], promedio_voto, n)
            canton = random.randrange(len(lista_promedio_cant
ones))
            indicadores = obtener_inidicadores_canton(
                lista_promedio_cantones[canton], cantones)
            sexo = asignar_sexo(indicadores[23], indicadores[
24])
```

```
            voto_segunda= voto_segunda_ronda_canton("",lista_
promedio_cantones[canton])
            tempo = [[lista_promedio_cantones[canton]] + indi
cadores + [sexo] + [edad]+ [partidos_politicos[1]]+[voto_segu
nda]]
            votantes_simulados += tempo
            tempo = []
        elif (numero > suma_partes_lista(votos[:2]) and numer
o <= suma_partes_lista(votos[:3])):
            partidos[2] += 1
            lista_promedio_cantones = escoger_canton_segun_vo
to(
                partidos_politicos[2], promedio_voto, n)
            canton = random.randrange(len(lista_promedio_cant
ones))
            indicadores = obtener_inidicadores_canton(
                lista_promedio_cantones[canton], cantones)
            sexo = asignar_sexo(indicadores[23], indicadores[
24])
            voto_segunda= voto_segunda_ronda_canton("",lista_
promedio_cantones[canton])
            tempo = [[lista_promedio_cantones[canton]] + indi
cadores + [sexo] + [edad]+ [partidos_politicos[2]]+[voto_segu
nda]]
            votantes_simulados += tempo
            tempo = []
        elif (numero > suma_partes_lista(votos[:3]) and numer
o <= suma_partes_lista(votos[:4])):
```

```
            partidos[3] += 1
            lista_promedio_cantones = escoger_canton_segun_vo
to(
                partidos_politicos[3], promedio_voto, n)
            canton = random.randrange(len(lista_promedio_cant
ones))
            indicadores = obtener_inidicadores_canton(
                lista_promedio_cantones[canton], cantones)
            sexo = asignar_sexo(indicadores[23], indicadores[
24])
            voto_segunda= voto_segunda_ronda_canton("",lista_
promedio_cantones[canton])
            tempo = [[lista_promedio_cantones[canton]] + indi
cadores + [sexo] + [edad]+ [partidos_politicos[3]]+[voto_segu
nda]]
            votantes_simulados += tempo
            tempo = []
        elif (numero > suma_partes_lista(votos[:4]) and numer
o <= suma_partes_lista(votos[:5])):
            partidos[4] += 1
            lista_promedio_cantones = escoger_canton_segun_vo
to(
                partidos_politicos[4], promedio_voto, n)
            canton = random.randrange(len(lista_promedio_cant
ones))
            indicadores = obtener_inidicadores_canton(
                lista_promedio_cantones[canton], cantones)
            sexo = asignar_sexo(indicadores[23], indicadores[
```

```
24])
            voto_segunda= voto_segunda_ronda_canton("",lista_
promedio_cantones[canton])
            tempo = [[lista_promedio_cantones[canton]] + indi
cadores + [sexo] + [edad]+ [partidos_politicos[4]]+[voto_segu
nda]]
            votantes_simulados += tempo
            tempo = []
        elif (numero > suma_partes_lista(votos[:5]) and numer
o <= suma_partes_lista(votos[:6])):
            partidos[5] += 1
            lista_promedio_cantones = escoger_canton_segun_vo
to(
                partidos_politicos[5], promedio_voto, n)
            canton = random.randrange(len(lista_promedio_cant
ones))
            indicadores = obtener_inidicadores_canton(
                lista_promedio_cantones[canton], cantones)
            sexo = asignar_sexo(indicadores[23], indicadores[
24])
            voto_segunda= voto_segunda_ronda_canton("",lista_
promedio_cantones[canton])
            tempo = [[lista_promedio_cantones[canton]] + indi
cadores + [sexo] + [edad]+ [partidos_politicos[5]]+[voto_segu
nda]]
            votantes_simulados += tempo
            tempo = []
        elif (numero > suma_partes_lista(votos[:6]) and numer
```

```
o <= suma_partes_lista(votos[:7])):
            partidos[6] += 1
            lista_promedio_cantones = escoger_canton_segun_vo
to(
                partidos_politicos[6], promedio_voto, n)
            canton = random.randrange(len(lista_promedio_cant
ones))
            indicadores = obtener_inidicadores_canton(
                lista_promedio_cantones[canton], cantones)
            sexo = asignar_sexo(indicadores[23], indicadores[
24])
            voto_segunda= voto_segunda_ronda_canton("",lista_
promedio_cantones[canton])
            tempo = [[lista_promedio_cantones[canton]] + indi
cadores + [sexo] + [edad]+ [partidos_politicos[6]]+[voto_segu
nda]]
            votantes_simulados += tempo
            tempo = []
        elif (numero > suma_partes_lista(votos[:7]) and numer
o <= suma_partes_lista(votos[:8])):
            partidos[7] += 1
            lista_promedio_cantones = escoger_canton_segun_vo
to(
                partidos_politicos[7], promedio_voto, n)
            canton = random.randrange(len(lista_promedio_cant
ones))
            indicadores = obtener_inidicadores_canton(
                lista_promedio_cantones[canton], cantones)
```

```
            sexo = asignar_sexo(indicadores[23], indicadores[
24])
            voto_segunda= voto_segunda_ronda_canton("",lista_
promedio_cantones[canton])
            tempo = [[lista_promedio_cantones[canton]] + indi
cadores + [sexo] + [edad]+ [partidos_politicos[7]]+[voto_segu
nda]]
            votantes_simulados += tempo
            tempo = []
        elif (numero > suma_partes_lista(votos[:8]) and numer
o <= suma_partes_lista(votos[:9])):
            partidos[8] += 1
            lista_promedio_cantones = escoger_canton_segun_vo
to(
                partidos_politicos[8], promedio_voto, n)
            canton = random.randrange(len(lista_promedio_cant
ones))
            indicadores = obtener_inidicadores_canton(
                lista_promedio_cantones[canton], cantones)
            sexo = asignar_sexo(indicadores[23], indicadores[
24])
            voto_segunda= voto_segunda_ronda_canton("",lista_
promedio_cantones[canton])
            tempo = [[lista_promedio_cantones[canton]] + indi
cadores + [sexo] + [edad]+ [partidos_politicos[8]]+[voto_segu
nda]]
            votantes_simulados += tempo
            tempo = []
```

```
        elif (numero > suma_partes_lista(votos[:9]) and numer
o <= suma_partes_lista(votos[:10])):
            partidos[9] += 1
            lista_promedio_cantones = escoger_canton_segun_vo
to(
                partidos_politicos[9], promedio_voto, n)
            canton = random.randrange(len(lista_promedio_cant
ones))
            indicadores = obtener_inidicadores_canton(
                lista_promedio_cantones[canton], cantones)
            sexo = asignar_sexo(indicadores[23], indicadores[
24])
            voto_segunda= voto_segunda_ronda_canton("",lista_
promedio_cantones[canton])
            tempo = [[lista_promedio_cantones[canton]] + indi
cadores + [sexo] + [edad]+ [partidos_politicos[9]]+[voto_segu
nda]]
            votantes_simulados += tempo
            tempo = []
        elif (numero > suma_partes_lista(votos[:10]) and nume
ro <= suma_partes_lista(votos[:11])):
            partidos[10] += 1
            lista_promedio_cantones = escoger_canton_segun_vo
to(
                partidos_politicos[10], promedio_voto, n)
            canton = random.randrange(len(lista_promedio_cant
ones))
            indicadores = obtener_inidicadores_canton(
```

```
                lista_promedio_cantones[canton], cantones)
            sexo = asignar_sexo(indicadores[23], indicadores[
24])
            voto_segunda= voto_segunda_ronda_canton("",lista_
promedio_cantones[canton])
            tempo = [[lista_promedio_cantones[canton]] + indi
cadores + [sexo] + [edad]+ [partidos_politicos[10]]+[voto_seg
unda]]
            votantes_simulados += tempo
            tempo = []
        elif (numero > suma_partes_lista(votos[:11]) and nume
ro <= suma_partes_lista(votos[:12])):
            partidos[11] += 1
            lista_promedio_cantones = escoger_canton_segun_vo
to(
                partidos_politicos[11], promedio_voto, n)
            canton = random.randrange(len(lista_promedio_cant
ones))
            indicadores = obtener_inidicadores_canton(
                lista_promedio_cantones[canton], cantones)
            sexo = asignar_sexo(indicadores[23], indicadores[
24])
            voto_segunda= voto_segunda_ronda_canton("",lista_
promedio_cantones[canton])
            tempo = [[lista_promedio_cantones[canton]] + indi
cadores + [sexo] + [edad]+ [partidos_politicos[11]]+[voto_seg
unda]]
            votantes_simulados += tempo
```

```
            tempo = []
        elif (numero > suma_partes_lista(votos[:12]) and nume
ro <= suma_partes_lista(votos[:13])):
            partidos[12] += 1
            lista_promedio_cantones = escoger_canton_segun_vo
to(
                partidos_politicos[12], promedio_voto, n)
            canton = random.randrange(len(lista_promedio_cant
ones))
            indicadores = obtener_inidicadores_canton(
                lista_promedio_cantones[canton], cantones)
            sexo = asignar_sexo(indicadores[23], indicadores[
24])
            voto_segunda= voto_segunda_ronda_canton("",lista_
promedio_cantones[canton])
            tempo = [[lista_promedio_cantones[canton]] + indi
cadores + [sexo] + [edad]+ [partidos_politicos[12]]+[voto_seg
unda]]
            votantes_simulados += tempo
            tempo = []
        elif (numero > suma_partes_lista(votos[:13]) and nume
ro <= suma_partes_lista(votos[:14])):
            partidos[13] += 1
            lista_promedio_cantones = escoger_canton_segun_vo
to(
                partidos_politicos[13], promedio_voto, n)
            canton = random.randrange(len(lista_promedio_cant
ones))
```

```
            indicadores = obtener_inidicadores_canton(
                lista_promedio_cantones[canton], cantones)
            sexo = asignar_sexo(indicadores[23], indicadores[
24])
            voto_segunda= voto_segunda_ronda_canton("",lista_
promedio_cantones[canton])
            tempo = [[lista_promedio_cantones[canton]] + indi
cadores + [sexo] + [edad]+ [partidos_politicos[13]]+[voto_seg
unda]]
            votantes_simulados += tempo
            tempo = []
        elif (numero > suma_partes_lista(votos[:14]) and nume
ro <= suma_partes_lista(votos)):
            partidos[14] += 1
            lista_promedio_cantones = escoger_canton_segun_vo
to(
                partidos_politicos[14], promedio_voto, n)
            canton = random.randrange(len(lista_promedio_cant
ones))
            indicadores = obtener_inidicadores_canton(
                lista_promedio_cantones[canton], cantones)
            sexo = asignar_sexo(indicadores[23], indicadores[
24])
            voto_segunda= voto_segunda_ronda_canton("",lista_
promedio_cantones[canton])
            tempo = [[lista_promedio_cantones[canton]] + indi
cadores + [sexo] + [edad]+ [partidos_politicos[14]]+[voto_seg
unda]]
```

```
            votantes_simulados += tempo
            tempo = []
    # print(partidos)
    return votantes_simulados
```

Regresión lineal.

Para el desarrollo del modelo lineal se utiliza el algoritmo de regresión logística ; los datos que se utilizan para entrenar este algoritmo eson aquellos que se generan en el simulador explicado anteriormente. Este modelo utiliza las siguientes librerías:

- **pandas:** Es un paquete de Python que proporciona estructuras de datos similares a los dataframes de R. Pandas depende de Numpy, la biblioteca que añade un potente tipo matricial a Python. Esta biblioteca se utiliza principalmente para abrir archivos .csv.
- **numpy:** Es una biblioteca que se utiliza para el manejo de vectores y matrices.
- **sklearn:** Esta es una biblioteca que se utiliza para aprendizaje automatizado.Esta biblioteca se utiliza para crear el modelo lineal.
- **matplotlib:** Es una biblioteca utilizada para generar gráficos a partir de vectores.
- **seaborn:** Es una librería para visualizar los datos, se basa en matplotlib .

  Este modelo cuenta con seis funciones; las cuáles se encargan de predecir los votos por de la primera y segunda ronda para: provincia, país. Además hay una función que realiza ambas

predicciones.

Para crear el modelo se utiliza *LogisticRegression(loss)* y para entrenar dicho modelo se utiliza *fit*; para predecir un los datos de prueba se utiliza *predict*.

En este modelo se busca ver que tan preciso es el modelo a la hora de realiar las pruebas; para eso se obtiene la exactitud(accuracy) y la perdida.

A continuación; se muestra el código utilizado para realizar la predicción de las dos rondas electorales a nivel nacional.

```
def regresion_logistica_segunda_y_primera_pais(loss,cantidad_
muestra):
    x1 = generar_muestra_pais_con_segunda(cantidad_muestra)
    y = dividir_entradas_salidas2(x1)
    x1 = escribir_csv2(x1)

    data = pd.read_csv('testfile.csv', encoding='cp1252')
    data = data.dropna()
    data2 = pd.get_dummies(data, columns =['CANTON', 'Poblaci
on_total', 'Superficie (km2)', 'Densidad de población','Porce
ntaje de alfabetismo', 'Porcentaje de población ocupada no as
egurada', 'Porcentaje de población nacida en el extranjero',
'Porcentaje de población con discapacidad', 'Porcentaje de po
blación no asegurada', 'Porcentaje de hogares con jefatura fe
menina', 'Porcentaje de hogares con jefatura compartida','Sex
o','Edad','Voto primera','PROVINCIA'])
    data2.columns
```

```
    X = data2.iloc[:,1:]
    y = data2.iloc[:,0]

    X_train, X_test, y_train, y_test = train_test_split(X, y,
random_state=0)

    classifier = LogisticRegression(loss)
    classifier.fit(X_train, y_train)
    y_pred = classifier.predict(X_test)
    from sklearn.metrics import confusion_matrix

    confusion_matrix = confusion_matrix(y_test, y_pred)
    print('Accuracy of logistic regression classifier on test
set: {:.2f}'.format(classifier.score(X_test, y_test)))
```

Redes neuronales.

Para la creación de la red neuronal fue necesario la utilización de diversas librerías como:

- **Numpy:** Librería utilizada para la lectura de archivos CSV y manipulación de las matrices con los datos necesarios para la creación del modelo.
- **Keras:** Librería utilizada para la creación del modelo de red neuronal, también se utilizó para la división del sub-conjunto de entrenamiento y el sub-conjunto de pruebas.
  Como parte del proceso de la implementación de la red neuronal

encargada de la predicción de votos a partir del simulador previamente mencionado, se realizaron varios cambios a la estructura de la lista retornada por el simulador, esto con el fin de poder utilizar keras. Los cambios realizados consisten en cambiar cada atributo de tipo *“str”* por un valor numérico, ya que queras solo permite el procesamiento de valores numéricos (int, float).

Atributos por cambiar.

- **Provincias:** Para cambiar el valor original de las provincias se realizó un mapeo entre provincias y un número en específico como se nuestra a continuación.

- **Cantones:** Para cambiar el valor original de los cantones se realizó un mapeo entre cantón y un número en específico como se nuestra a continuación.

- **Partidos:** Para cambiar el valor original de los partidos se realizó un mapeo entre partido y un número en específico como se nuestra a continuación.

- **Sexo:** Para cambiar el valor original del sexo se realizó un mapeo entre hombre y/o mujer con un número en específico (mujer = 0, hombre = 1).

  Para la red neuronal utiliza un 80% de los datos para entrenamiento y un 20% para pruebas, para crear esta división se utiliza la función

```
train_test_split(x, y, test_size=0.25, random_state=0)
```

Donde el primer parámetro es el conjunto de datos que ayudan a predecir, el segundo son los datos a predecir, el tercero el porcentaje de datos que se quiere para los conjuntos de prueba y el cuarto es el grado de aleatoriedad al seleccionar los conjuntos.

Creación red neuronal.

La red implementada cuenta con dos capas “hidden” y una capa de salida.

```
    model = Sequential()
    model.add(Dense(64, activation='softmax', input_dim=8))
    model.add(Dropout(0.5))
    model.add(Dense(64, activation='softmax'))
    model.add(Dropout(0.5))
    model.add(Dense(150, activation='softmax'))
```

Para la creación se utiliza *"sequential()”* para indicar que es un red neuronal secuencial y con full conexión.
Como podemos ver las dos capas intermedias cuentan con 64 nodos cada una y la función de activación *“softmax”*. Se elige softmax como función de activación ya que es la que mejor resultados nos brinda.

Aprendizaje red neuronal.

```
model.compile(loss='categorical_crossentropy',
```

```
                    optimizer='adam',
                    metrics=['accuracy'])
```

Para el aprendizaje del modelo se utilizó: Categorical_crossentropy como función de perdida, “adam” como optimizador, esto debido a que Adam combina dos técnicas, por lo que era mejor para nuestro proyecto. Por último, se especifica que la métrica a utilizar es la exactitud.

Entrenamiento red neuronal.

Para entrenar la red se utiliza *“.fit()”*.

```
model.fit(x_train, y_train,
                epochs=1000,
                batch_size=128, verbose=0)
```

Evaluacion red neuronal.

Para evaluar la red se utiliza la siguiente función:

```
 score = model.evaluate(x_test, y_test, verbose=0, batch_size
=128)
```

Funciones.

```
def red_neuronal_segunda_y_primera_pais(cantidad_muestra):
    x1 = generar_muestra_pais_con_segunda(cantidad_muestra)
    y = dividir_entradas_salidas2(x1)
```

```
    x1 = escribir_csv4(x1)
    from numpy import genfromtxt
    my_data = genfromtxt('testfile.csv', delimiter=',', encod
ing='cp1252')
    x = []
    y = []
    for i in my_data:
        x += [[i[-1], i[1], i[-2], i[-4], i[-3], i[4], i[5],
i[8], i[9]]]
        y += [[i[0]]]
    x = np.array(x)
    y = np.array(y)

    x_train, x_test, y_train, y_test = train_test_split(
        x, y, test_size=0.25, random_state=0)
    y_train = keras.utils.to_categorical(y_train, num_classes
=150)
    y_test = keras.utils.to_categorical(y_test, num_classes=1
50)

    model = Sequential()
    model.add(Dense(64, activation='softmax', input_dim=9))
    model.add(Dropout(0.5))
    model.add(Dense(64, activation='softmax'))
    model.add(Dropout(0.5))
    model.add(Dense(150, activation='softmax'))

    model.compile(loss='categorical_crossentropy',
```

```
                    optimizer='adam',
                    metrics=['accuracy'])

    model.fit(x_train, y_train,
              epochs=1000,
              batch_size=128, verbose=0)
    score = model.evaluate(x_test, y_test, verbose=0, batch_s
ize=128)
    print("Loss:", score[0], " Accuracy:", score[1])
    return
```

Árbol de decisión.

Un árbol de decisión, que proporciona un conjunto de reglas que se van aplicando sobre los ejemplos nuevos para decidir qué clasificación es la más adecuada a sus atributos.
Un árbol de decisión está utiiza los siguientes términos:
-**Nodo de decisión:** Está asociado a uno de los atributos y tiene 2 o más ramas que salen de él, cada una de ellas representando los posibles valores que puede tomar el atributo asociado. De alguna forma, un nodo de decisión es como una pregunta que se le hace al ejemplo analizado, y dependiendo de la respuesta que de, el flujo tomará una de las ramas salientes.
-**Nodo-respuesta:** Está asociado a la clasificación que se quiere proporcionar, y nos devuelve la decisión del árbol con respecto al ejemplo de entrada.
-**Entropía:** La entropía se refiere a la forma en que se mide el grado de

incertidumbre de una muestra; para calcular la entropía se utliza la siguiente fórmula: $E(S)=\sum Ci=1-pilog2(pi)$.
Es umportante recalcar que si la entropía de algún atributo es igual a 0 ese será el nodo-respuesta.
-**Ganancia:** La ganancia es una medida de discriminación, un indicador del siguiente atributo a ser seleccionado para continuar con el proceso de división, discriminando el atributo seleccionado entre los demás atributos aún no clasificados, y se calcula utilizando la fórmula: $Gain(T,X)=E(T)-E(T,X)$.
El atributo con mayor ganacia se convierte en un nodo de desición.

Algoritmo implementado:

1. **Obtener la ganancia:** Evalúa las divisiones en el conjunto de datos. Una división en el conjunto de datos implica un atributo de entrada y un valor para ese atributo. Se puede usar para dividir los patrones de entrenamiento en dos grupos de filas. Un puntaje de ganancia da una idea de cuán buena es una división por cuán mezcladas están las clases en los dos grupos creados por la división. La ganancia para cada grupo se debe ponderar por el tamaño del grupo.
2. **Crear división:** Una división se compone de un atributo en el conjunto de datos y un valor. Es el índice de un atributo para dividir y el valor por el cual dividir filas en ese atributo; este paso se divide en dos partes:

   2.1 **Dividir un conjunto de datos:** Dividir un conjunto de datos significa separar un conjunto de datos en dos listas de filas dado el índice de un atributo y un valor dividido para ese atributo. Una vez

que se tienen los dos grupos, se puede usar el puntaje de ganancia explicado anteriormente para evaluar el costo de la división.
Dividir un conjunto de datos implica iterar sobre cada fila, verificar si el valor del atributo está por debajo o por encima del valor dividido y asignarlo al grupo izquierdo o derecho, respectivamente.
2.2 **Evaluar todas las divisiones:**. Una vez que se obtiene la ganancia y de dividir el conjunto de datos,se evalúan las divisiones. Dado un conjunto de datos, se debe verificar cada valor de cada atributo como una división candidata, evaluar el costo de la división y encontrar la mejor división posible que se pueda realizar. Una vez que se encuentra la mejor división, se puede usarla como un nodo del árbol de decisiones.
Al seleccionar la mejor división y usarla como un nuevo nodo para el árbol, se almacenará el índice del atributo elegido, el valor de ese atributo por el cual dividir y los dos grupos de datos divididos por el punto de división elegido.
Cada grupo de datos es su propio pequeño conjunto de datos de solo aquellas filas asignadas al grupo izquierdo o derecho por el proceso de división.
La función dividir() es la que itera sobre cada atributo (excepto el valor de la clase) y luego cada valor para ese atributo, dividiendo y evaluando las divisiones a medida que avanza. La mejor división se registra y luego se devuelve después de que se completan todos los controles

3. **Construír un árbol:** Para crear el nodo raíz del árbol se llamamos a la función dividir () anterior utilizando todo el conjunto de datos.
La construcción de un árbol se puede dividir en 3 partes principales:

3.1 **Nodos terminales:** Se necesita decidir cuándo dejar de crecer un árbol; para eso se utiliza la profundidad y el número de filas de las que el nodo es responsable en el conjunto de datos de capacitación.

-**Profundidad máxima del árbol:** Esta es la cantidad máxima de nodos desde el nodo raíz del árbol. Una vez que se alcanza la profundidad máxima del árbol, se debe dejar de dividir la adición de nuevos nodos. Los árboles más profundos son más complejos y es más probable que sobreabastezcan los datos de entrenamiento.

-**Registros mínimos de nodo:** Este es el número mínimo de patrones de entrenamiento de los que es responsable un nodo determinado. Una vez que esté a este mínimo o por debajo de este, se debe dejar de dividir y agregar nuevos nodos. Se espera que los nodos que representan muy pocos patrones de entrenamiento sean demasiado específicos y es probable que sobreabastezcan los datos de entrenamiento.

Es posible elegir una división en la que todas las filas pertenecen a un grupo. En este caso, no no se puede continuar dividiendo y agregando nodos secundarios ya que no se tendrán registros para dividir en un lado u otro.

3.2 **División recursiva:** La construcción de un árbol de decisión implica invocar la función dividir () desarrollada anteriormente una y otra vez en los grupos creados para cada nodo.Los nuevos nodos agregados a un nodo existente se denominan nodos secundarios. Un nodo puede tener cero hijos (un nodo-respuesta), un hijo (un lado hace una predicción directamente) o dos nodos secundarios (nodo desición).

Una vez que se crea un nodo, se pueden crear nodos secundarios recursivamente en cada grupo de datos de la división llamando nuevamente a la misma función.
A continuación se muestra una función que implementa este procedimiento recursivo. Toma un nodo como argumento así como la profundidad máxima, el número mínimo de patrones en un nodo y la profundidad actual de un nodo. Puede imaginar cómo se podría llamar primero pasar en el nodo raíz y la profundidad de 1.
Esta función se explica mejor en pasos:

- En primer lugar, los dos grupos de datos divididos por el nodo se extraen para su uso y se eliminan del nodo. A medida que se trabaja en estos grupos, el nodo ya no requiere acceso a estos datos.
- A continuación, se verifica si el grupo de filas izquierdo o derecho está vacío y, de ser así, se crea un nodo terminal utilizando los registros que se tienen.
- Luego si verifica si se ha alcanzado la profundidad máxima y si es así se crea un nodo terminal.
- Luego se procesa el elemento secundario izquierdo, creando un nodo terminal si el grupo de filas es demasiado pequeño, de lo contrario se crea y agrega el nodo izquierdo en primer lugar hasta que se llegue a la parte inferior del árbol en esta rama.
- El lado derecho se procesa de la misma manera, a medida que ascendemos una copia de seguridad del árbol construido a la raíz.

3.3 **Construyendo el árbol:** Construir el árbol implica crear el nodo raíz y llamar a la función dividir () que luego se llama recursivamente para construir todo el árbol. La función

creear_arbol () es la que implementa este procedimiento.

4. **Realizar predicciones:** Hacer predicciones con un árbol de decisión implica navegar por el árbol con la fila de datos proporcionada específicamente.Se puede implementar esto usando una función recursiva, donde se llama nuevamente a la misma rutina de predicción con los nodos secundarios izquierdo o derecho, dependiendo de cómo la división afecta los datos proporcionados. Se Debe verificar si un nodo hijo es un valor de terminal que se devolverá como predicción, o si se trata de un nodo de diccionario que contiene otro nivel del árbol que se considerará.

Las principales funciones que se utilizan en este algoritmo son las siguiente:

-**General:** Permite conocer cuales son los puntaciones de los datos generados.

```
def general(dataset, algorithm, cortes, *args):
    tempo = divisiones_cross_validation(dataset, cortes)
    puntuaciones = []
    for i in tempo:
        train_set = list(tempo)
        train_set.remove(i)
        train_set = sum(train_set, [])
        test_set = []
        for j in i:
            j_copy = list(j)
            test_set.append(j_copy)
```

```
            j_copy[-1] = None
        predicted = algorithm(train_set, test_set, *args)
        actual = [j[-1] for j in i]
        exact = exactitud(actual, predicted)
        puntuaciones.append(exact)
    return puntuaciones
```

-**divisiones_cross_validation:** Emplea el metodo cross validation para obtener las divisiones correspondientes de los datos de entrada.

```
def divisiones_cross_validation(dataset, cortes):
    dataset_split = []
    copia_dataset = list(dataset)
    tamano_corte = int(len(dataset) / cortes)
    for i in range(cortes):
        corte = []
        while len(corte) < tamano_corte:
            indice = randrange(len(copia_dataset))
            corte.append(copia_dataset.pop(indice))
        dataset_split.append(corte)
    return dataset_split
```

-**Ganancia:** Permite calcular la ganancia que se obtine de cada dato.

```
def ganancia(grupos, tipos):
    cortes = float(sum([len(subgrupo) for subgrupo in grupos]
))
```

```
    result = 0.0
    for subgrupo in grupos:
        largo = float(len(subgrupo))
        if largo == 0:
            continue
        puntuacion = 0.0
        for i in tipos:
            puntos = [j[-1] for j in subgrupo].count(i) / lar
go
            puntuacion += puntos * puntos
        result += (1.0 - puntuacion) * (largo / cortes)
    return result
```

-**Exactitud:** Está función mide el accuracy de los datos.

```
def exactitud(actual, predicted):
    correctos = 0
    for i in range(len(actual)):
        if actual[i] == predicted[i]:
            correctos += 1
    return correctos / float(len(actual)) * 100.0
```

Knn: K-D tree.

El modelo para kNN es todo el conjunto de datos de entrenamiento. Cuando se requiere una predicción para una instancia de datos no vista, el algoritmo kNN buscará en el conjunto de datos de capacitación las k

instancias más similares. El atributo de predicción de las instancias más similares se resume y se devuelve como la predicción para la instancia no vista.
La medida de similitud depende del tipo de datos. Para datos de valor real, se puede usar la distancia euclidiana. Se pueden usar otros tipos de datos, como los datos categóricos o binarios, la distancia de Hamming.
En el caso de problemas de regresión, se puede devolver el promedio del atributo predicho. En el caso de la clasificación, la clase más prevalente puede ser devuelta.
Este algoritmo pertenece a la familia de algoritmos de aprendizaje competitivo y perezoso basados en instancias.
Los algoritmos basados en instancias son aquellos algoritmos que modelan el problema usando instancias de datos (o filas) para tomar decisiones predictivas. El algoritmo kNN es una forma extrema de métodos basados en instancias porque todas las observaciones de entrenamiento se conservan como parte del modelo.
Es un algoritmo de aprendizaje competitivo, porque internamente utiliza la competencia entre los elementos del modelo (instancias de datos) para tomar una decisión predictiva. La medida de similitud objetivo entre las instancias de datos hace que cada instancia de datos compita para “ganar” o sea más similar a una instancia de datos no vista dada y contribuya a una predicción.
El aprendizaje lento se refiere al hecho de que el algoritmo no crea un modelo hasta el momento en que se requiere una predicción. Es flojo porque solo funciona en el último segundo. Esto tiene el beneficio de incluir solo datos relevantes para los datos no vistos, llamados modelos localizados. Una desventaja es que puede ser computacionalmente costoso repetir búsquedas iguales o similares en conjuntos de datos de

entrenamiento más grandes.
Finalmente, kNN es poderoso porque no asume nada sobre los datos, salvo que una medida de distancia se puede calcular de manera consistente entre dos instancias. Como tal, se llama no paramétrico o no lineal ya que no asume una forma funcional.

Algoritmo implementado.

1. **Manejar datos:** Lo primero que se debe hacer es cargar el archivo de datos.UNa vez realizada la carga del archivo,se deben dividir los datos en un conjunto de datos de entrenamiento que kNN pueda usar para hacer predicciones y un conjunto de datos de prueba que podamos usar para evaluar la precisión del modelo.
2. **Similitud:** Para hacer predicciones se necesita calcular la similitud entre dos instancias de datos dadas. Esto es necesario para que se puedan ubicar las k instancias de datos más similares en el conjunto de datos de entrenamiento para un miembro dado del conjunto de datos de prueba y, a su vez, hacer una predicción.
   Para calcular la similitud se va a utilizar la distancia Eucladiana; esto se define como la raíz cuadrada de la suma de las diferencias cuadradas entre las dos matrices de números.
3. **Vecinos:** Una vez que se tiene la medida de similitud, se puede usar para recopilar las k instancias más similares para una determinada instancia no vista.
   Este es un proceso directo de calcular la distancia para todas las instancias y seleccionar un subconjunto con los valores de distancia más pequeños.Pare esta funcionalidad se va a utilizar la función *obtenerVecinos()*.

4. **Respuesta:** Una vez que se han localizado los vecinos más similares para una instancia de prueba, la siguiente tarea es diseñar una respuesta pronosticada basada en esos vecinos; para es se permite que cada vecino vote por su atributo de clase, y tomar el voto de la mayoría como la predicción.
5. **Precisión:** Para evaluar la precisión del modelo se va a calcular una proporción de las predicciones correctas totales entre todas las predicciones realizadas,denominada precisión de clasificación; para esto se utiiza la función *obtenerExactitud()*.
   Las principales funciones de este algoritmo son:
   -**distanciaEuclideana:** Es la función que se encrga de obtener la distancia euclideana para los puntos.

```
def distanciaEuclideana(x1, x2, largo):
    distancia = 0
    for i in range(largo):
        distancia += pow((x1[i] - x2[i]), 2)
    return math.sqrt(distancia)
```

-**obtenerVecinos:** Obtiene los vecinos par los datos de training y test.

```
def obtenerVecinos(entrenamiento, testeo, k):
    distancias = []
    largo = len(testeo) - 1
    for x in range(len(entrenamiento)):
        dist = distanciaEuclideana(testeo, entrenamiento[x],
largo)
```

```
        distancias.append((entrenamiento[x], dist))
    distancias.sort(key=operator.itemgetter(1))
    vecinos = []
    for x in range(k):
        vecinos.append(distancias[x][0])
    return vecinos
```

-**obtenerRespuesta:** Permite obtener el conjunto de datos con vecinos asignados
y tenerlos ordenados.

```
def obtenerRespuesta(vecinos):
    votos = {}
    for x in range(len(vecinos)):
        respuesta = vecinos[x][-1]
        if respuesta in votos:
            votos[respuesta] += 1
        else:
            votos[respuesta] = 1
    votosOrdenados = sorted(
        votos.items(),
        key=operator.itemgetter(1),
        reverse=True)
    return votosOrdenados[0][0]
```

-**obtenerExactitud:** Permite calcular el accuracy para los datos.

```
def obtenerExactitud(testeo, predicciones):
    correctos = 0
    for x in range(len(testeo)):
        if testeo[x][-1] == predicciones[x]:
            correctos += 1
    return (correctos / float(len(testeo))) * 100.0
```

Resultados.

A continuación se muestran los resultados obtenidos después de ejecutar 5 veces cada algoritmo.

Regresión lineal.

En los gráficos que se muestran a continuación se puede ver la diferencia entre *l1* y *l2* con una muestra de 10 000.
Es importante recordar que la fórmula para la función de pérdida l1 es:

$$S = \sum_{i=1}^{n} |y_i - f(x_i)|.$$

Mientras que para l2 es:

$$S = \sum_{i=1}^{n} (y_i - f(x_i))^2$$

A continuación, se puede observar que la diferencia entre *l1* y *l2* en todas ejecuciones realizadas son muy similares.

Primera ronda por país.

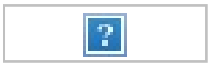
Primera ronda por provincia.

Segunda ronda por país.

Segunda ronda por provincia.

Primera y segunda ronda por país.

Primera y segunda ronda por provincia.

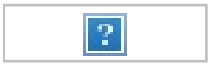


Redes neuronales.

Los siguientes datos muestran las diferencias que se obtienen entre ala exactitud (accuracy) y la pérdida (loss) a la hora de predecir los votos con la red neuronal después de ejecutar el modelo 5 veces.

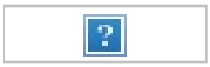
Red neuronal primer ronda por país.

A la hora de predecir los votos de la primera ronda a nivel de provincia; se puede observar que para los datos generados a nivel nacional la pérdida es en promedio 1.9; lo cuál es una pérdida pequeña. Por otro lado se puede observar que la exatitud es un promedio de 23.6 lo cuál está por encima del valor esperado que es de 6.7



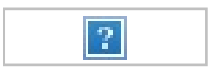
Red neuronal primer ronda por provincia.

A la hora de predecir los votos de la primera ronda a nivel de provincia; se puede observar que para los datos generados para la provincia de San José la pérdida es en promedio 1.9; lo cuál es una pérdida pequeña. Por otro lado se puede observar que la exatitud es un promedio de 23.6 lo cuál está por encima del valor esperado que es de 6.7



Red neuronal segunda ronda por país.

A la hora de predecir los votos de la segunda ronda a nivel nacional, se puede observar que a diferencia de la primera ronda, la pérdida es mucho menor que la primera ronda mientras que la exactitud es un valor mucho más alto al espereado que es un 50% debidoa que solose predicen dos partidos.



Red neuronal segunda ronda por provincia.

A la hora de predecir los votos de la primera ronda a nivel de provincia; se

puede observar que para los datos generados para la provincia de Cartago,la pérdida es mucho menor que la primera ronda, mientras que la exactitud es un valor mucho más alto al espereado que es un 50% debidoa que solose predicen dos partidos.

Red neuronal primer y segunda ronda por país.

A la hora de predecir los votos de la tanto de la primera como la segunda ronda a nivel nacional, se puede observar que a diferencia de la primera ronda individual y segunda ronda individual, la pérdida es mucho menor y la exactitud está en un promedio del 70%.

Red neuronal primer y segunda ronda por provincia.

A la hora de predecir los votos de la tanto de la primera como la segunda ronda en la provincia de Cartago, se puede observar que a diferencia de la primera ronda individual y segunda ronda individual, la pérdida es mucho menor y la exactitud está en un promedio del 70%.

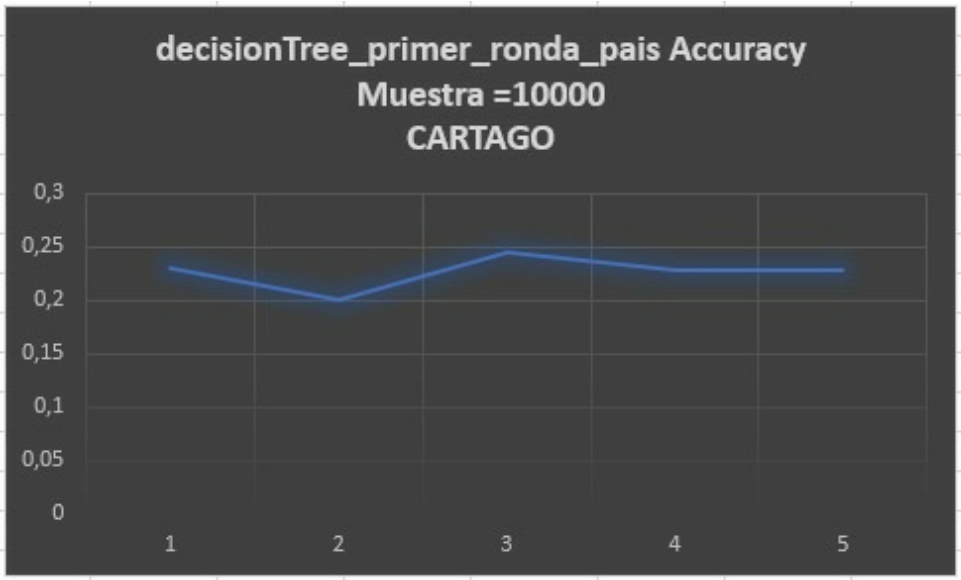
Árbol de decisión.

Árbol de decisión primer ronda.

En el momento de ejecutar este modelo 5 veces, se puede observar que la

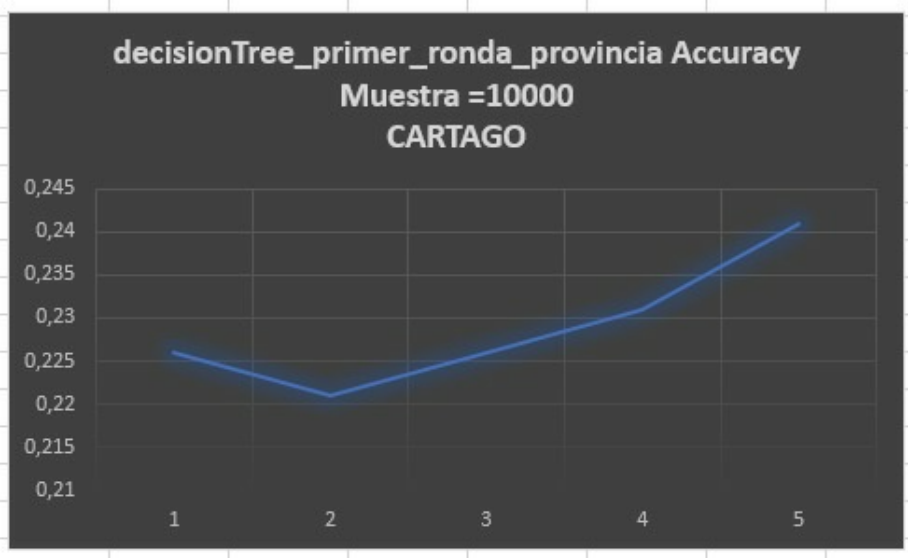
exactitud está por encima del valor esperado que es aproximadamente 22%.

Árbol de decisión primer ronda por país.



	decisionTree_primer_ronda_pais
	Accuracy
1	0,23
2	0,201
3	0,244
4	0,228
5	0,228

Árbol de decisión primer ronda por provincia.

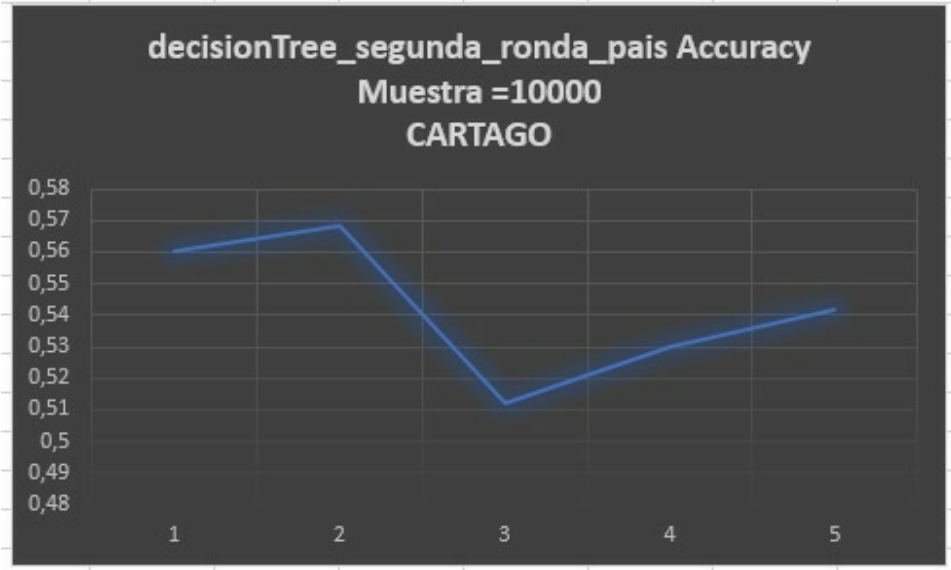


	decisionTree_primer_ronda_provincia
	Accuracy
1	0,226
2	0,221
3	0,2261
4	0,231
5	0,241

Árbol de decisión segunda ronda.

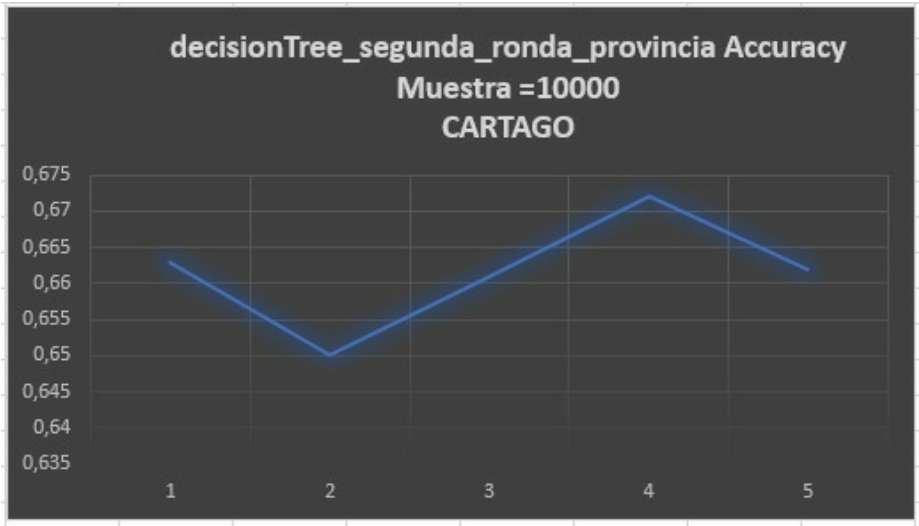
En el momento de ejecutar este modelo 5 veces, se puede observar que la exactitud está por encima del valor esperado que es aproximadamente 50%.

Árbol de decisión segunda ronda por país.



	decisionTree_segunda_ronda_pais
	Accuracy
1	0,56
2	0,568
3	0,51201
4	0,53
5	0,542

Árbol de decisión segunda ronda por provincia.

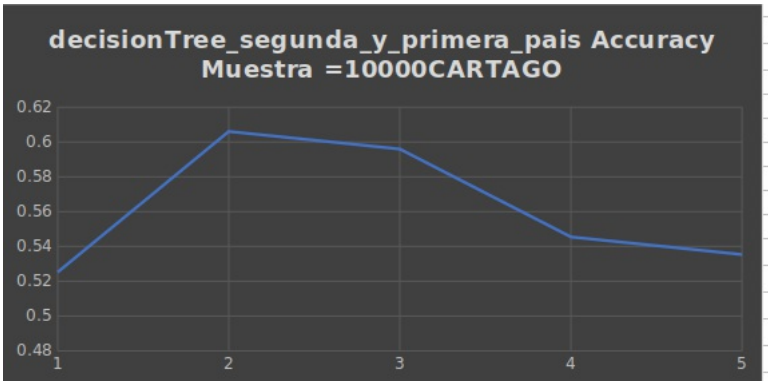


	decisionTree_segunda_ronda_provincia
	Accuracy
1	0,663
2	0,65
3	0,661
4	0,672
5	0,662

Árbol de decisión primer y segunda ronda.

En el momento de ejecutar este modelo 5 veces, se puede observar que la exactitud está por encima del valor esperado que es aproximadamente 70% ya que predice ambas rondas electorales.

Árbol de decisión primer y segunda ronda por país.



	decisionTree_segunda_y_primera_pais
	Accuracy
1	0.52525
2	0.60606
3	0.59596
4	0.54545
5	0.53535

Árbol de decisión primer y segunda ronda por provincia.

	decisionTree_segunda_y_primera_provincia
	Accuracy
1	0,71627907
2	0,730964467
3	0,671568627
4	0,709251101
5	0,716666667

/Imagenes/kd_tree/kd_primera_pais.PNG)

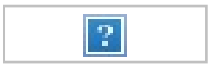
Knn: K-D tree.

Primer ronda.

En el momento de ejecutar este modelo 5 veces, se puede observar que la exactitud está por encima del valor esperado que es aproximadamente 22%.

K-D tree primer ronda por país.

K-D tree primer ronda por provincia.

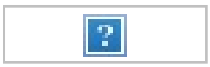


Segunda ronda.

En el momento de ejecutar este modelo 5 veces, se puede observar que la exactitud está por encima del valor esperado que es aproximadamente 50%.

K-D tree segunda ronda por país.

K-D tree segunda ronda por provincia.



Primer y segunda ronda.

En el momento de ejecutar este modelo 5 veces, se puede observar que la exactitud está por encima del valor esperado que es aproximadamente 70% ya que predice ambas rondas electorales.

K-D tree primer y segunda ronda por país.

K-D tree primer y segunda ronda por provincia.

Manual de instalación.

A continuación, se adjunta el link del manual de instalación:

https://github.com/ferAlvarado/Proyecto1_IA/blob/master/tec/ic/ia/pc1/g02/I

Link de GitHub.

A continuación, se adjunta el link al proyecto en GitHub:

https://github.com/ferAlvarado/Proyecto1_IA