FACULTAD DE ARTES
UNIVERSIDAD DE CHILE

Proyecto N°1: Desafío Perceptrón

Nombre: Fernando J. Manosalva Volke

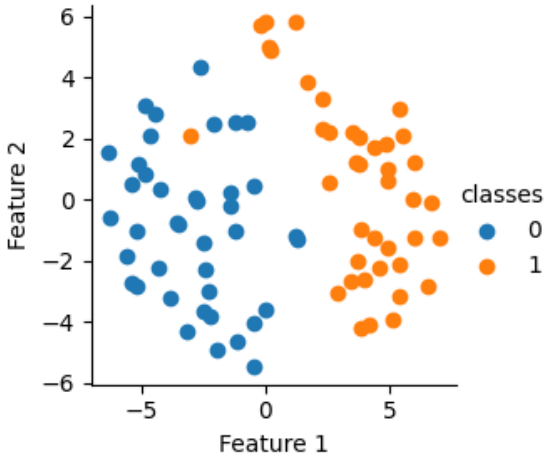
Profesor: Rodolfo A. Lobo Carrasco

Introducción

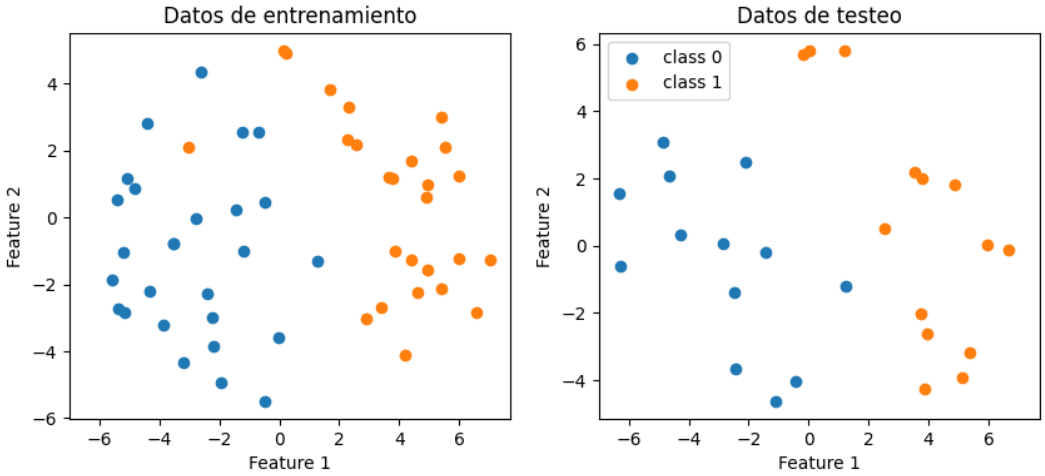
En el presente informe se expone lo que fue el proceso de clasificación de audios de dos clases distintas (bombo y caja de batería) mediante dos métodos diferentes: el primero usando un perceptrón y el segundo utilizando un modelo de regresión logística. Finalmente se comparan los resultados obtenidos con ambos modelos para así determinar cuál fue más efectivo para esta tarea.

Procesamiento de los datos

Se cargaron desde Google Drive cuarenta muestras de audio de cada clase (bombo y caja), las cuales mediante el algoritmo *t – SNE* fueron llevadas a dos dimensiones. Las muestras quedaron distribuidas de la siguiente manera:



Donde la clase 1 corresponde a BOMBO y la clase 0 corresponde a CAJA. A continuación, usando la librería *sklearn* se dividió el *dataset* en un conjunto de entrenamiento, conformado por 53 muestras, y en un conjunto de testeo, conformado por 27 muestras:



Entrenando un modelo de Perceptrón

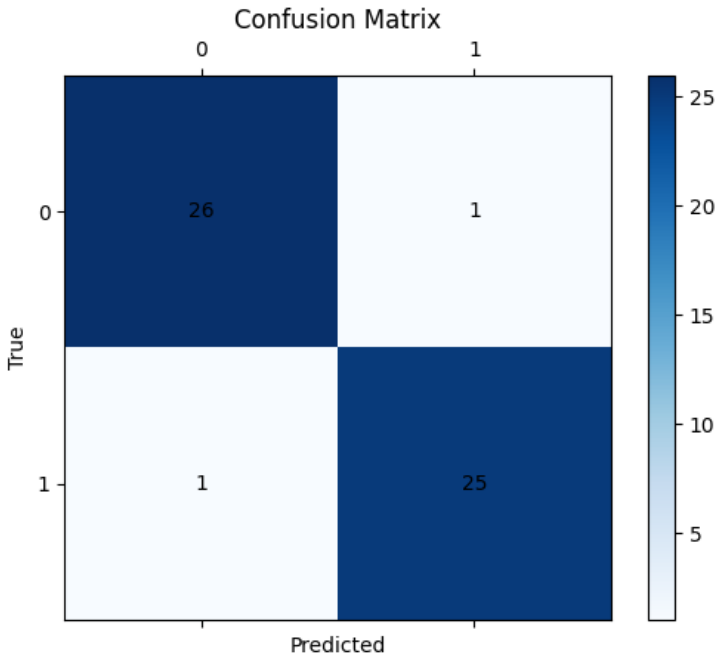
La clase de nuestro modelo fue definida de la siguiente manera:

```python
class Perceptron():
    def __init__(self, num_features = 2):
        self.num_features = num_features
        self.weights = torch.zeros(num_features, 1,
                                   dtype=torch.float32)
        self.bias = torch.zeros(1, dtype=torch.float32)

        self.ones = torch.ones(1)
        self.zeros = torch.zeros(1)

    def forward(self, x):
        linear = torch.mm(x, self.weights) + self.bias
        y_pred = torch.where(linear > 0., self.ones, self.zeros)        # funcion de activacion
        return y_pred

    def backward(self, x, y):
        y_pred = self.forward(x)
        error = y - y_pred
        return error

    def train(self, x, y, epochs, learning_rate = 0.01):

        for epoch in range(epochs):

            for i in range(y.shape[0]):

                error = self.backward(x[i].reshape(1, self.num_features), y[i]).reshape(-1)
                self.weights += (error * x[i] * learning_rate).reshape(self.num_features, 1)
                self.bias += error * learning_rate
            print(f'Epoch [{epoch+1}/{epochs}]')

    def evaluate(self, x, y):
        y_pred = self.forward(x).reshape(-1)
        accuracy = accuracy_score(y, y_pred)
        return accuracy

    def confusion_matrix(self, y, y_pred):
        cm = confusion_matrix(y, y_pred)
        return cm
```

Además de los métodos solicitados se incluyó uno que generara una matriz de confusión, una vez que las muestras hayan pasado a través de la red. Esta clase se basa en el modelo tutorial de Sebastian Raschka[1].

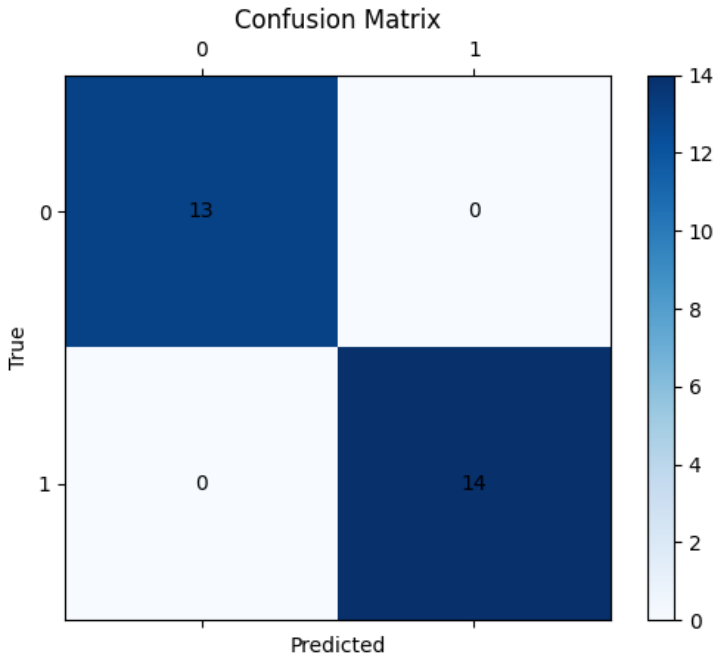
Una vez el modelo fue instanciado, se entrena con una cantidad de 20 *epochs*. La performance del modelo se midió utilizando el descriptor *accuracy* y graficando su matriz de confusión. A continuación, tenemos los gráficos correspondientes para cada uno de los conjuntos del *dataset*:

Conjunto de entrenamiento: (*accuracy* = 96.23 %)

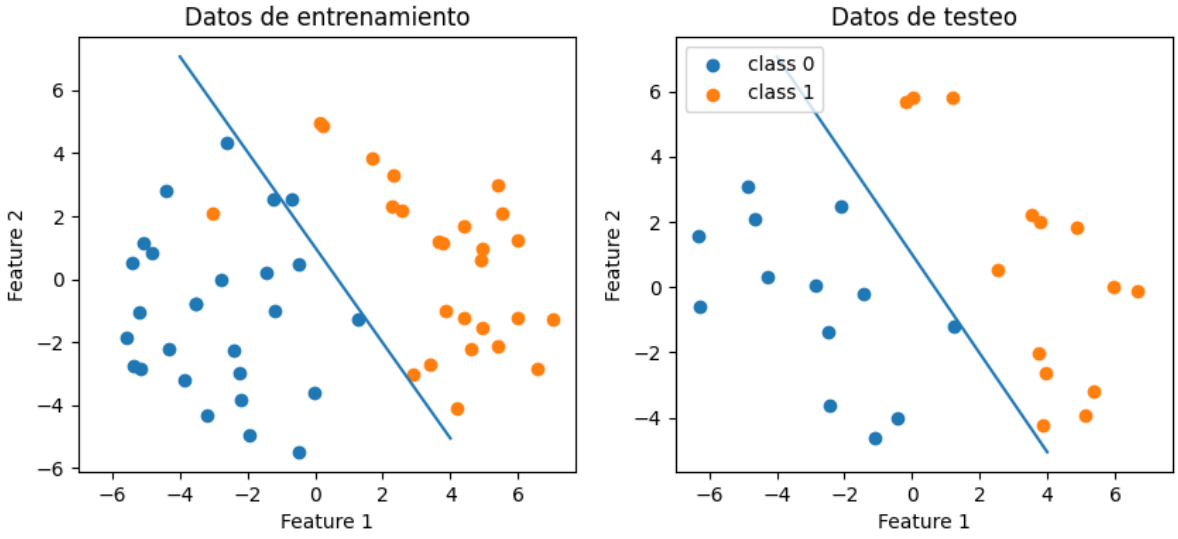


[1] https://github.com/rasbt/stat453-deep-learning-ss21/blob/main/L03/code/perceptron-pytorch.ipynb

Conjunto de testeo: (*accuracy* = 100 %)



Como referencia extra, tenemos también la distribución de las muestras, pero incluyendo la frontera de decisión que trazó el modelo según los pesos y el bias calculados:



Entrenando un modelo de Regresión Logística

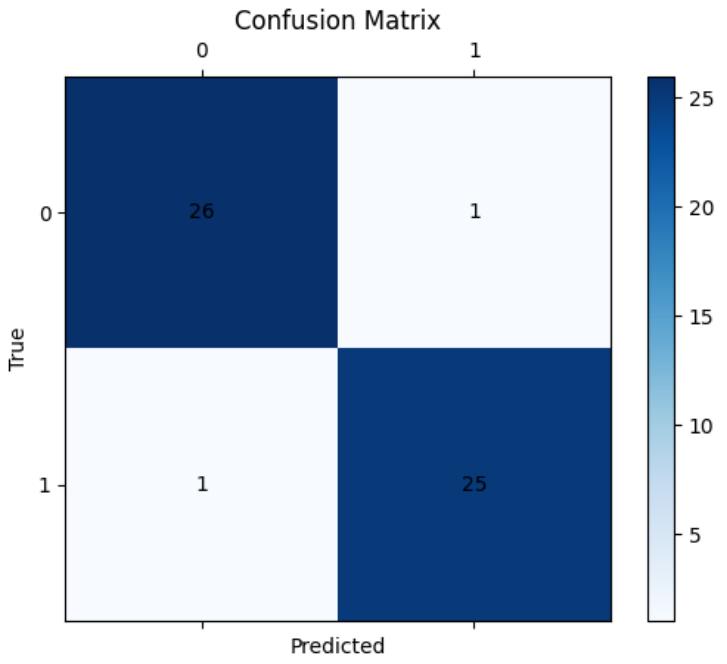
El modelo de regresión logística que se eligió es el mismo utilizado en clases:

```
[61] from sklearn.linear_model import LogisticRegression

     lr_modelo = LogisticRegression(max_iter = 1000)
     lr_modelo.fit(X_train, y_train)
```

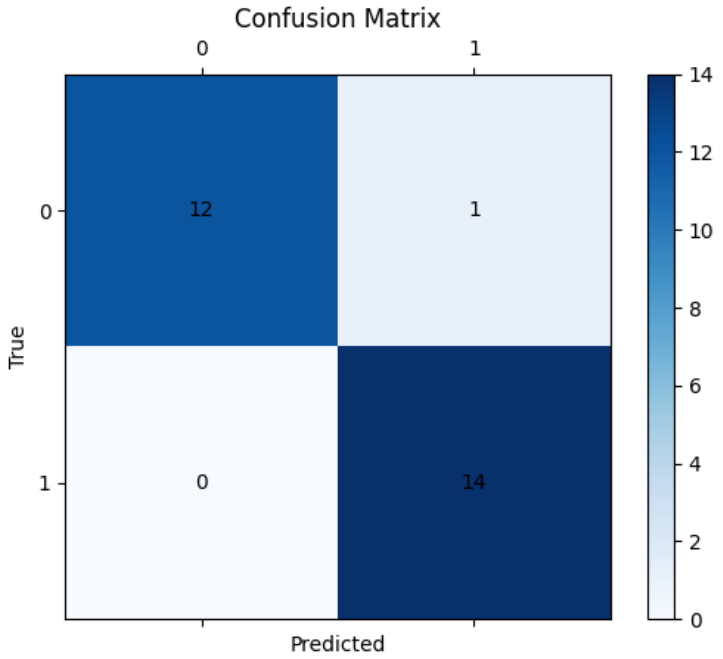
```
LogisticRegression(max_iter=1000)
```

Una vez entrenado el modelo, se midió su *accuracy* y se graficaron sus respectivas matrices de confusión usando los datos de entrenamiento y de testeo. A continuación, se exponen los gráficos:

Conjunto de entrenamiento: (*accuracy* = 96.23 %)



Conjunto de testeo: (*accuracy* = 96.30 %)

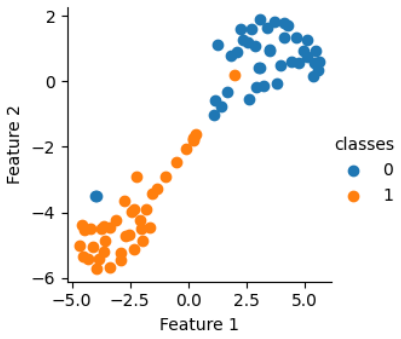


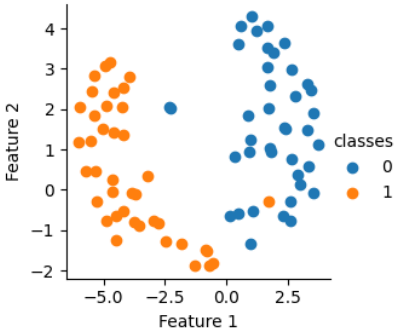
Comparación y Conclusiones

Como se pudo observar en los gráficos anteriores, en conjunto con los resultados de las mediciones utilizadas, la performance de ambos modelos al procesar los datos de entrenamiento fue la misma. Sin embargo, esta difirió con los datos de testeo, demostrando ser mejor el Perceptrón, al no equivocarse con ninguna muestra al clasificarlas.

A pesar de esto, el modelo de regresión logística solo se equivocó por una muestra.

Gracias a que el algoritmo *t – SNE* puede extraer diferentes pares de características de un mismo audio, dependiendo del número de veces que este se ejecute, es que se pudieron poner a prueba los modelos con distintas distribuciones, tales como





Al realizar distintas pruebas se observó que a veces el Perceptrón era más efectivo, y otras resultaba serlo el modelo de regresión logística. Es por todo esto que es posible afirmar que ambos modelos pueden considerarse igual de competentes para realizar esta tarea en particular, con esta cantidad de muestras. Probablemente con una mayor cantidad de muestras se pueda determinar cuál de los dos modelos podría llegar a ser más efectivo para un *dataset* bidimensional.