



Comisión Nacional
de Energía Atómica



Trabajo Práctico N° 1: Introducción a las Técnicas de Machine Learning

Aprendizaje Profundo y Redes Neuronales Artificiales

4 de septiembre de 2020

Alumnos :

Tomás LIENDRO tomas.liendro@ib.edu.ar

Docentes :

Ariel CURIALE
Germán MATO
Lucca DELLAZOPPA

San Carlos de Bariloche, Argentina

Índice

1. Ejercicio 1	1
2. Ejercicio 2	2
3. Ejercicio 3	4
4. Ejercicio 4	6
5. Ejercicio5	7
Referencias	11

1. Ejercicio 1

Para el primer ejercicio, a partir de un conjunto de datos n-dimensionales, debe implementarse una regresión lineal utilizando mínimos cuadrados, de forma que un ajuste lineal representa una aproximación razonable. El conjunto de datos se genera de forma aleatoria.

Para este ejercicio, se crea una clase *Regresion* que toma como parámetros de entrada la dimensión **n**, la cantidad de datos **cant_datos** y el tamaño del espacio de trabajo **size**. Al inicializarse esta clase, se genera un conjunto de datos *x* e *y* aleatorios de dimensiones $(cant_datos + 1) \times n$ con valores tomados entre 0 y **size**, donde se suma 1 en la primera dimensión para resolver una ecuación de la forma:

$$y = A \cdot X. \quad (1.1)$$

Para hallar la matriz de coeficientes **A** para la implementación de la regresión lineal múltiple, se utiliza la ecuación [1]:

$$A = (X^T X)^{-1} X Y. \quad (1.2)$$

El conjunto de datos generado para el caso particular de 1D, es el que se muestra en la Fig. 1. En la misma figura, se muestra la regresión lineal 1D (línea roja), cuya ecuación resulta ser:

$$y_{reg} = 3,00 \cdot x + 2,19$$

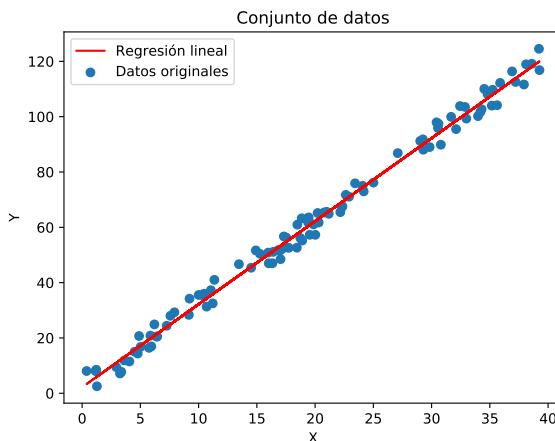


Figura 1: Conjunto de datos generados al azar para el ejercicio 1.

Finalmente, se obtiene el error de estimación calculando el Error Cuadrático Medio (ECM):

$$ECM = \frac{1}{ndatos} \sum_{i=1}^{ndatos} (Y_i - Y_{estimado_i})^2,$$

donde *ndatos* es la cantidad de datos del vector *Y*. Si se grafica el error en función de la dimensión del problema se obtiene la gráfica de la Fig. 2.

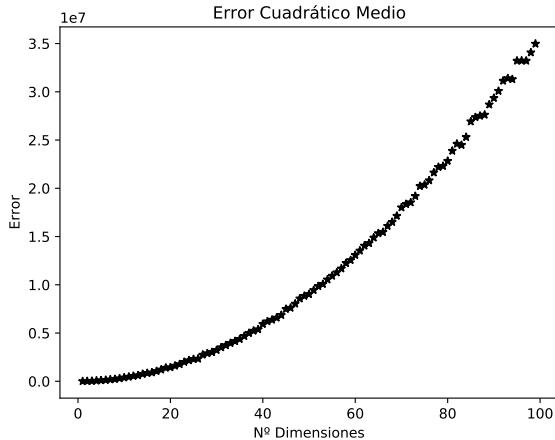


Figura 2: Error Cuadrático Medio: cantidad de datos fijos, dimensión variable.

En la Fig. 2, se ve que el error crece exponencialmente a medida que aumenta la dimensión del problema. Esto se conoce como maldición de la dimensionalidad, y conceptualmente significa que si uno toma la misma cantidad de datos *n*_{datos} aleatoriamente en un problema, cuando la dimensión del problema aumenta, la proporción de datos tomados para realizar la estimación es (porcentualmente) menor y el error de estimación aumenta exponencialmente.

El segundo efecto que se puede ver es el del aumento de la cantidad de datos frente a una dimensionalidad del problema fijo ($n=10$). En la Fig. 3 se muestra que el error de estimación (medido a lo largo de una sola dimensión) decrece a medida que se aumenta la cantidad de datos.

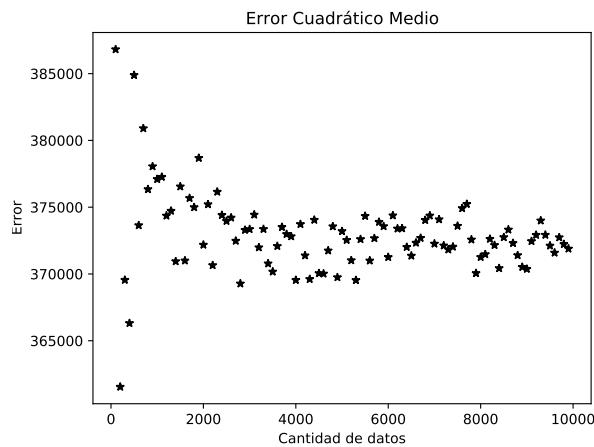


Figura 3: Error Cuadrático Medio: dimensión fija, cantidad de datos variable

2. Ejercicio 2

El segundo ejercicio consiste en la implementación del método K-means. Este método sirve para clasificar un grupo de **nd** distribuciones en **K** clases en función de la posición de las medias de los grupos. Este método implica un proceso iterativo para clasificar que sigue el siguiente algoritmo:

1. Se generan datos a partir de **nd** distribuciones normales,
2. se toman las posiciones iniciales de **K** medias, donde cada una pertenece a una clasificación distinta,

3. se calcula la distancia euclídea de cada punto a cada una de las **K** medias y se clasifica cada punto dentro de la clase cuya media se encuentra más cerca,
4. una vez que todos los datos fueron clasificados, se recalcula el valor de la media de cada clase como el promedio de los datos,
5. se vuelven a calcular las distancias de los puntos a las medias de las clases y se reclasifican,
6. se itera el proceso desde el paso 3 hasta la convergencia de la clasificación.

Para el caso particular bidimensional en el que se inicializan los datos a partir de dos distribuciones normales ($nd=2$) y se tienen 4 clases ($K=4$), se tiene la distribución de los puntos inicialmente como se ve en la Fig. 4a, donde inicialmente se tomaron 4 medias aleatorias y se asignó la clase de cada punto en función de la clase de la media más cercana. Luego de cinco iteraciones los puntos quedaron reclasificados como se muestra en la Fig. 4b hasta llegar finalmente al estacionario de la Fig. 4c.

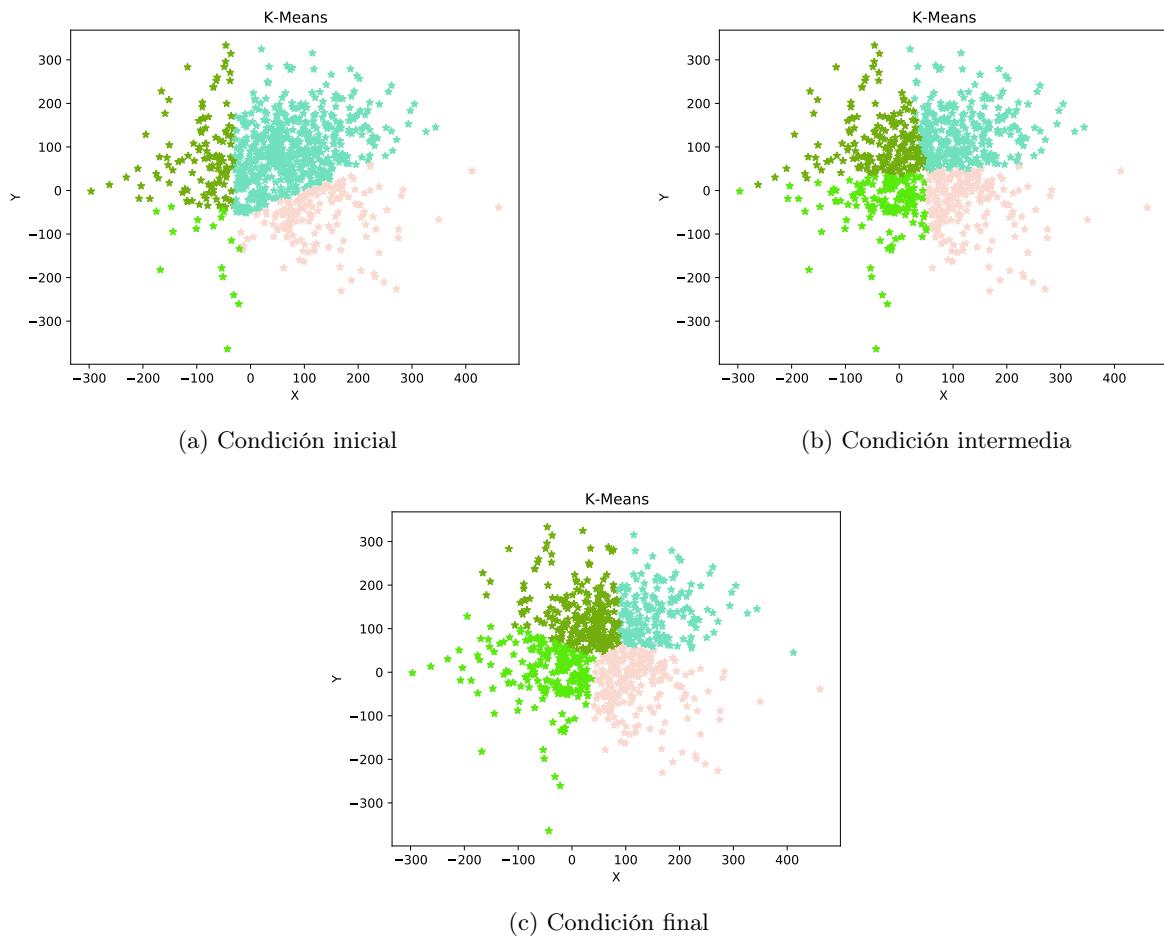


Figura 4: Evolución del clasificador K-Means

El método tiene problemas cuando los datos se encuentran muy cercanos entre sí. Si se fija la media en 0 y una desviación estándar de 0, el método no puede clasificar los puntos por encontrarse demasiado cerca. En la Fig. 5 se ve que el clasificador no le asigna una clase a los datos que se encuentran todos superpuestos con media 0 y desviación estándar 0.

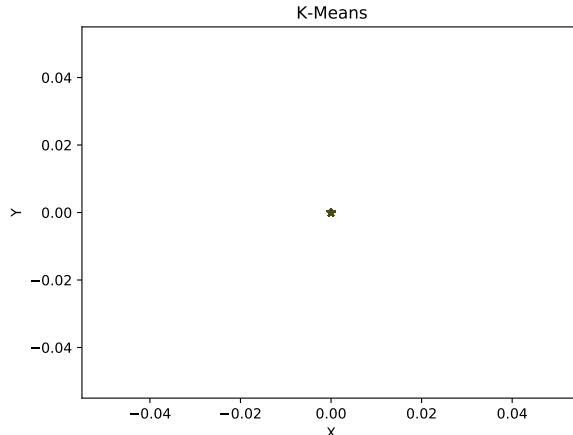


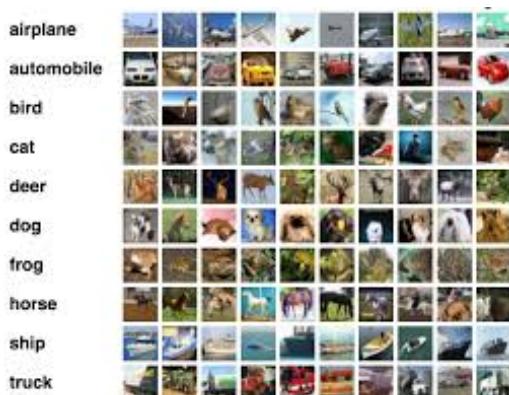
Figura 5: Caso en que el clasificador K-Means falla.

3. Ejercicio 3

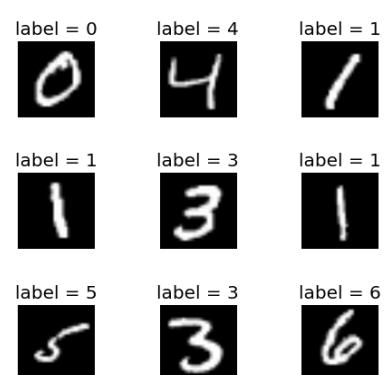
El ejercicio 3 consiste en la implementación del clasificador k-nearest neighbors (KNN) para clasificar las imágenes de las bases de datos CIFAR-10 y MNIST, que pueden ser importados usando la librería Keras de Tensorflow. El método KNN consiste en asignar la clase que tiene un dato en función del parecido que tiene dicho dato con las clases de los datos de entrenamiento. El método se basa en calcular las distancias (por ejemplo, euclídea) del dato que se quiere clasificar con todos los datos del conjunto de datos de entrenamiento. La idea es poder determinar a qué clase pertenece un dato en función de la clase a la que pertenecen los K datos (vecinos) cuya distancia es menor con el dato en cuestión.

MNIST es una base de datos que contiene diez clases (los números del 0 al 9) y cada imagen tiene un tamaño de 28x28x1 píxeles. Por su parte, la base de datos CIFAR-10 contiene imágenes a color (RGB) de tamaño 32x32x3 píxeles y también clasifica las imágenes en 10 clases. Ambos sets de datos contienen un conjunto de datos para entrenamiento y otro para la verificación.

Mediante el método *KNN*, se espera ser capaz de clasificar las imágenes de ambas librerías consiguiendo un *accuracy* del 100 % para los 20 primeros datos de MNIST. Las imágenes que contiene CIFAR-10 son del estilo al que se muestran en la Fig. 6a, y para el MNIST se muestran en la Fig. 6b.



(a) Conjunto de datos CIFAR-10



(b) Conjunto de datos MNIST

Figura 6: Conjunto de datos CIFAR-10 y MNIST

Para este ejercicio, lo primero que se hizo fue importar la librería CIFAR-10 utilizando Keras, y separar los

conjuntos de datos para entrenamiento (x_{train}, y_{train}) del conjunto de datos para la verificación (x_{test}, y_{test}). Luego, se inicializa la clase que da inicio al clasificador KNN, el cual toma como parámetro de entrada la cantidad de vecinos K con el cual va a aplicar el método. Una vez inicializada la clase, se procede a definir el conjunto de datos de entrenamiento que serán las referencias para determinar a qué clase pertenece un dato de prueba. En el entrenamiento, se acomodan las dimensiones de los datos para trabajar de manera conveniente, por ejemplo cada imagen de entrenamiento se coloca en una fila de una matriz de entrenamiento de dimensiones (nimagenes x (32x32x3)). Con una función **predict**, que toma como entrada un dato de prueba, se calculan las distancias euclídeanas entre el dato de prueba y todos los datos de entrenamiento. De este conjunto de distancias se toman los índices de las K distancias más chicas y se clasifica la imagen de prueba en función de la clase que tiene la mayoría de los K vecinos más cercanos. Una función **accuracy** permite evaluar que tan certera fue la predicción de la clase del dato de prueba para un valor de K fijo.

El resultado de aplicar el método KNN con K entre 1, 3, 5 y 7 a los 20 primeros datos de los sets de datos de CIFAR-10:

```
CIFAR-10: K=1, Accuracy=0.3  
CIFAR-10: K=3, Accuracy=0.35  
CIFAR-10: K=5, Accuracy=0.35  
CIFAR-10: K=7, Accuracy=0.3
```

y para MNIST:

```
MNIST: K=1, Accuracy=1.0  
MNIST: K=3, Accuracy=1.0  
MNIST: K=5, Accuracy=1.0  
MNIST: K=7, Accuracy=1.0
```

En ambos casos, los datos de entrenamiento se corresponden con los sets (x_{train}, y_{train}) que es un set de 50000 datos para CIFAR-10 y 60000 datos para MNIST.

Como se ve en el caso de CIFAR-10, la precisión del método aumenta a medida que se aumenta el K, pero a partir de K=7 empieza a disminuir la precisión. Esto se debe a que cada vez se consideran vecinos más lejanos para la clasificación, y esto puede hacer que el método falle porque hay una gran presencia de vecinos (más alejados) de otra clase a la que debería ser. Para resolver este problema, podría asignarse un peso a cada vecino cercano en función de la distancia a la que se encuentra del dato a evaluar, y que la decisión de en qué clase clasificar el dato de prueba se base en la cantidad de vecinos cercanos pesados por la distancia a la que se encuentran. De esta manera, cuando K aumentan, los vecinos más lejanos tendrán menor "poder de clasificación". Por motivos de tiempo no se llegó a implementar este último arreglo al método. En cuanto a los primeros 20 datos de MNIST, el método tiene un 100 % de precisión para los valores de K que se proponen. Sin embargo, si en vez de tomar los primero 20 tomamos los primeros 100 datos, los resultados obtenidos son los siguientes:

```
MNIST: K=1, Accuracy=1.0  
MNIST: K=3, Accuracy=0.99  
MNIST: K=5, Accuracy=0.99  
MNIST: K=7, Accuracy=0.99  
MNIST: K=11, Accuracy=0.98  
MNIST: K=13, Accuracy=0.98  
MNIST: K=15, Accuracy=0.98
```

de donde se ve nuevamente que para elevados valores de K se pierde precisión en el método.

4. Ejercicio 4

Para este ejercicio debe generarse un conjunto de datos bidimensionales (x, y) formados por 5 clases y luego debe aplicarse el método KNN para clasificar un conjunto de datos (x_{test}, y_{test}) tomando $K=1,3,5,7$. Finalmente, deben graficarse las fronteras de decisión.

En la implementación del código, lo primero que se hace es generar los datos (x, y) a partir de una distribución normal cuya media y desviación estándar es aleatoria. Se dice que un par (x, y) pertenece a la clase j cuando la norma cuadrática de (x, y) está contenida en un círculo de radio r. Particularmente, si se definen 5 clases y se trabaja en un entorno de tamaño 100, la primera clase queda definida por el conjunto de datos cuya norma cuadrática es menor a 20, la segunda clase contiene a los datos cuya norma está entre 20 y 40, la tercera contiene a los datos cuya norma está entre 40 y 60, la cuarta contiene a los datos cuya norma está entre 60 y 80 y la quinta contiene a los datos cuya norma es mayor a 80. Se usan 300 datos para el entrenamiento y 50 datos para el testeo. Para graficar las fronteras de decisión, se hace un muestreo del espacio de trabajo utilizando la función *meshgrid* de Numpy, con un paso de 2 en el espacio de trabajo de -200x200 para cada eje. Se tomó un espacio de trabajo más amplio para visualizar los puntos que quedan dispersos al momento de la inicialización de los datos.

Los resultados obtenidos se muestran a continuación:

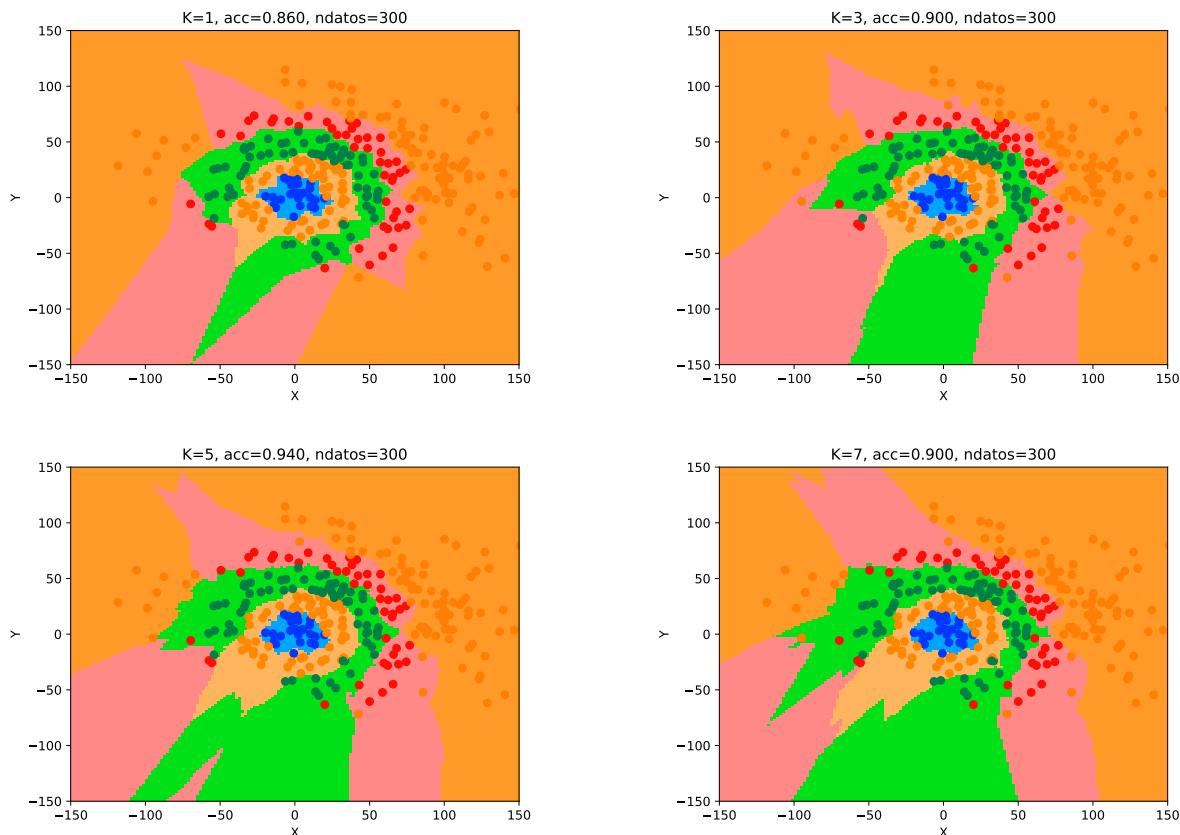


Figura 7: Fronteras de decisión para $K = 1,3,5,7$

Se ve que si se aumenta la cantidad de datos a 1000, las fronteras de decisión se parecen más al criterio de clasificación en función de las normas descripto anteriormente:

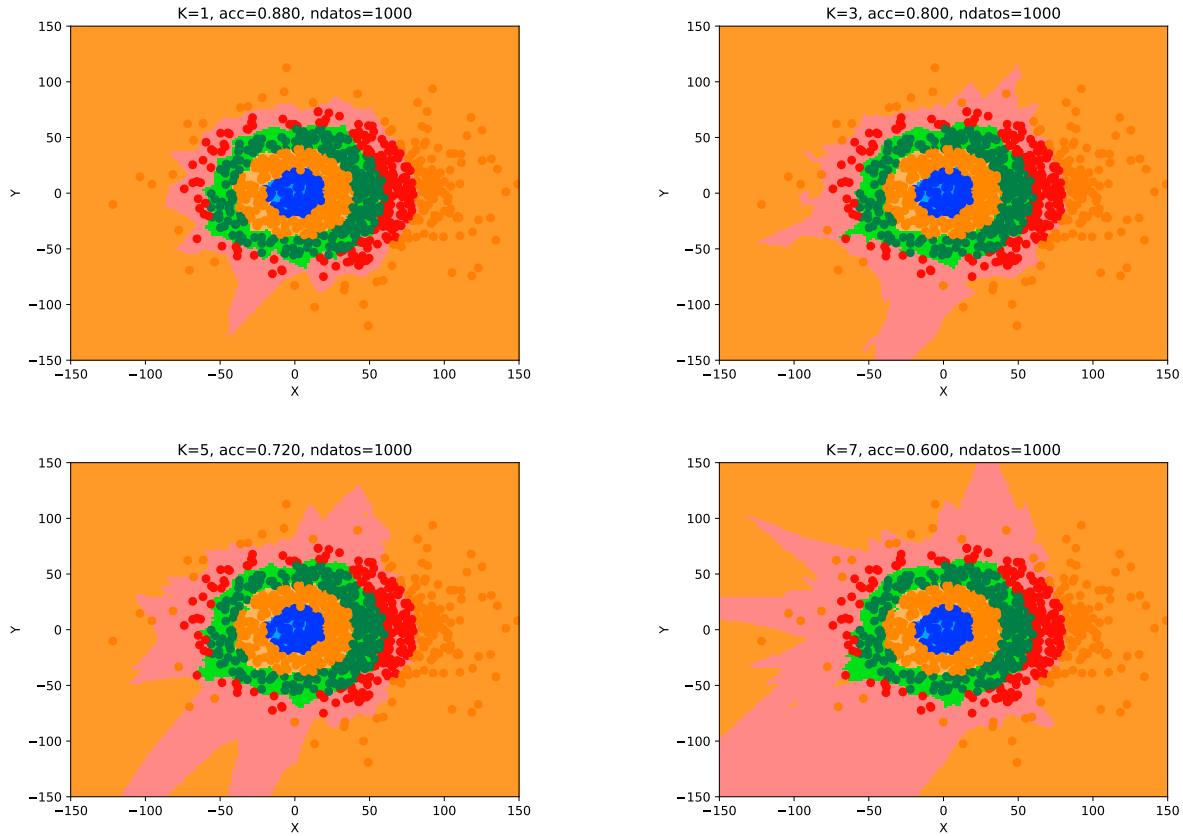


Figura 8: Fronteras de decisión para $K = 1, 3, 5, 7$

5. Ejercicio 5

En el último ejercicio, deben implementarse los clasificadores lineales **SVM** y **SoftMax** usando la regularización L2 para clasificar los datos de CIFAR-10 y MNIST. La idea detrás de estos clasificadores es determinar la clase y a la que un conjunto de datos x pertenece, resolviendo un sistema de ecuaciones lineales:

$$y = W \cdot x, \quad (5.1)$$

donde \mathbf{y} es una matriz que contiene la información de la clase a la que pertenece cada dato del conjunto de datos \mathbf{x} y \mathbf{W} es la matriz de pesos que clasifica cada dato y cuyos pesos deben ser ajustados penalizando las predicciones incorrectas y compensando las correctas.

La manera en la que se calibran los pesos depende de qué clasificador usar y se utilizó un método basado en tomar pequeños lotes de datos aleatorios para entrenar el clasificador. El proceso de calibración de los pesos se realiza en n épocas. Las **épocas** se refieren a la cantidad de veces que se itera tomando un batch de datos aleatorios para ajustar los valores de la matriz \mathbf{W} .

Para ajustar los pesos de \mathbf{W} se utilizó el método de gradiente descendente y se aplica actualiza el valor de cada peso según:

$$w_{actualizada} = W_{anterior} - learning_rate \cdot dW, \quad (5.2)$$

donde el **learning_rate** es el paso de aprendizaje del método y el **dW** es la tasa de cambio de los pesos de la matriz \mathbf{W} que se calcula mediante el método del gradiente descendente. Cabe notar que un **learning_rate** grande puede generar problemas de convergencia del método, mientras que un valor demasiado bajo hace la convergencia muy lenta. El **learning_rate** es un hiperparámetro del problema junto con el valor de regularización

λ y el número de épocas de entrenamiento.

Como se dijo anteriormente, el dW se obtiene calculando el gradiente de la función Loss en cada una de las variables W_{ij} del problema. Para el caso del método Support Vector Machine (SVM) la función Loss se calcula haciendo:

$$\begin{aligned} loss &= \frac{1}{N} \sum_i L(f(x_i, W), y_i) + \lambda R(W), \\ L &= \sum_{j \neq y_i} \max(0, S_j - S_{y_i} + \Delta), \\ R &= \sum_{i,j} W_{i,j}, \end{aligned} \tag{5.3}$$

donde Δ es otro hiperparámetro del problema que es una medida que mientras más alto sea penaliza más la predicción incorrecta del método y R es la regularización L2 del problema.

Para el método de SoftMax, la función Loss se calcula haciendo:

$$\begin{aligned} loss &= \frac{1}{N} \sum_i L(f(x_i, W), y_i) + \lambda R(W), \\ L &= -\log\left(\frac{\exp f_{y_i}}{\sum_j \exp f_j}\right) \\ R &= \sum_{i,j} W_{i,j}. \end{aligned} \tag{5.4}$$

Para la implementación de ambos clasificadores se define la clase *LinearClassifier* con los métodos *fit*, *predict*, *accuracy* y *loss_gradient*. El método *fit* se encarga de arreglar los datos de entrada en un arreglo conveniente para trabajar e inicializa la matriz W a partir de datos generados aleatoriamente. Cabe mencionar que si nuestro conjunto de datos de entrenamiento es una sola imagen de dimensiones (32x32x3) como lo sería para el caso de CIFAR-10, entonces el vector resultante sería un vector de (1x3072). Para poder trabajar con una ecuación lineal de la forma $y = W \cdot x$ se debe agregar una dimensión que condense el vector de términos independientes, lo que resultaría en que x ahora es un vector de (1x3073). De igual manera se agrega una dimensión a W para que sea posible el producto matricial.

A continuación, se inicia la primera época de entrenamiento y para ello se sortean aleatoriamente los lotes de datos que serán tomados para el entrenamiento del método. Luego, se toman sucesivamente los distintos batches para ajustar el valor de W hasta utilizar todos los datos de entrenamiento. Para ajustar los pesos de la matriz W se utiliza alguno de los métodos de clasificación lineal mencionados (SVM o SoftMax), donde cada uno contiene su propia manera de calcular el gradiente de W y el valor de la desviación Loss. Finalmente se actualizan los valores de W mediante la fórmula 5.2, se acumula el valor de la Loss y se acumula el valor del accuracy. Cuando ya se utilizaron todos los batches de datos, se calcula una Loss promedio dividiendo el valor acumulado por el total de batches y de igual manera se calcula el accuracy del método para el conjunto de datos de entrenamiento y de testeo tomando el valor de accuracy acumulado y dividiéndolo en la cantidad de batches utilizados.

Los detalles de implementación de las funciones gradientes se omiten en este trabajo para que no resulte en un desarrollo extenso del formalismo detrás del método, pero puede ser consultado en [2], [3] y el código puede ser consultado en [4].

Los resultados para 1000 épocas de entrenamiento, tomando batches de 50 datos, con un coeficiente de regulación de 0.001, un learning_rate de 0.000001 y tomando 5000 datos para el entrenamiento y 500 para el testeo se muestran en la Fig. 9 para CIFAR-10:

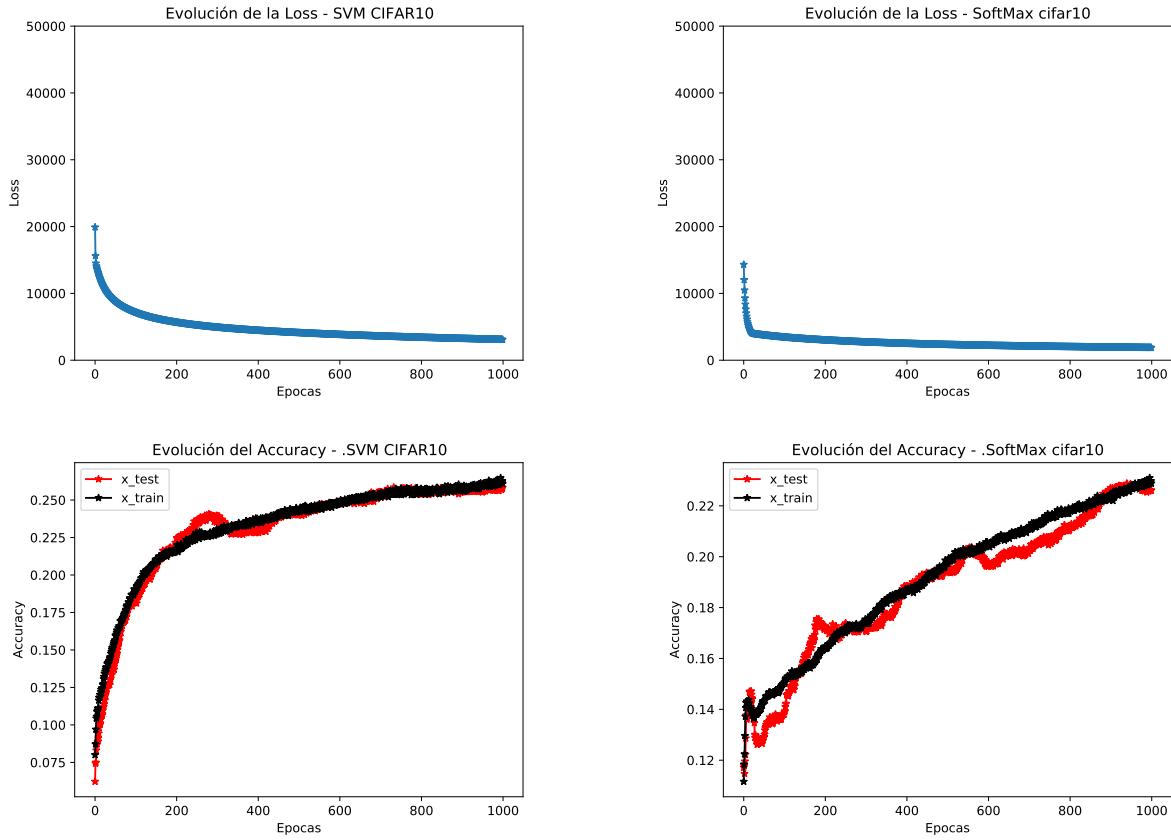


Figura 9: Resultados de la implementación de los clasificadores lineales SVM y SoftMax para clasificar el set de datos de CIFAR-10.

Para el set de datos de MNIST utilizando los mismo hiperparámetros, los resultados se muestran en la Fig. 10.

De las Fig. 9 y 10, se ve que para ambos conjuntos de datos el método SVM tuvo una velocidad de convergencia mayor que el SoftMax. Para el caso de CIFAR-10, el accuracy del método llega al 27 % utilizando el método SVM y 23 % para el SoftMax. Sin embargo, no se vio que el método converja para la cantidad de épocas propuestas, por lo que se espera que la presición máxima alcanzable sea ligeramente mayor. Para el caso del valor de Loss, se ve que el método SoftMax es más rápido para reducir el error de estimación por tener una pendiente más pronunciada en las primeras épocas del método. Para los datos de MNIST, nuevamente se ve que el método SVM tiene una convergencia más rápida que el SoftMax y el accuracy para el SVM supera el 75 % dentro de las 1000 épocas, mientras que para el SoftMax luego de 1000 épocas se llega a 68 % de accuracy.

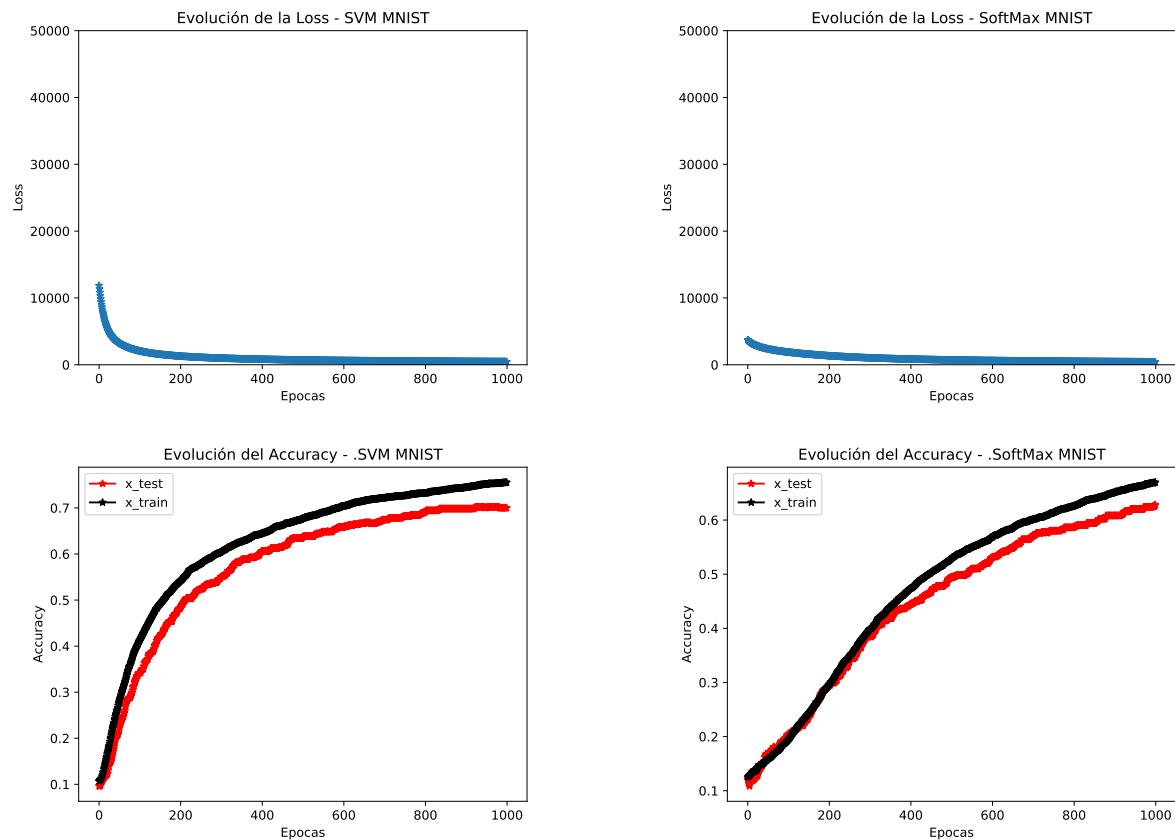


Figura 10: Resultados de la implementación de los clasificadores lineales SVM y SoftMax para clasificar el set de datos de MNIST.

Referencias

- [1] Julien Jacques, Linear Regression in High Dimension and/or for Correlated Inputs. Article in EAS Publications Series · January 2015. <https://www.researchgate.net/publication/276249836>
- [2] Ariel Curiale, German Matos, *Notas de la cátedra: Aprendizaje Profundo y Redes Neuronales Artificiales.* <https://classroom.google.com/u/2/c/MTQxNDcxMTM5NjEx>
- [3] Fei-Fei Li, *Convolutional Neural Networks for Visual Recognition.* <http://cs231n.stanford.edu/2017/>
- [4] Códigos del TP1: https://github.com/TomasLiendro/Deep_Learning/blob/master/TP1_Introduction%20to%20ML/