Módulo 0: Introducción a machine learning

Fecha de entrega límite: 3 de noviembre de 2021

Objetivo:
El objetivo de la presente guía práctica es comprender el algoritmo del perceptrón simple y perceptrón multicapa (MLP). A qué tipo de problemas se los puede aplicar, sus ventajas y limitaciones.

Requerimientos:
- Conocimientos de Python básico (y tener Python instalado o utilizar alguna plataforma online de Python en notebooks: Google Colaboratory, Kaggle, etc)
- Teoría de ML: regresión lineal, perceptrón, redes neuronales multicapa.
- Uso básico de Git y GitHub.

Ejercicio 1: Regresión lineal

A partir de lo visto en la clase práctica sobre regresión lineal, utilizar el dataset “ejercicio_1_precios.csv” (obtenido de la plataforma Kaggle; “House Prices - Advanced Regression Techniques”). Separar en conjuntos de entrenamiento y test. Investigar otros métodos de split de conjuntos de datos, aplicarlos al dataset. Implementar el algoritmo del descenso del gradiente en Numpy y Pytorch. Mida y compare el error de predicción.

Como variable x utilice a ‘GrLivArea' que representa los pies cuadrados habitables de la vivienda en la planta baja.
Y como variable a predecir (y) utilice el precio de venta 'SalePrice'

Ejercicio 2: Perceptrón simple

Implementar el algoritmo del Perceptrón simple en una clase que mínimamente conste de un método para entrenamiento y un método para realizar predicciones a partir de entradas dadas.

```
class Perceptron:

    def __init__(self):
        pass
```

```
    def train(self):
        pass

    def predict(self):
        pass
```

A. Entrenar un perceptrón simple para intentar resolver los problemas AND, OR y XOR utilizando los datasets que se proveen a continuación. ¿Qué conclusiones puede sacar? ¿Cuán rápida es la convergencia, y de qué depende el resultado? ¿Funciona para todos los casos? ¿Por qué?

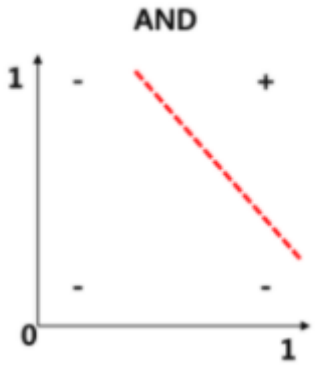
```
import numpy as np

# AND
X_train = np.array([[0,0],[0,1],[1,0],[1,1]])
y_train = np.array([[0, 0, 0, 1]]).T

# OR
X_train = np.array([[0,0],[0,1],[1,0],[1,1]])
y_train = np.array([[0, 1, 1, 1]]).T

# XOR
X_train = np.array([[0,0],[0,1],[1,0],[1,1]])
y_train = np.array([[0, 1, 1, 0]]).T
```

B. Agregue métodos a la clase Perceptrón que permita visualizar la tasa de error por época de entrenamiento y otro que permita visualizar la recta de separación entre clases que se va ajustando durante el entrenamiento.



C. Utilizando el dataset llamado “ejercicio_2c_15.csv” y “ejericicio_2c_30.csv” que se proveen en el repositorio de Github dentro de la carpeta “datasets/guia/”, vuelva a entrenar el perceptrón simple desarrollado. ¿Qué diferencias encuentra cuando entrena el perceptrón con estos datasets por separado? ¿Cuál es la velocidad de convergencia? ¿Cómo es la tasa de error con respecto al caso del punto A del presente ejercicio?

```
import pandas as pd

df = pd.read_csv('filename.csv')
```

Ejercicio 3: Perceptrón multicapa (MLP)

A. Utilizando el dataset llamado “ejercicio_3a_xor_5.csv” y “ejercicio_3a_xor_35.csv” que se proveen en el repositorio de Github dentro de la carpeta “datasets/guia/”, construya, entrene y pruebe un MLP utilizando PyTorch. Considere el número de capas, funciones de activación, etc. ¿Cómo seleccionó el número correcto de neuronas y capas del MLP? ¿Es posible?

B. Explore el dataset CIFAR10. Desarrolle, entrene y valide un perceptrón multicapa en PyTorch. Pruebe utilizar y comparar diferentes funciones de activación en la capa de salida y capas intermedias. ¿Cómo se asegura que no exista ***overfitting*** o ***underfitting***?

C. Investigue sobre el concepto de regularización. Apliquelo al punto anterior. ¿Qué conclusiones puede obtener?

Ejercicio OPCIONAL:

Implementar en Python una clase que modelice a un perceptrón multicapa, donde donde cantidad de capas, neuronas por capa, cantidad de entradas sean atributos de la clase