Ciencia de Datos en Python
Proyecto #1

Descripción: El proyecto consiste en aplicar los conocimientos aprendidos en clase (y apoyándose de referencias adicionales útiles) para crear modelos predictivos de regresión lineal uni-variable sencillos de la forma:

$$y = \beta_0 + \beta_1 * x$$

Donde:

- y es la variable dependiente,
- x es la variable independiente,
- β_0 es el intercepto de la recta,
- β_1 es la pendiente de la recta.

Tanto x como y son parte del dataset usado , β_0 y β_1 son parámetros del modelo los cuales buscamos estimar con los datos, esto significa que se busca encontrar que valores de β_0 y β_1 producen una recta que describa de la mejor manera posible la relación entre los datos x y y.

Se trabajara con un set de datos de muchas variables y se realiza un análisis exploratorio para visualizar y analizar los datos y entender cómo se comportan, luego de esto podremos elegir las variables independientes x a trabajar(según el potencial predictivo de estas a través de medir la correlación) , esto significa que aunque el dataset posee múltiples variables (columnas), en vez de crear un modelo multi-variable crearemos múltiples modelos uni-variable.

Los datos se encuentran dados en el formato binario de NumPy .npy por lo tanto usaremos la función **load** de numpy para poderlos utilizar: `https://docs.scipy.org/doc/numpy/reference/generated/numpy.load.html`

Usando slicing sobre el dataset este deberá separarse de la siguiente forma:

- 80 % del dataset(filas) se usará para todo el proceso de entrenamiento es decir: el análisis exploratorio, selección de variables a usar, creación de modelos predictivos. Les llamamos ?set de entrenamiento?.
- 20 % del dataset(filas) se usará para probar ,validar y evaluar los modelos resultantes. Esto significa que el 20 % de data no es usado durante todo el proyecto y es usado solo al final cuando ya poseemos los modelos predictivos. Les llamamos ?set de validación y pruebas?

Los datos del proyecto podrá encontrarlos en el GES en la sección de Material de Apoyo con el nombre de **Datos para Proyecto #1**

A continuación se le listan los pasos que debería seguir para poder desarrollar su proyecto:

1. Crear un entorno de anaconda con los paquetes que considere necesarios.
2. Usando sclicing con NumPy separar los datos en 2 datasets: entrenamiento(80 %) y validación y pruebas(20 %).
3. Análisis exploratorio de datos: Para cada variable en el dataset calcular((usando numpy o pandas):
   - media
   - valor máximo
   - valor mínimo
   - rango(peak to peak, no el rango del tensor que por ser vector sabemos que es 1)
   - desviación estándar.
4. Para cada variable en el dataset usar seaborn(función distplot `https://seaborn.pydata.org/generated/seaborn.distplot.html`) para graficar un histograma de la variable.
5. Para cada variable independiente x :
   - Calcular el coeficiente de correlación entre x y y.
   - Graficar x vs y(scatterplot) usando matplotlib.
   - Colocar el coeficiente de correlación y colocarlo como parte del título de la gráfica.
   - Basado en la gráfica y el coeficiente de correlación de cada par x,y elegir las 2 variables con más potencial predictivo es decir las 2 variables que presentan mayor correlación entre dicha variable y la variable dependiente.
6. Crear una función para entrenar un modelo de regresión lineal de una variable $y = \beta_0 + \beta_1 * x$. La función recibe como argumentos:
   - 6.1 Vector con la variable independiente x,
   - 6.2 Vector con la variable dependiente y,
   - 6.3 un entero **epochs** que indica por cuantas iteraciones entrenar el modelo.
   - 6.4 un entero **imprimir_error_cada** , que nos indica cada cuantas iteraciones queremos imprimir a través de print: el número de iteración, el error del modelo en esa iteración, si imprimir_error_cada = 10, se despliega en pantalla el error en las iteraciones: 10,20,30,40,50.
   - 6.5 escalar α(learning rate): es usado como parte de la expresión matemática para actualizar en cada iteración los parámetros del modelo.
7. Para crear su función debe considerar lo siguiente:
   - Crear una matriz de 2 columnas, la primera columna corresponde al vector de datos x y la segunda columna de la matriz para todas las filas es igual a 1.
   - Inicializar los parámetros del modelo en un vector β_0 y β_1, esto es equivalente a empezar el proceso con una recta inicial la cual en cada iteración actualizaremos hasta encontrar una que aproxime de buena manera los datos x, y.

- por cada epoch(iteración) debe:
  - Calcular $\hat{y}$(predicción o estimación) para todas las observaciones de manera simultánea(vectorizada) utilizando el modelo correspondiente a la iteración(es decir , los valores de β_0 y β_1,): esto produce un vector $\hat{y}$ con el mismo número de elementos que y.

    Conceptualmente el calcular vectorizadamente la predicción para una única observación del datdataset, significaría aplicar el modelo lineal a esta observación por ejemplo, si $x = 2$, y los parámetros son $\beta_1 = 0{,}1$, $\beta_0 = 0{,}2$ tendríamos :

    $$\hat{y} = 0{,}1(2) + 0{,}2$$

    Tomando en cuenta que agregamos una columna adicional con el valor 1, podemos usar el producto punto para realizar este mismo cálculo , por ejemplo(lo siguiente no es código real solo ejemplo): observación = [2,1], parámetros = [0.1,0.2]

    $\hat{y}$ = np.dot(observacion, parametro)

    Esto es igual que : 2(0.1) + 1(0.2) = 2(0.1) + 0.2

    Por lo tanto para calcular la aproximación $\hat{y}$ de manera simultánea a todas las observaciones debemos aplicar una multiplicación matricial(matriz ,vector)
  - Calcular el error o costo usando: y ,$\hat{y}$

    $$\epsilon = \frac{1}{2n}\sum_{i=0}^{n}(y - \hat{y})^2$$

    Esto produce un escalar que indica el error (mientras más alto peor el modelo,mientras mas bajo mejor el modelo) producido por el modelo correspondiente a la iteración(es decir, el error para ciertos parámetros β_0, β_1)

  - Almacenar en un vector el error de cada iteración.
  - Calcular el gradiente del error respecto de cada parámetro con las expresiones:

    $$\frac{\delta e}{\delta \beta_1} = \frac{1}{n}\sum_{i=0}^{n}((\hat{y} - y) * x)$$

    $$\frac{\delta e}{\delta \beta_0} = \frac{1}{n}\sum_{i=0}^{n}((\hat{y} - y))$$

    Nota: Este es conceptualmente el cálculo a realizar , traducido a programación debemos hacerlo vectorizadamente ,es decir en vez de realizar 2 cálculos independientes debemos realizar un solo cálculo cuyo resultado es un vector, el primer elemento del vector es el gradiente de m y el segundo vector el gradiente de b. Para hacer esto nos podemos apoyar en el hecho de que el dataset tiene una columna con el valor de 1 para todos los elementos y la expresión para calcular el gradiente de b se puede calcular a la forma equivalente:

$$\frac{\delta e}{\delta \beta_0} = \frac{1}{n}\sum_{i=0}^{n}((\hat{y} - y)) * 1$$

- Actualizar los parámetros del modelo con la expresión:

$$\beta_1^{i+1} = \beta_1^i - \alpha * \frac{\delta e}{\delta \beta_1}$$

$$\beta_0^{i+1} = \beta_0^i - \alpha * \frac{\delta e}{\delta \beta_0}$$

  Donde: β_0 y β_1: son los parámetros del modelo $\frac{\delta e}{\delta \beta_0}$, $\frac{\delta e}{\delta \beta_1}$ del resultado del paso anterior y α es el parámetro ? enviado a la función, un valor muy grande de este puede hacer que nunca se logre la convergencia a un modelo adecuado y un valor muy pequeño puede hacer que se necesiten demasiadas iteraciones(y mucho tiempo) para obtener el modelo óptimo. Nuevamente esto es conceptualmente el cálculo a realizar, al traducirlo a programación debemos hacerlo vectorizadamente utilizando el vector de parámetros(con los valores de la iteración anterior) y el vector gradiente calculado en el paso anterior.
- Almacenar en una estructura de datos el modelo resultante(a discreción del estudiante, por ejemplo un diccionario donde la llave es el número de iteración y el valor es un vector con los parámetros).
- La función devuelve 2 resultados: La estructura de datos conteniendo el modelo de cada iteración y el error de cada iteración.
- Crear una función que nos permita visualizar con matplotlib cómo cambia el error en el tiempo: crear una función que tome como parámetro el vector de errores generados por la función de entrenamiento y grafique en el eje x el número de iteración y en el eje y el error para esa iteración.
- Crear una función que nos permita visualizar con matplotlib cómo evoluciona el modelo entrenado en el tiempo : Crear una función que tome como parámetro la estructura de datos conteniendo el historial de modelos ,y un valor ?n? que indica cada cuantas iteraciones graficar el modelo resultante, por ejemplo para n=3 la función debe graficar la recta correspondiente al modelo junto a los datos cada 3 iteraciones, si se ejecutaron 15 iteraciones , para n = 3 se grafica el modelo de las iteraciones 3,6,9,12,15 junto con los datos de entrenamiento.
- Utilizar las funciones del punto anterior para entrenar modelos de regresión lineal $y = \beta_0 + \beta_1 * x$, para cada una de las variables x elegidas, basándose en las funciones para graficar las curvas de aprendizaje y comparación del modelo vs los datos elegir el número de ?epochs? o iteraciones a entrenar el modelo(y experimentar con el α)

  Criterio: El error debe disminuir lo más posible por lo cual la curva de aprendizaje debe disminuir hasta que ya no disminuya mucho (se estabilice o converja) , si el error baja y comienza a subir debemos usar menos iteraciones.
- Para cada una de las variables x seleccionadas, usar scikit-learn para entrenar un modelo de regresión lineal: `https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html`
- Para cada variable independiente x elegida, crear una función que usando el modelo entrenado manualmente y el modelo de scikit-learn(enviados como parámetros a la

función) calcule la estimación o predicción de cada uno de estos y devuelva como resultado una predicción estimada promediando las predicciones de los 2 modelos, la función debe recibir como parámetro adicional un vector de cualquier tamaño de x y devolver 3 vectores cada uno del mismo tamaño del vector x, estos vectores son:
    - La predicción con el modelo entrenado manualmente
    - La predicción con el modelo de scikit-learn
    - La combinación(promedio) de las 2 anteriores.
- Usando el 20 % de los datos de validación, aplicamos los modelos predictivos para estimar la variable dependiente y usando los valores reales de y calculamos el error de cada modelo. Para cada variable independiente x elegida:
    - Graficar el error de cada modelo para esta variable ?x?
    - Concluir cuál modelo para la variable x es el mejor a ser usado (error mas bajo)

- Todo este desarrollo debe hacerse por medio de programación orientada a objetos, es decir que deberá crear una clase que realice todo lo mencionado y posea los campos y métodos necesarios para que funcione correctamente.

Tome en cuenta las siguientes consideraciones para desarrollar su proyecto:

1. El proyecto es individual.
2. Se utilizará la herramienta de verificación de similitud de código(plagio) para detectar copias y código bajado de internet, en caso de encontrarse plagio se anulará el proyecto.
3. Es requisito entregar y aprobar el proyecto para aprobar el curso.
4. El proyecto se trabajará usando el formato usado para las tareas prácticas: Jupyter notebook.
5. No se permite usar librerías y herramientas adicionales a las mencionadas en este enunciado:
    - Python
    - Numpy
    - Pandas
    - matplotlib
    - seaborn
    - scikit-learn
6. Usar código vectorizado y ufuncs siempre que sea posible.
7. Usar git para llevar un tracking del avance del proyecto y así mismo practicar el proceso de versionamiento. El repositorio de github debe contener al menos 5 commits para el notebook del proyecto.

Entregable: En su entrega deberá subir al GES el link a su repositorio de git donde se encuentre todo su código y una referencia a un vídeo que tenga una duración entre máxima de 5 minutos donde explica como funciona su proyecto. **La entrega será a más tardar el Domingo 17 de marzo a las 23:55PM.**