Modelo Lineal y Algoritmos Genéticos

Josué Fernández Díaz
Instituto Tecnológico de Costa Rica
Ingeniería en Computación
Email: josue.0795@gmail.com

21 de Marzo del 2018

Abstract

En los algoritmos genéticos, la eficiencia de estos es crucial a la hora de realizar los distintos experimentos. En este proyecto se pueden ver varias investigaciones de como un valor aleatorio (y posteriormente mutado) afecta una población de datos, y a partir de estos resultados se obtiene una función de perdida, con resultados como gráficas.

1 Introducción

Dada la gran cantidad de problemas que hay en el mundo, es imposible realizar algoritmos para todos y cada uno de estos problemas, y mucho menos de forma rápida, eficiente y libre de fallos. Por eso nos vimos en la necesidad de crear algoritmos que puedan enseñarse a si mismos, y también formas de probar la eficiencia de dichos algoritmos. En este trabajo se puede ver el uso de algoritmos genéticos para la modificación de una arreglo de números flotantes, de forma que estos puedan predecir la exactitud, usando una función de perdida Hinge Loss.

2 Software Utilizado

Programa	Versión	Función
Python	3.6.4	Software de programación
Numpy	1.14.1	Manejo de arrays multidimensionales
Scipy	1.0.0	Manejo de imágenes
Sklearn	0.19.1	Librería para acceso a Iris
Matplotlib	2.2.0	Librería para el manejo de gráficos

2.1 Instalación

Las diferentes librerías fueron instaladas según el tutorial de Scipy [5]. Pero al haber un error, se procedió a cambiar Numpy por Numpy+MKL [3]. Se

usó la librería de Sklearn[2] para importar los datos de Iris. Instalándolo en otra computadora, solo con el primero tutorial fue suficiente, no dio paso a dicho error. Esta computadora estaba recién formateada, por lo que es poco probable que se presente el error mencionado instalándolo en otras máquinas. Se obtuvo un tutorial [4] de como usar los gráficos de forma eficiente con la librería Matplotlib.

3 Trabajo Relacionado

No se encontraron trabajos que sean completamente similares al presentado en este proyecto, pero si fragmentos, de los cuales logré obtener conocimientos para poder hacer avanzar con la investigación. Como por ejemplo una forma de mutar datos[6] donde se mencionan formas de manipular los datos, de forma que puedan dar resultados favorables. También una investigación bastante similar al proyecto[1] donde se puede ver varios pasos vistos en este proyecto, y que llevan a unos resultados importantes en la investigación.

4 Metodología

4.1 Selección del W

Para obtener el W de la función, se tiene un numero flotante aleatorio entre 0 y 8. Esto para cada una de las posiciones del array. Esto porque son similares a los números originales del set de datos Iris, a pesar que no se trabajan con datos reales, deben ser realistas. Para esto se usó una funcionalidad propia del numpy, que obtiene el resultado deseado:

```
np.random.uniform(0, 8, size=(row,column))
```

4.2 Selección del X

Se tiene la opción de seleccionar una cantidad establecida de datos del dataset (variable steps en la función Main), siempre y cuando esta cantidad sea múltiplo del largo original del dataset. Por ejemplo, para CIFAR se pueden seleccionar 100 imágenes para ser evaluadas a la vez, esto con el fin de acelerar el procedimiento sin dañar el resultado final. Así, de 10000 datos, se harán 10000/100=100 recorridos. Para Iris, se recomienda usar un dato por iteración, de forma que sean 150 recorridos, la cantidad total de datos.

4.3 Hinge Loss

La función de perdida utilizada es la de Hinge Loss, que permite obtener los cambios en la matriz aleatoria W al operarla con los datos obtenidos, así como su

funcionalidad obteniendo la perdida de esta con respecto al resultado correcto.

$$L_i = \sum_{t \neq y} \max(0, 1\mathbf{s}_j - \mathbf{s}_{yi} + 1), s_j = f(x_i, W) \quad (1)$$

Para esto, se deberá tomar los datos incorrectos y restarlos al dato correcto, sumando 1. Si el valor es negativo, quiere decir que el valor correcto es superior al incorrecto, por lo tanto el modelo hizo una buena predicción, pero al ser un máximo entre 0 y dicho valor, se selecciona 0. Por otro lado, si es positivo, ese será su valor de perdida o fallo, ya que el modelo no pudo predecir con exactitud la decisión correcta.

5 Primeros Métodos de Mutaciones

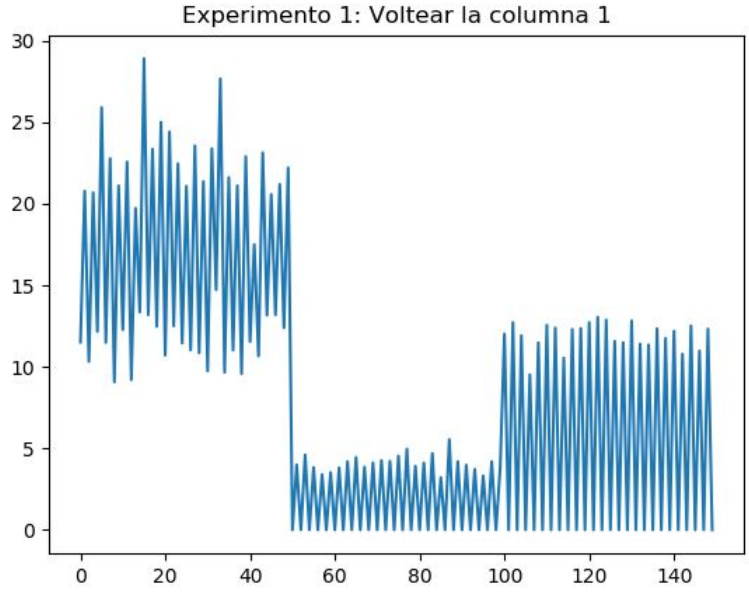
Cabe mencionar que todas las pruebas fueron realizadas con los 150 datos de Iris, tomando los datos de uno en uno, aplicando el W, obteniendo el Hinge Loss, mutando W y repitiendo el proceso, esto 150 veces, una vez por dato. Entonces, el resultado presentado es la recuperación de todos los hinge loss para todos los datos del dataset de Iris.

5.1 Criterio usado para las mutaciones

No se tomó ninguna decisión particular a la hora de realizar los algoritmos de mutación de W. Estos simplemente se pensaron en el momento y fueron programados, sin esperar ningún resultado especifico de los experimentos. Lo único que se tomo en cuenta fue probar cada una de las mutaciones para verificar que su funcionamiento es similar en todas las corridas, y cambiar la lógica de cada experimento para poder dar un ámbito mas grande para la investigación.

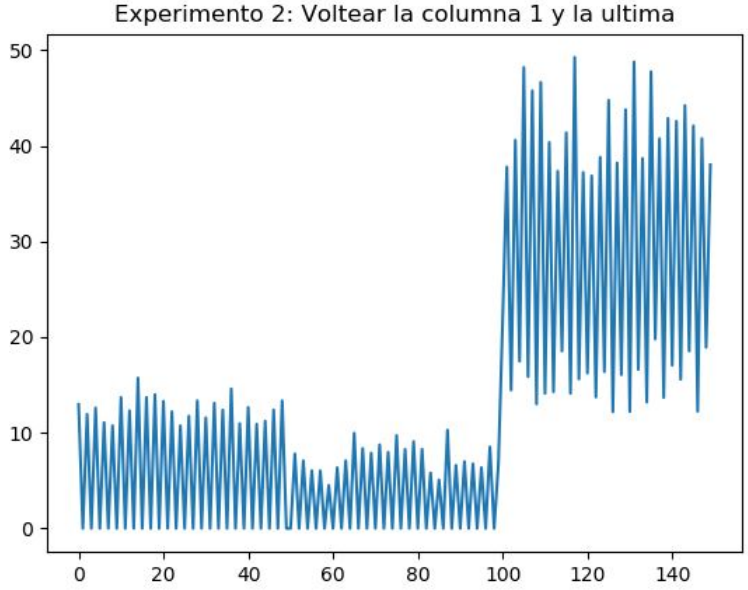
5.1.1 Experimento 1

Se empezó por una mutación simple, la cual fue darle vuelta completamente a una columna de W, mediante el método flip de numpy. Es decir que el primer elemento será el ultimo, y el ultimo el primero, esto solo para la primera columna. El método no muestra funcionalidad ya que sube y baja en su valor de perdida, no logrando disminuir como se quiere. Es decir se mantiene entre un rango donde baja hasta cierto punto, y vuelve a subir al punto inicial, no logrando nada.



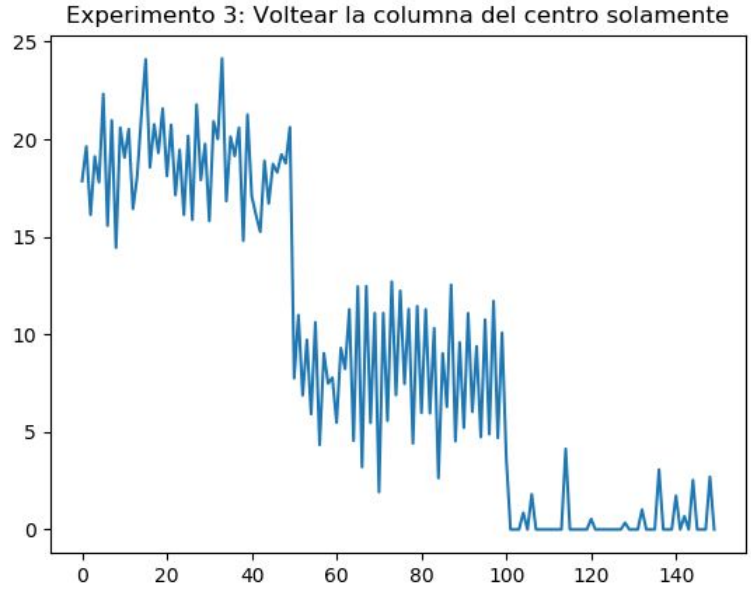
5.1.2 Experimento 2

Dado el mismo experimento de antes, se rotó la ultima columna también. Pero no muestra resultados favorables.



5.1.3 Experimento 3

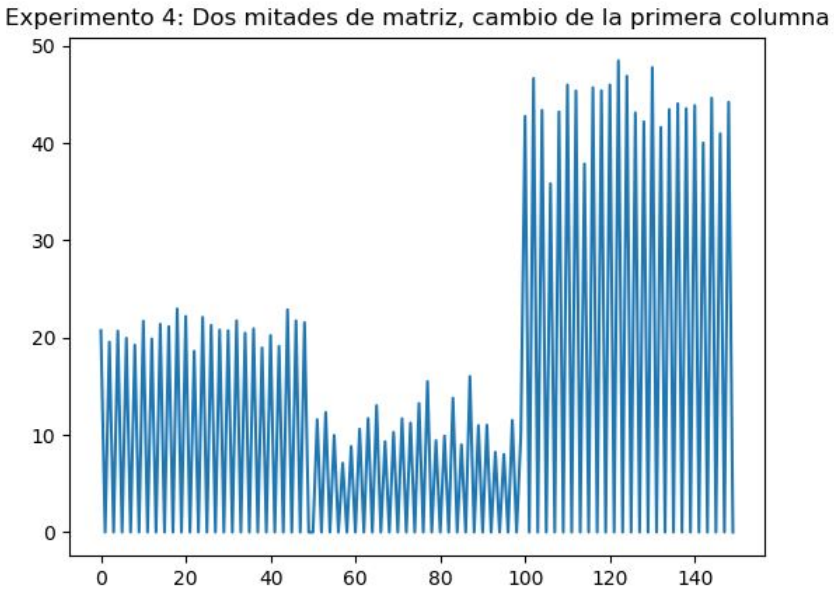
De igual forma, se rotó únicamente la columna del centro, mostrando la siguiente gráfica:



Aunque lastimosamente esto no muestra una mejora, ya que en otras pruebas no resultó de forma tan espléndida, y solo es ocasional.

5.1.4 Experimento 4

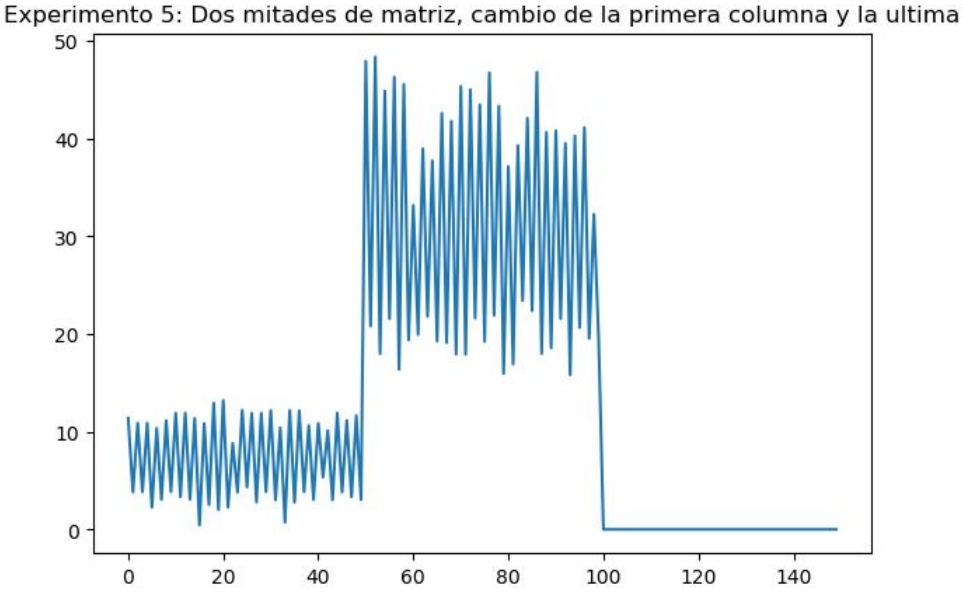
En este experimento se dividió la matriz en dos mitades, donde se intercambian valores entre el primer elemento y la mitad de la matriz original (el final de la matriz-mitad), avanzando desde el inicio y disminuyendo desde la mitad, de forma que ambos índices se encuentren en algún punto. Se hace de igual forma con la segunda mitad, es decir desde el medio hasta el final de la lista original. En este primer acercamiento se intentó cambiar solamente la primera columna de la matriz.



Como se puede ver, no se presenta una mejoría en el Hinge Loss, por lo que el algoritmo sigue sin aproximarse al indicado.

5.1.5 Experimento 5

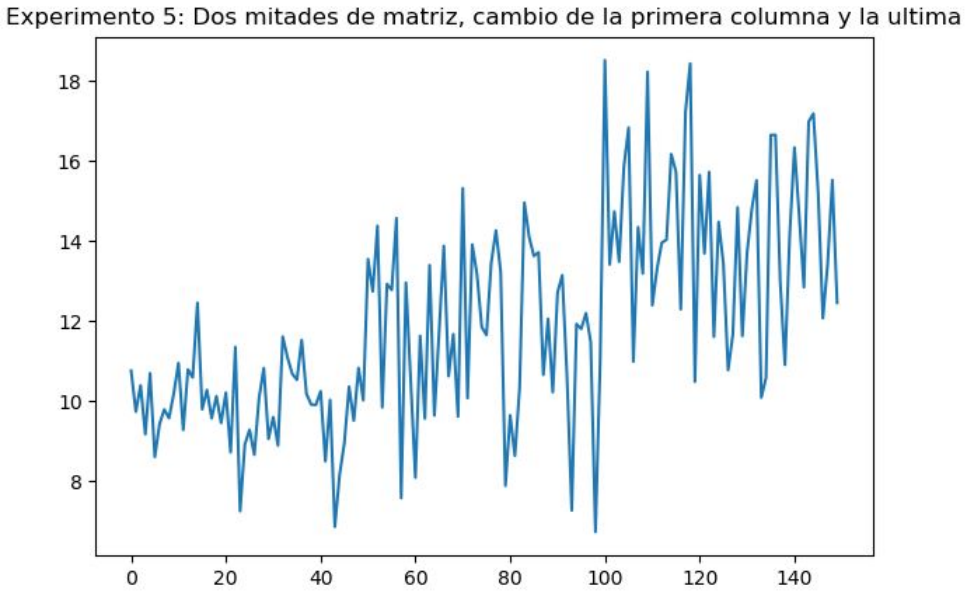
De igual forma que en el anterior, pero se cambiaron la primera columna y la ultima.



A pesar de que se muestra una mejoría, esto no es así porque no todos los casos dan el mismo resultado.

5.1.6 Experimento 6

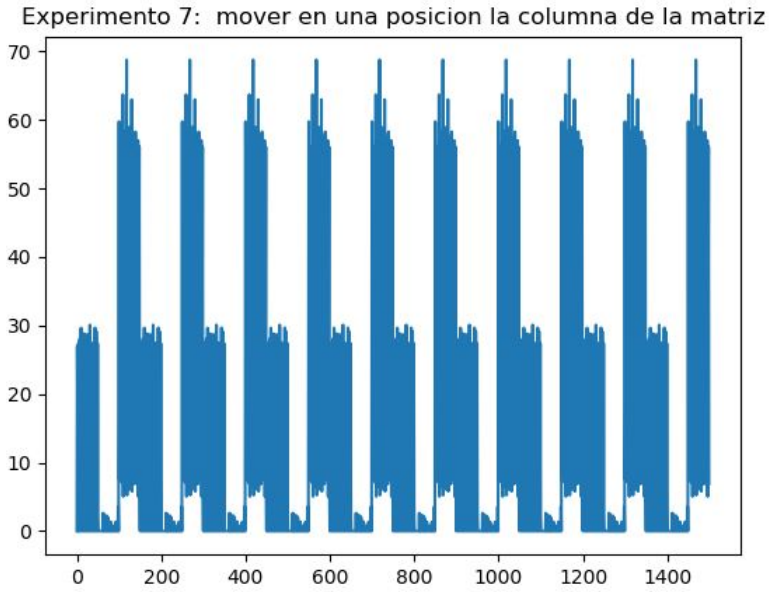
De igual forma que en el anterior, pero se cambiaron la segunda columna y la ultima. Se ve una diferencia al cambiar varias columnas en la aplicación, pero solo en esta se logró ver un cambio.



Este por ejemplo, no muestra ninguna mejoría, pero es el primero que se sale de lo común, comparándolo del resto y muestra trazos mas distantes e irregulares.

5.1.7 Experimento 7

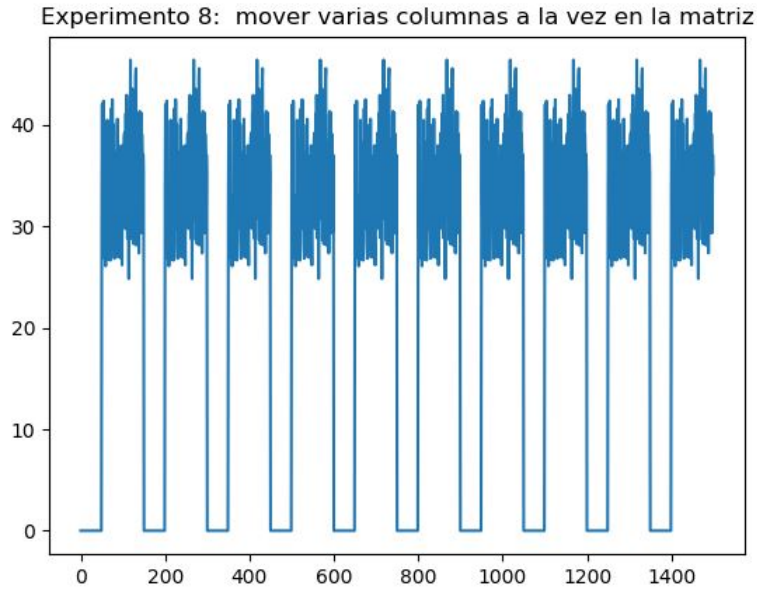
En este experimento, se intentó otro enfoque, el cual es mover una posición hacia abajo una columna. De forma que el primer valor de la primera fila de la matriz, sea el primer valor de la segunda fila, así para todas las filas. Para esto se usó la funcionalidad roll predefinida por numpy. Además, a partir de aquí se empiezan a trabajar con 1500 datos, repitiendo por lo tanto 10 veces los datos provistos por Iris.



Siguiendo esta lógica, se intentó mover las demás columnas, e incluso varias a la vez, pero el resultado no es muy distinto a lo visto en la imagen.

5.1.8 Experimento 8

Siguiendo el ejemplo anterior, se hizo un ciclo donde todas las columnas se mueven cierta cantidad de veces. Es decir, la columna 0 se mueve 0 veces hacia abajo, la columna 1 se mueve 1 vez hacia abajo, y así sucesivamente.



Tomando en cuenta esto, no se vio ningún cambio de mayor significancia en los datos recolectados.

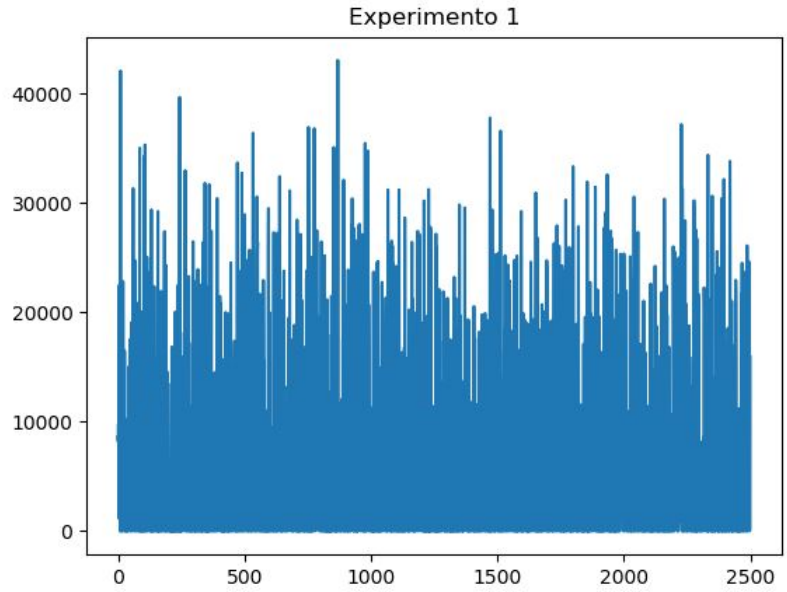
5.2 Función de Parada del Algoritmo

Ahora que se tiene un mejor manejo del programa debido a estos experimentos anteriores, se programó la función de parada automática. Esta funciona de la siguiente forma: Dada una precisión (entero) que es un hiper parámetro, se marca cual debe ser la diferencia de los últimos Hinge Loss obtenidos. Es decir, supongamos que tenemos una precisión de 20, esto quiere decir que el ultimo valor de Hinge Loss obtenido y el penúltimo deben tener una diferencia menor a 20. A su vez, el penúltimo y el anterior a este también deben tener una diferencia menor a 20. Esto para los últimos 10 elementos obtenidos. De ser así, el algoritmo se detiene y despliega el gráfico. De igual forma, los datos de Iris están programados para repetirse las veces que sean necesarias. Se recomienda no dar una precisión muy baja, ya que debido a la aleatoriedad del programa, puede que nunca llegue a un final. En este proyecto, se le puso un tope de 150000 recorridos, en caso de que no encuentre una solución antes, sepa detenerse y no enciclarse.

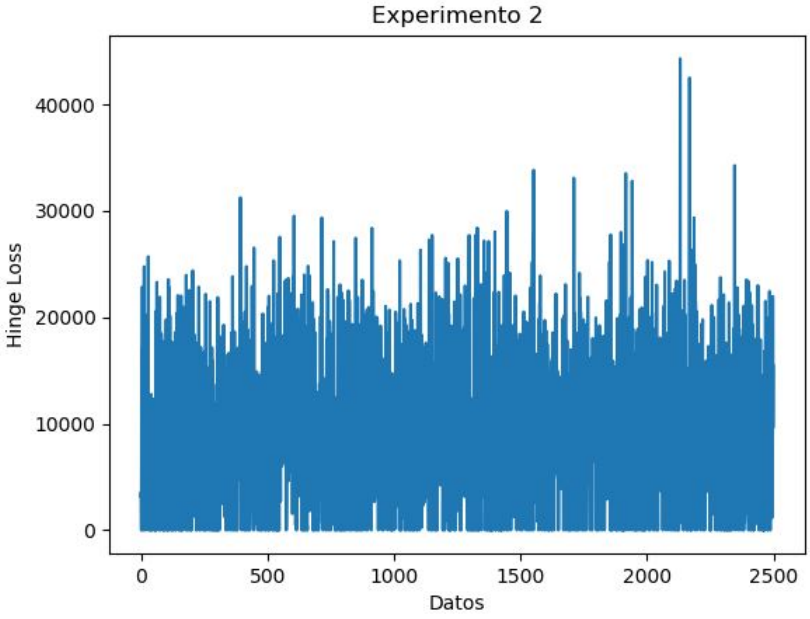
6 Resultados

Tomando en cuenta el set de datos CIFAR, se escogieron las primeras 4 clases, con la diferencia de que se le restaron 1 al índice de la clase. Originalmente van de 1..4, pero para que sea completamente funcional e igual a Iris que va de 0..3, se resto uno para que el set de CIFAR también sea de 0..3. Además, se corrió una prueba por dato, es decir 25000 a menos que la función de parada lo detenga antes. Los experimentos a continuación son los mismos realizados para Iris, de forma que se pueda hacer una comparación de las gráficas.

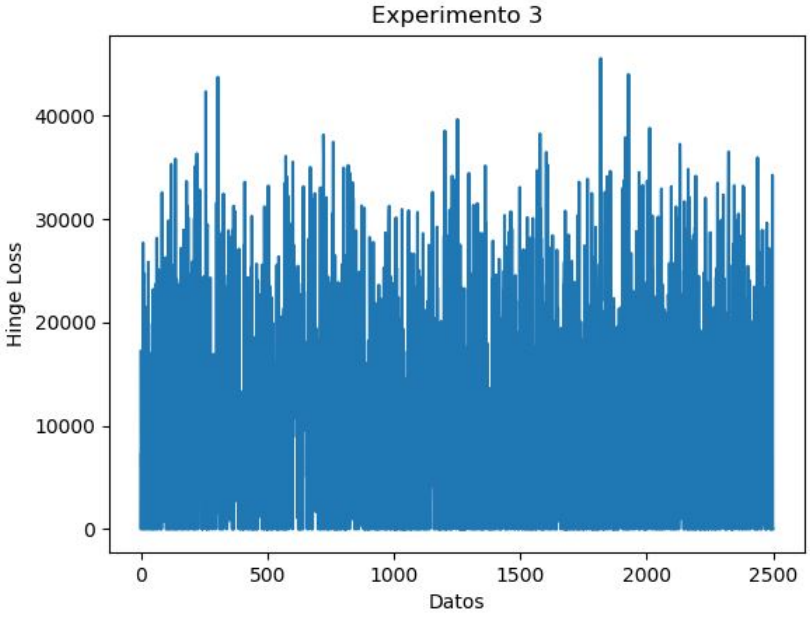
6.0.1 Experimento 1



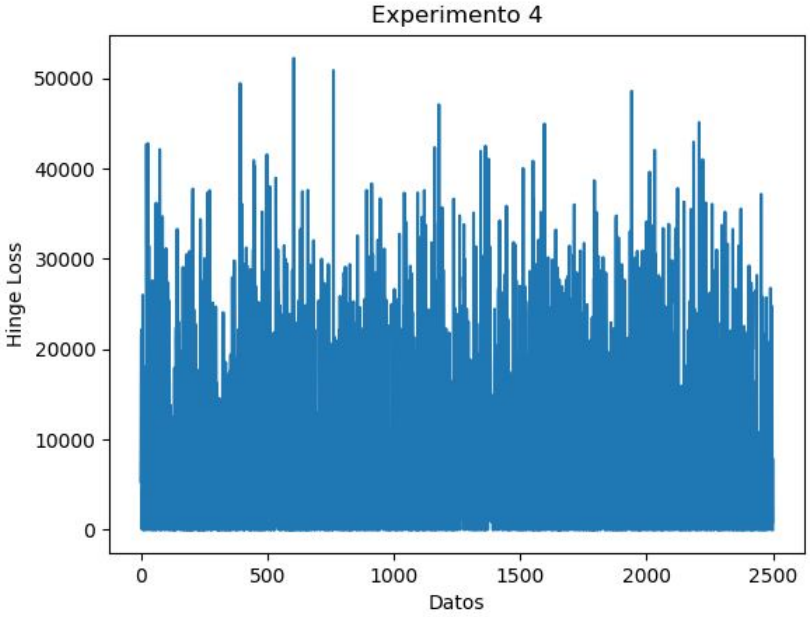
6.0.2 Experimento 2



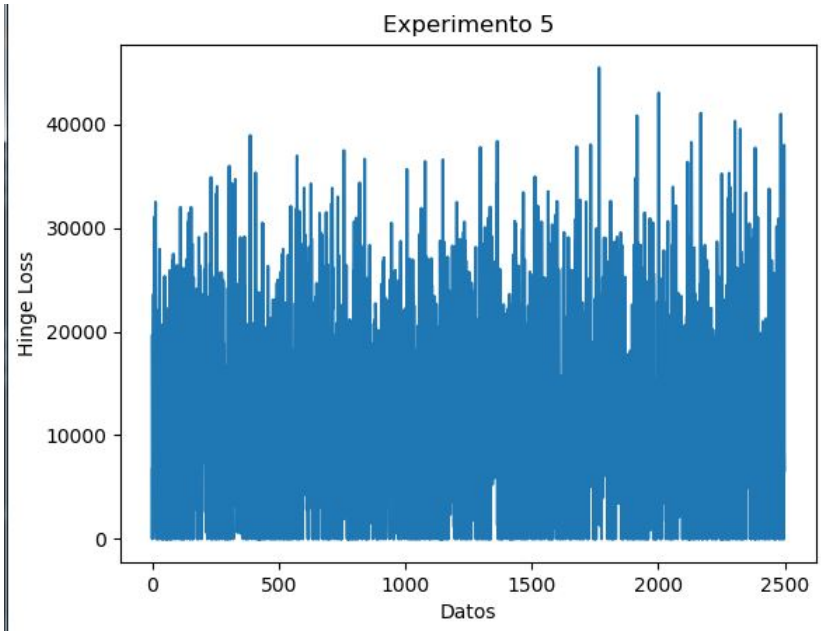
6.0.3 Experimento 3



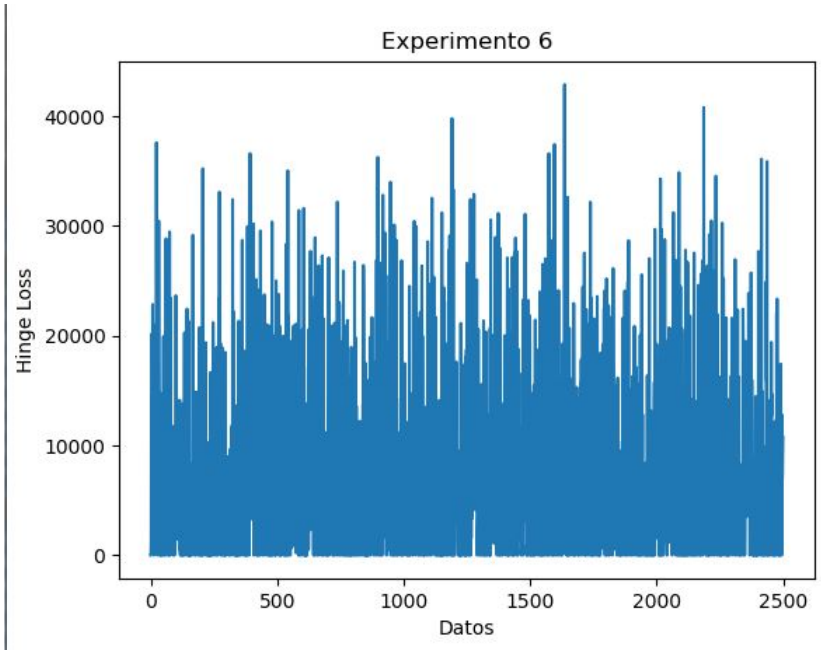
6.0.4 Experimento 4



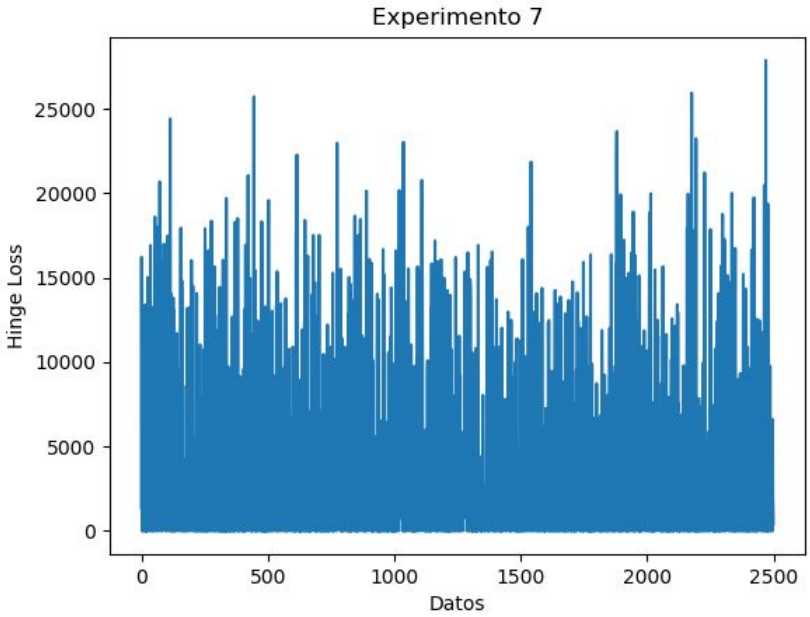
6.0.5 Experimento 5



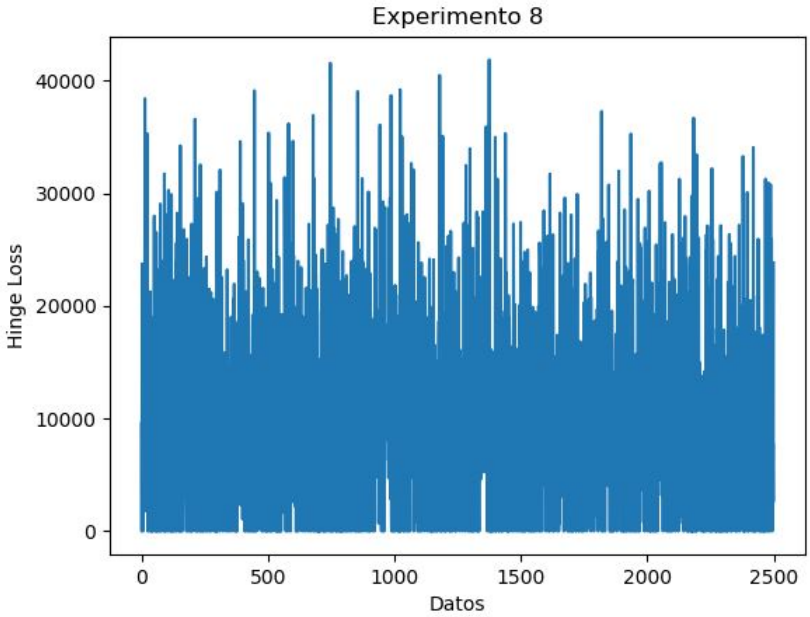
6.0.6 Experimento 6



6.0.7 Experimento 7



6.0.8 Experimento 8



7 Conclusiones

Desafortunadamente, en ninguno de los casos se puede ver un caso favorable, ya sea para el set de datos de Iris como para CIFAR. El algoritmo tiende a caer de alguna forma en antiguos pasos, y envía una serie similar de resultados, creando el resultado visto en los experimentos. Ignoro de donde proviene esto, ya que las pruebas fueron realizadas dese varios enfoques, por lo que debería de hacer diferencias en los datos, pero aun así se vieron similares los datos afectados. Se intentó revisar todo el procedimiento desde cero pero sin éxito, ya que no se encontró ninguna pista de donde puede estar fallando.

References

[1] Sin Autor. CS231n convolutional neural networks for visual recognition.

[2] Jason Brownlee. Installing scikit-learn, 2014.

[3] Jirka. Error importing scikit-learn modules, 2015.

[4] Pherkad. Gráficos en ipython, 2014.

[5] Scipy.org. Installing packages.

[6] Tr4nsduc7or. Cómo programar un algoritmo genético – parte i: In theory, 2015.