INF-393: Maquinas de Aprendizaje
Tarea 1

Pablo Ibarra Sepúlveda
201173086-0

2 de Noviembre de 2015

1. Regresión Lineal

En esta sección utilizaremos distintos algoritmos que están relacionados con regresión lineal, el objetivo de esta sección es comparar estos algoritmos en distintos contextos, estos son cuando los datos se encuentran normalizados, estandarizados o simplemente sin ningún tipo de intervención. Comenzaremos analizando algoritmo los algoritmos individualmente, en los escenarios ya mencionadas, para ver como se comportan y que relación tiene ese comportamiento con su parte teórica.

1.1. Grandiante Descendente

Los algoritmos de Radiante Descendiente en regresión lineal intentan aproximar una función lineal, estimando sus parámetros, estos algoritmos lo que hacen es minimizar una función de perdida, para ir actualizando unos coeficiente, que finalmente serán los parámetros que buscamos. Utilizando estos parámetros construiremos una función que llamaremos Learner, el fin de esta función es que cuando nosotros la evaluemos, esta nos dará un valor muy similar al valor que la función que buscamos aproximar tomaría si la evaluamos en los mismos valores. Ahora bien, es importante mencionar la función que utilizaremos en este informe para utilizar algún algoritmo de gradiente descendiente, y esta función es la función de perdida de los mínimos cuadrados.

$$\mathrm{J}(\beta) = \frac{1}{2}\sum_{i=1}^{M}(f(x^{(i)}) - y^{(i)})^2$$

Esta función básicamente lo que hace es medir la diferencia entre el valor esperado y el valor predicho, para cada vector de nuestra matriz.
Ahora que sabemos la función a utilizar presentaremos el modelo de gradiente descendente, que actualizara los coeficientes, se compone de la derivada de alguna función de costo, y de un α que es llamado Learning Rate, esta alpha cumplirá una función clave, ya que nos dará el grado de rapidez en el que el algoritmo encontrara los parámetros, pero si este coeficiente es muy grande, puede ser que nuestra función diverja, es decir jamás encuentre los parámetros, y si es muy pequeño le tomara mucho tiempo en encontrar la solución.

$$\beta_j^{\rho+1} = \beta_j^{\rho} - \alpha\frac{\varphi}{\varphi\beta_j}J(\beta)$$

Ahora para poder usar este modelo, debemos obtener la derivada de nuestra función de costo, para poder remplazarla en el modelo de gradiente descendente.

$$\begin{gathered}\frac{\varphi}{\varphi\beta_i}J(\beta) = \frac{\varphi}{\varphi\beta_i}\frac{1}{2}(f(x)-y)^2 \\ (f(x)-y)\frac{\varphi}{\varphi\beta_j}\sum_{i=}^{n}(\beta_j x_j - y) \\ (f(x)-y)x_j\end{gathered}$$

Obtenida la derivada la podemos remplazar en nuestro modelo de gradiente descendente, obteniendo finalmente.

$$\beta_j^{\rho+1} = \beta_j^{\rho} - \alpha(f(x^{(i)}) - y^{(i)})x_j^{(i)}$$

Esta expresión se llama la regla de aprendizaje de Widrow-Hoff, y será la base para los algoritmos de gradiente descendente que expondremos a continuación.

1.1.1. Gradiente Descendente Batch

El primer algoritmo que utilizaremos será el Gradiente Descendente Batch, este algoritmo se caracteriza por lo siguiente, el sumara todos los errores cuadráticos de todos los datos de entrenamiento que tengamos respecto a nuestros objetivos, los multiplicara por el Learning Rate y hará la actualización correspondiente a los parámetros, acá el Learning Rate es muy importante, ya que al ser la suma de todos los errores, el numero será muy grande, y al momento de actualizar no controlamos este valor, los parámetros crecerán haciendo que nuestro algoritmo diverja.
Es importante recalcar que la función errores cuadráticos es una función convexa, por lo que tiene solo un mínimo local, y lo que intentara el gradiente descendente es encontrar ese mínimo.

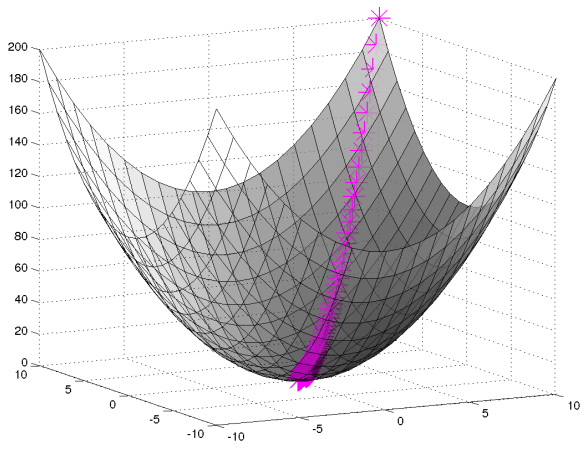

Figura 1: Gradiante Descendente Batch busca el minimo de la función de perdida

El algoritmo de Gradiente Descendente Batch básicamente es repetir la regla de actualización hasta que los parámetros converjan en algún valor.

$$\beta_j^{\rho+1} = \beta_j^{\rho} - \alpha \sum_{i=1}^{M} (f(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

Lo anterior, se puede escribir de manera matricial, esto ayuda en la programación y entender el algoritmo multidimensionalmente, nos quedaría:

$$\beta^{p+1} = \beta^{p} - \alpha X^T (Y - X\beta^{p})$$

Ahora utilizaremos Gradiente Descendente Batch y utilizando el Dataset de Cereales, se corrieron ciertas pruebas sobre 20 Training Set previamente generados, Observaremos como se comporta este algoritmo en los contextos presentados en la introducción, y como el Learning es un factor importante al momento de generar los parámetros.

Se comenzó con analizar los datos sin ninguna intervención, debido a esto utilizamos 5 α candidatos bien pequeños, para que el algoritmo converja, estos son $10^{-8}, 3x10^{-8}, 6x10^{-8}, 9x10^{-8}, 10^{-7}$, utilizando "5- cross validationz sobre los 20 Dataset previamente generados obtuvimos los siguiente errores cuadráticos:

DataSet1	10(-8)	3*10(-8)	6*10(-8)	9*10(-8)	10(-7)
Media	437.5776	418.1109	406.7616	406.7616	391.5556
Desviación	295.7561	266.1131	252.1729	244.5891	241.8173
Coeficiente de Variación	0.6758940	0.6364654	0.6199527	0.6185621	0.6175809

DataSet2	10(-8)	3*10(-8)	6*10(-8)	9*10(-8)	10(-7)
Media	470.4361	443.7976	415.6782	384.2710	378.2393
Desviación	238.8118	207.8994	195.5582	181.6679	178.4483
Coeficiente de Variación	0.5076391	0.4684554	0.4704558	0.4727600	0.4717868

Las dos tablas anteriores muestran, la Media, la Desviación y el coeficiente de variación de los errores cuadráticos medios, analizando estos datos vamos a determinar cual α será el ganador, o el mas indicado bajos los siguientes criterios.

1. El primer criterio que utilizare será la Media, el que tenga la Media menor será ganador, ya que podremos decir que tiene un error menor comparado a los otros Learning Rate.

2. Si sucediera que las Medias son muy parecidas entre Learning Rate, utilizaremos el coeficiente de variación para determinar el ganador, debido a que el que tenga el coeficiente de variación menor, será mas homogéneo, es decir la relación entre la media y la desviación es menor, es decir los distintos errores esta mas agrupados alrededor de la Media.

Establecidos los criterios, podemos decir el ganador en la Tabla numero uno fue 10^{-7} ya que tiene la media menor y el coeficiente de variación mas pequeño, en la Tabla numero dos fue 10^{-7}, ya que tiene la menor media.

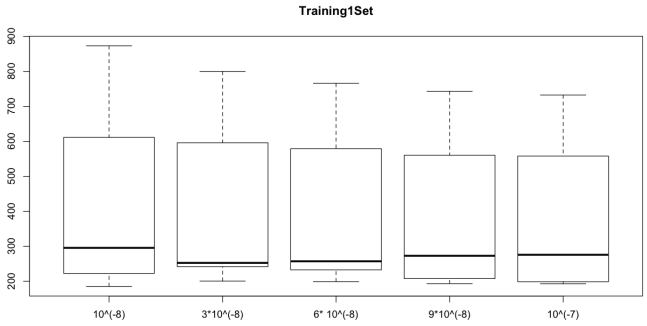

Figura 2: Boxplot de el primer Dataset

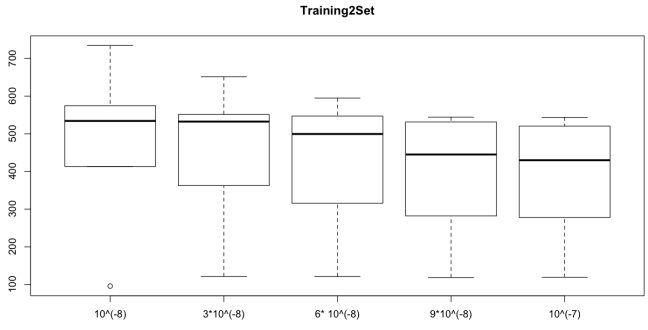

Figura 3: Boxplot del segundo Dataset

Siguiendo el mismo criterio y repitiendo el mismo proceso a los 18 dataset restantes, se obtuvo que el α ganador fue 10^{-7}.

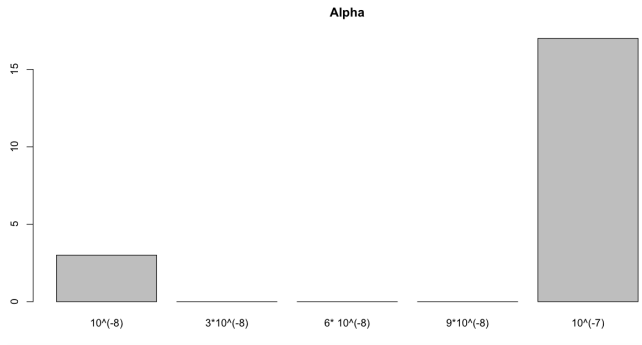

Figura 4: Diagrama de Barra de los dist

Del mismo modo, repetiremos el mismo proceso, pero cambiaremos los α, para normalización y estandarización. Los α a utilizar serán $10^{-6}, 2*10^{-6}, 3*10^{-6}, 4*10^{-6}, 5*10^{-6}$
Como mencionamos el Learning Rate es un factor importante, como anteriormente utilizamos datos crudos era necesario utilizar un alpha pequeño, si no el algoritmo divergía.

A continuación presentaremos el comportamiento de gradiente descendente batch con datos normalizados entre 0,1.

Dataset2	10^(-4)	2*10^(-4)	10^(-3)	2*10^(-3)	10^(-5)
Media	250.1012	241.6023	230.3424	229.8142	693.5352
Desviación	145.5292	132.0385	118.3671	116.9852	256.2282
Coeficiente de Variación	0.5818811	0.5465119	0.5138743	0.5090426	0.3694523

DataSet 20	10^(-4)	2*10^(-4)	10^(-3)	2*10^(-3)	10^(-5)
Media	237.3628	228.9596	217.8084	215.8323	694.7221
Desviación	153.9396	142.6910	132.1080	131.2344	298.9289
Coeficiente de Variación	0.6485414	0.6232147	0.6065333	0.6080385	0.4302856

Utilizando los criterios establecidos anteriormente, podemos decir que el ganador en el DataSet 2 fue $2x10^{-3}$ por el hecho de tener una media mas pequeña, lo mismo pasa en el DataSet2.

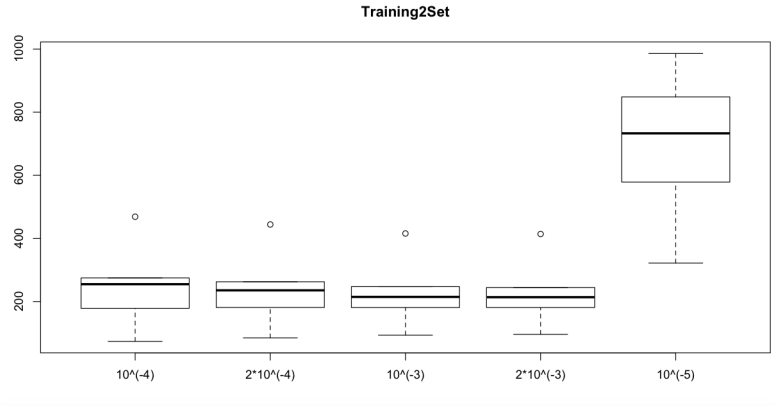

Figura 5: Boxplot de el primer Dataset

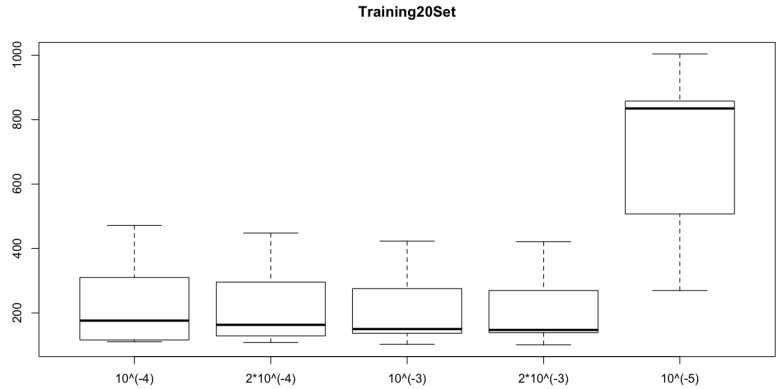

Figura 6: Boxplot de el segundo Dataset

Repitiendo lo mismo para los 18 Dataset restantes obtenemos que el α ganador fue 10^{-6}.

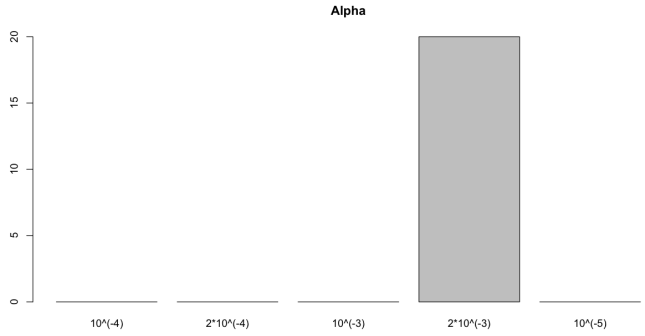

Figura 7: Diagrama de Barra de los distintos alpha

Ahora nos queda hacer lo mismo, esta vez con los datos estandarizados, utilizaremos los mismos α que en la sección de normalización.

TrainingSet3	10^(-4)	2*10^(-4)	10^(-3)	2*10^(-3)	10^(-5)
Media	90.82659	76.27120	62.60179	23.76187	189.26238
Desviación	17.283866	12.619405	14.928161	9.344606	51.314935
Coeficiente de Variación	0.1902952	0.1957247	0.2015822	0.3932606	0.2711312

TrainingSet5	10^(-4)	2*10^(-4)	10^(-3)	2*10^(-3)	10^(-5)
Media	80.73081	72.62930	63.69989	32.77656	160.44400
Desviación	27.10658	25.27642	25.08013	27.38582	53.06872
Coeficiente de Variación	0.3357650	0.3480197	0.3937233	0.8355307	0.3307617

Utilizando los criterios establecidos anteriormente, no podemos afirmar solo mirando la media quien es mejor, ya que son muy parecidas, para eso usaremos el coeficiente de variación para discriminar, en ambos casos el ganador fue $2x10^4$.

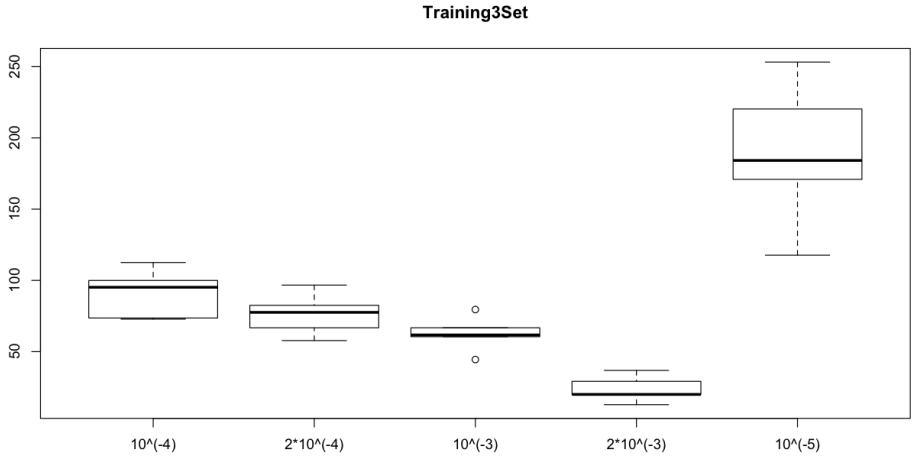

Figura 8: Boxplot de los distintos alpha del trainingset 3

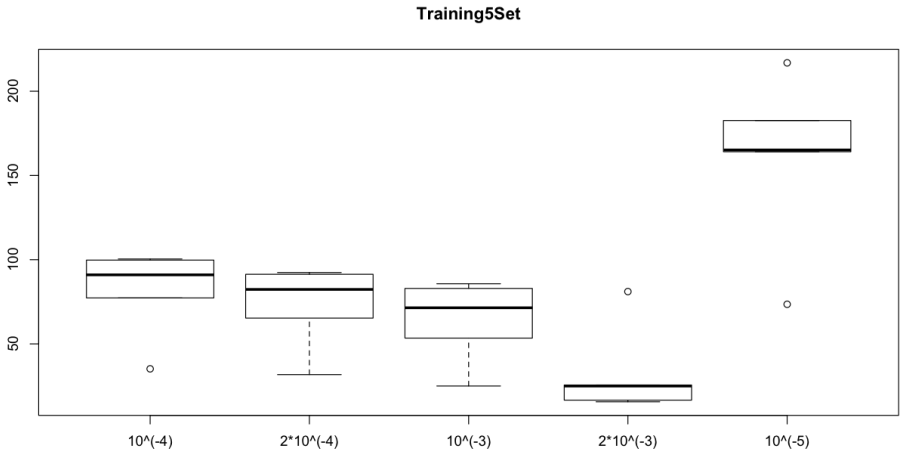

Figura 9: Boxplot de los distintos alpha del trainingset 5

Repetidiendo lo mismo para los 18 Dataset restantantes obtenemos que el α ganador fue $2x10^{-3}$.

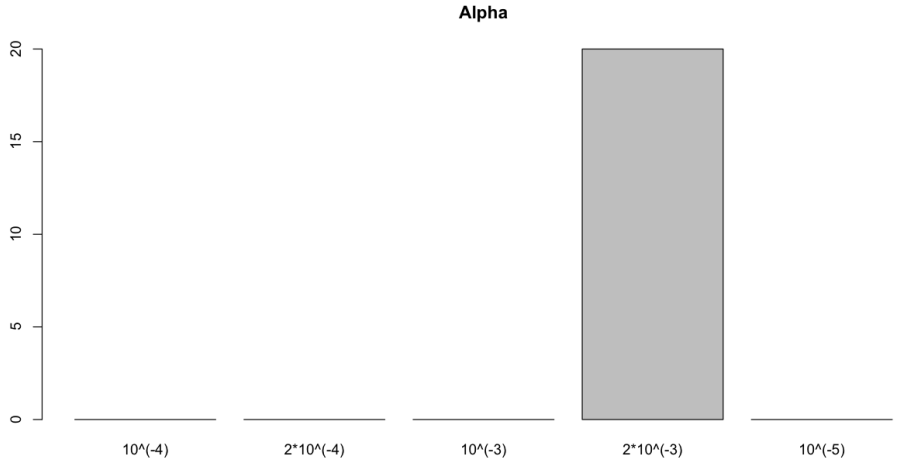

Figura 10: Diagrama de Barra de los distintos alpha

Luego de haber usado nuestro algoritmo gradiente descendente Batch, con datos normalizados, estandarizados y sin ninguna intervención notamos grandes diferencias, como las siguientes:

1. La primera y quizás mas importante es el cambio en el error cuadrático medio, se puede apreciar sobre todo entre los datos normalizados y estandarizados, en el primer caso el error cuadrático medio esta en el orden los múltiplos de cien, en cambio en el segundo caso el error cuadrático medio esta en el orden de las decenas.

2. Otra diferencia destacable, es sobre la variación de los Learning Rates, si nos vamos a los datos estandarizados, podemos ver que la variación en los Learning Rates desencadena un cambio brusco en el error cuadrático medio, pero en cambio si normalizados , la variación del error cuadrático medio es menos brusca que en el otro caso.

3. Otra punto destacable es el coeficiente de variación, nos damos cuenta que al estandarizar este coeficiente queda mas pequeño que respecto al normalizado.

1.1.2. Gradiante Descendente Online

El segundo algoritmo a usar es el Gradiente Descendente Online, este algoritmo se caracteriza por lo siguiente, a diferencia del Gradiente Descendente Batch, este no sumara todos los errores cuadráticos y luego actualizara los parámetros, si no que calculara un solo error cuadrático de un vector de nuestra data de entrenamiento respecto a nuestros objetivos, actualizara los parámetros y luego repetirá este proceso con toda la data de entrenamiento que dispongamos, hasta que nuestros parámetros converjan a algún valor. Acá el Learning Rate esta también presente, cumpliendo el mismo rol que en Gradiente Descendente Batch.
Al igual que Batch este algoritmo busca encontrar el mínimo del la función de costo, en este caso la de errores cuadráticos.

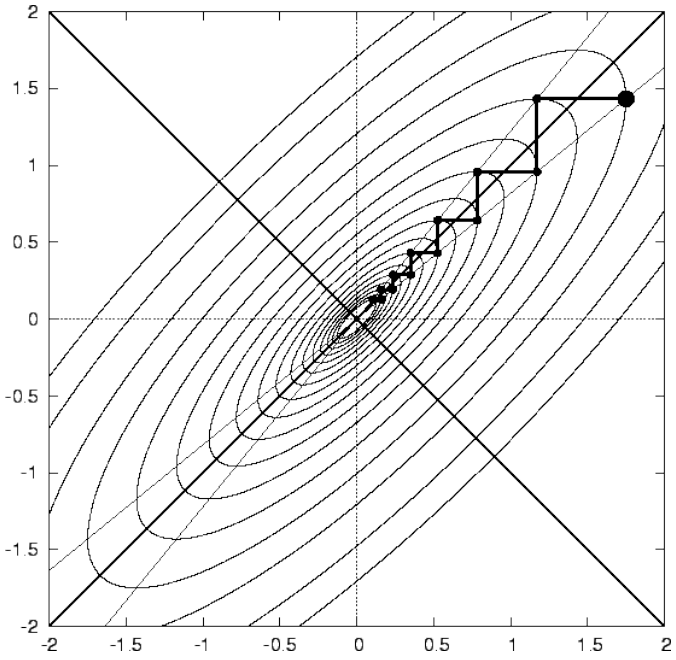

Figura 11: Gradiante Descendente Online busca el minimo de la función de perdida

Es destacable la diferencia entre Online y Batch, Online al estar constantemente actualizándose tiene un desplazamiento en zig-zag buscando el mínimo, en cambio Batch no muestra ese comportamiento.

El algoritmo de Gradiente descendente Online es repetir la regla de Widrow-Hoff hasta que los parámetros converjan a algún valor.

$$\beta_j^{\rho+1} = \beta_j^{\rho} - \alpha(f(x^{(i)}) - y^{(i)})x_j^{(i)}$$

Ahora usando Gradiente Descendente Online y utilizando el Dataset de Cereales, se correrán ciertas pruebas sobre 20 Training Set previamente generados. Observaremos como se comporta este algoritmo en los diferentes contextos, además de ver la relevancia del Learning Rate.
Se comenzó con analizar los datos sin ninguna intervención, se usaron 5 α candidatos bien pequeños, para que el algoritmo converja, estos fueron $2x10^{-6}, 4x10^{-6}, 6x10^{-6}8x10^{-6}, 10^{-5}$x. Utilizando 5 cross validation sobre los 20 Dataset previamente generados obtuvimos los siguientes errores cuadráticos:

TrainingSet3	2*10(-6)	4*10(-6)	6*10(-6)	8*10(-6)	10(-5)
Media	341.3107	332.4096	337.4961	359.7012	408.3957
Desviación	44.75974	26.54982	35.02080	45.34047	71.56078
Coeficiente de variación	0.13114077	0.07987082	0.10376653	0.12605037	0.17522414

TrainingSet13	2*10(-6)	4*10(-6)	6*10(-6)	8*10(-6)	10(-5)
Media	368.6653	378.5075	383.1473	387.3310	427.5143
Desviación	198.9555	173.8191	140.8075	134.0341	157.0416
Coeficiente de variación	0.5396642	0.4592223	0.3675022	0.3460454	0.3673365

Basandonos en nuestros criterios ya establecidos, el ganador en el TrainingSet3 es $4x10^6$ debido que tiene la media y coeficiente de variación mas pequeños, en el TrainingSet13 el ganador fue $2x10^6$, ya que tiene la media mas pequeña.

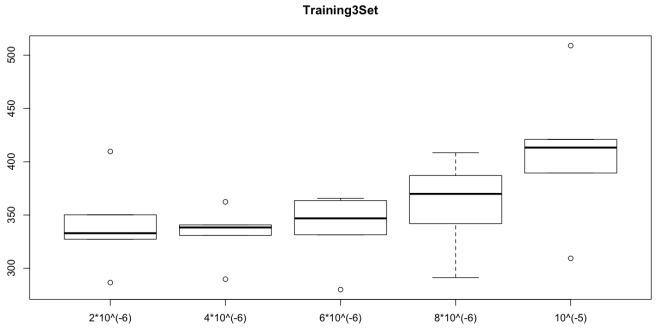

Figura 12: Boxplot de los distintos alpha del trainingset 3

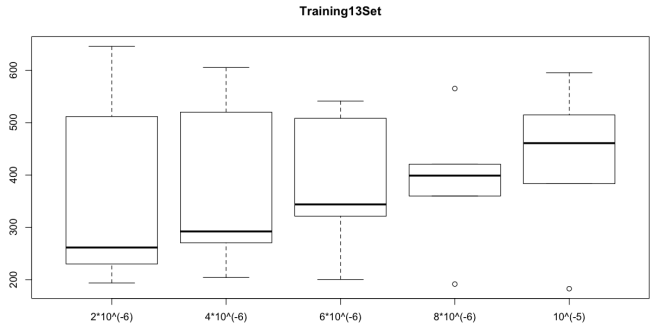

Figura 13: Boxplot de los distintos alpha del trainingset 13

Repetidiendo lo mismo para los 18 Dataset restantantes obtenemos que el α ganador fue $2x10^{-6}$.

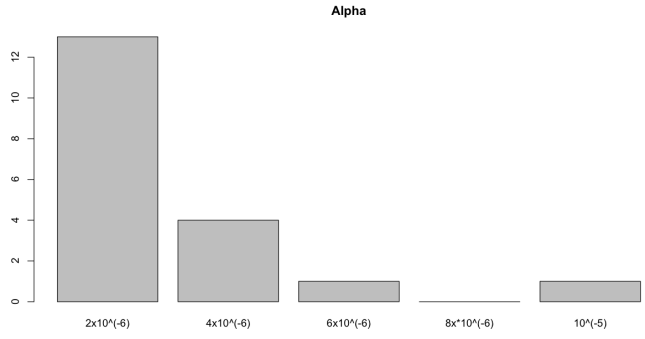

Figura 14: Diagrama de Barra de los distintos alpha

Ahora repetiremos el mismo proceso, con los mismos α, pero esta vez los datos serán estandarizados.

TrainingSet1	10$\hat{}$(-4)	2*10(-4)	10(-3)	2*10(-3)	10(-5)
Media	214.81643	204.77200	117.17982	91.01885	221.09890
Desviación	100.50506	99.03562	83.62469	58.14595	103.53550
Coeficiente de variación	0.4678649	0.4836385	0.7136441	0.6388342	0.4682769

TrainingSet2	10$\hat{}$(-4)	2*10(-4)	10(-3)	2*10(-3)	10(-5)
Media	200.18522	143.99973	54.73867	42.14816	207.02920
Desviación	108.49907	74.86137	24.68278	14.89026	111.45355
Coeficiente de variación	0.5419934	0.5198716	0.4509204	0.3532839	0.5383470

Basándonos en nuestros criterios ya establecidos, el ganador es $2x10^{-3}$ en ambos TrainingSet debido a que el error cuadrático medio es menor ahí.

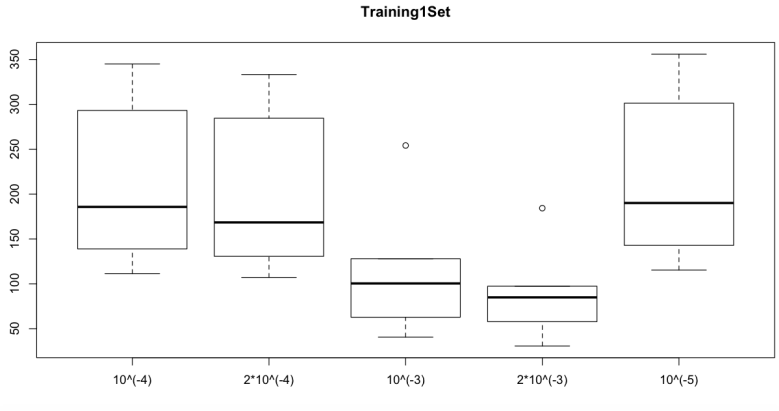

Figura 15: Boxplot de los distintos alpha del trainingset 1

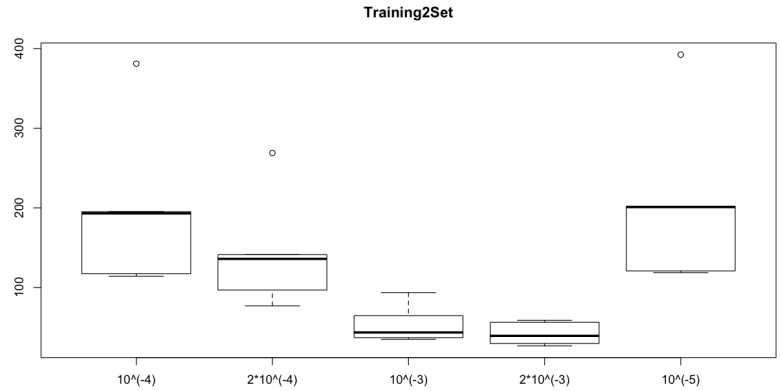

Figura 16: Boxplot de los distintos alpha del trainingset 2

Repetidiendo lo mismo para los 18 Dataset restantantes obtenemos que el α ganador fue $2x10^{-3}$.

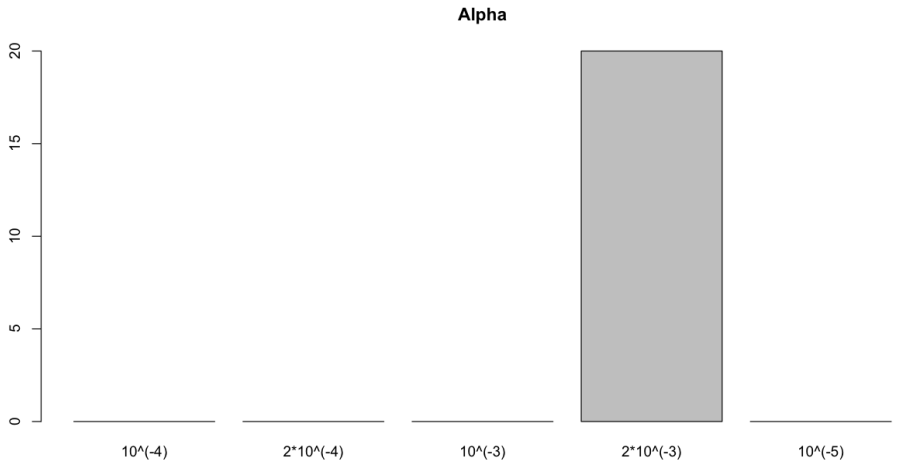

Figura 17: Diagrama de Barra de los distintos alpha

Finalmente nos queda hacer lo mismo, pero ahora con los datos normalizados entre 0 y 1. Repetiremos el mismo proceso para los 20 datasets.

TrainingSet6	10$\hat{}$(-4)	2*10(-4)	10(-3)	2*10(-3)	10(-5)
Media	802.3912	756.8119	291.0138	224.9752	846.7142
Desviación	177.1473	175.7226	131.0534	112.9220	178.3880
Coeficiente de variación	0.2207742	0.2321879	0.4503338	0.5019308	0.2106827

TrainingSet13	10$\hat{}$(-4)	2*10(-4)	10(-3)	2*10(-3)	10(-5)
Media	959.0752	904.4238	299.0349	238.8836	1012.0864
Desviación	435.7516	424.6869	217.9921	171.5495	446.0963
Coeficiente de variación	0.4543456	0.4695662	0.7289854	0.7181300	0.4407690

Siguiendo los criterios establecidos anteriormente, claramente el ganador seria $2x10^{-3}$, debido que el error cuadrático es el menor en ambos TrainingSets.

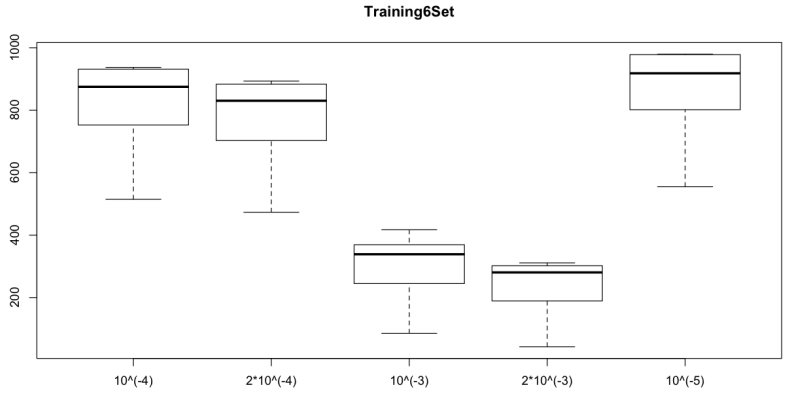

Figura 18: Boxplot de los distintos alpha del trainingset 3

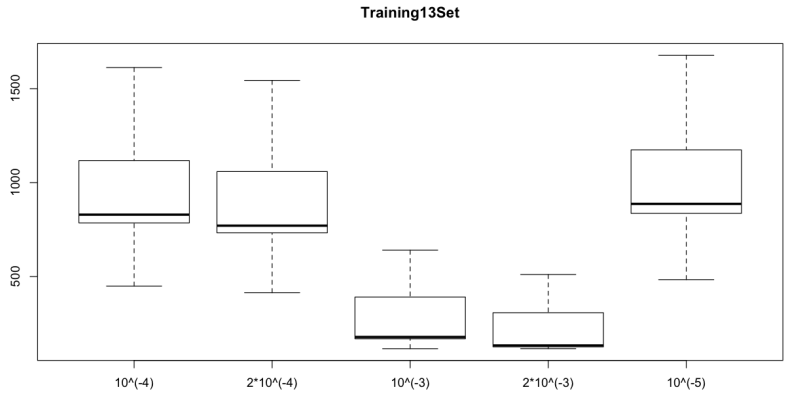

Figura 19: Boxplot de los distintos alpha del trainingset 13

Repetidiendo lo mismo para los 18 Dataset restantantes obtenemos que el α ganador fue $2x10^{-3}$.

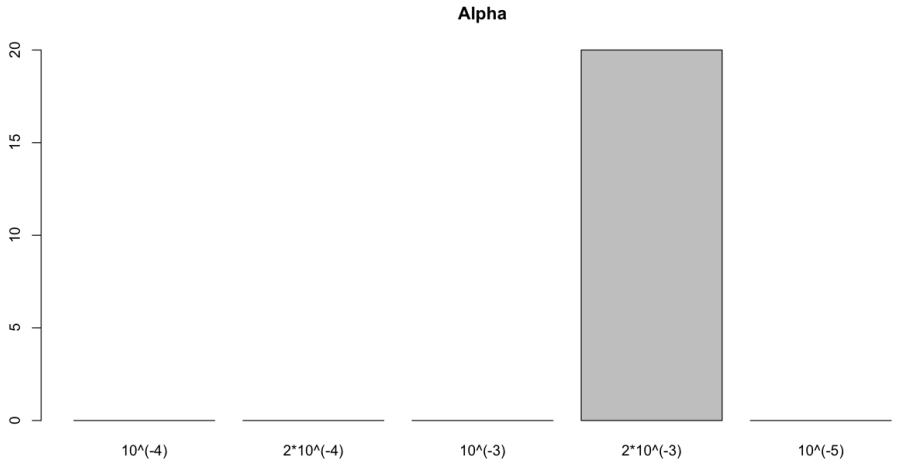

Figura 20: Boxplot de los distintos alpha del trainingset 3

Luego de haber utilizado nuestro algoritmo Gradiente descendente online logramos notar ciertas cosas:

1. Observamos la influencia del Learning Rate, claramente al hacerlo variar se observa un aumento o disminución del error cuadrático medio.

2. Al igual que en gradiente descendente batch, observamos que el error cuadrático medio es menor en los datos estandarizados que en los normalizados 0 y 1.

3. Observamos también un cambio notable en el coeficiente de variación, a medida que el error cuadrático medio se vuelve mas pequeño el coeficiente de variación tiende a crecer, eso se debe a que la desviación va disminuyendo a medida que hacemos variar el α.

1.2. Newton Raphson

El método de Newton es un conocido método iterativo utilizado para encontrar las raíces (ceros) de una función, básicamente lo que hace el método de Newton, es evaluar un punto x_n en la función f(x) (un punto relativamente cerca del cero de la función), y donde se encuentra el punto f(x_n) trazar una tangente a la función, el punto donde intersecte esta tangente será el nuevo x_{n+1}. Gráficamente se vería así:

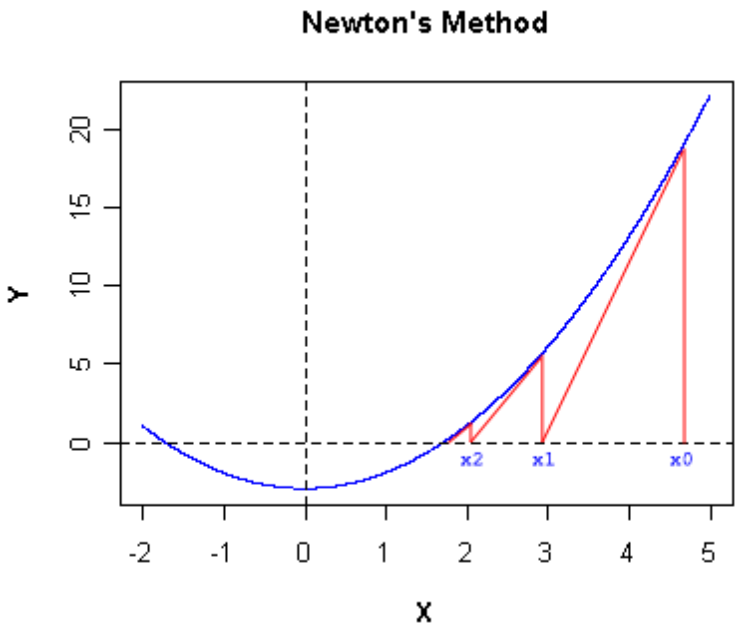

Figura 21: Metodo de Newton

La ecuación de la recta tangente a la curva f(x) en un punto igual a x = x_n es:

$$y = f'(x_n)(x - x_n) + f(x_n)$$

Ahora el intercepto de esta recta es usada para determinar el x_{n+1}, por lo tanto si hacemos y = 0, x = x_{n+1} tendremos:

$$0 = f'(x_n)(x_{n+1} - x_n) + f(x_n)$$

Ahora si ordenamos los términos tendremos:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Que corresponde ser la regla de Actualización de Newton. Ahora si llevamos este caso a un problema de optimización, es decir en vez de buscar raíces queremos encontrar los máximos o mínimos de una función, eso es equivalente a decir en encontrar los 0 de la derivada de la función, ya que sabemos que cuando la derivada de la función es 0, existirá un máximo o un mínimo, y solo bastaría ver la segunda derivada para ver de cual de las dos es, si la segunda derivada es negativa entonces tenemos un máximo, y si la segunda derivada es positiva tenemos un mínimo. Entonces basándonos en este razonamiento tendríamos:

$$x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)}$$

Que es conocido como el método de Newton-Raphson, que a diferencia del método de Newton este busca mínimos o máximo de la función.

Ahora que conocemos la regla de actualización que usaremos para calcular nuestros parámetros, es interesante analizar esta regla de actualización de Manera matricial. Recordar que estamos en el contexto de regresión lineal, por lo que nuestra función de costo es la de errores cuadráticos, esa función es la que nosotros buscamos minimizar, ya conocemos su derivada que era:

$$(f(x_m) - y_m)x_m^{(i)}$$

Y de manera matricial

$$X^T(X\beta - Y)$$

Ahora la segunda derivada, correspondería ser al Hessiano, que corresponde ser:

$$H = X^T X$$

Ahora si la remplazamos esto en nuestra método de Newton-Raphson nos queda:

$$\beta^{p+1} = \beta^p - (X^T X)^{-1}(X^T(X\beta - Y))$$

Si jugamos con las matrices, finalmente obtendremos:

$$\beta^{p+1} = (X^T X)^{-1} X^T Y$$

Que corresponden ser a las solución normal de error cuadrático medio.

Ahora calcularemos el error cuadrático medio, utilizando este algoritmo, para datos normalizados, estandarizados y sin ninguna intervención. Newton-Raphson no tiene α involucrados, por lo que usaremos el five folder para observar como básicamente se comporta Newton los diferentes contextos. Se comenzó con analizar los datos sin ninguna intervención.

Newton-Raphson	TrainingSet1	TrainingSet2	TrainingSet3	TrainingSet4	TrainingSet5
Media	25.63053	19.95975	22.97354	23.81599	29.49619
Desviación	6.086877	12.75822	8.596659	16.09104	15.82916
Coeficiente de Variación	0.2374854	0.6391974	0.3741984	0.6756402	0.5366511

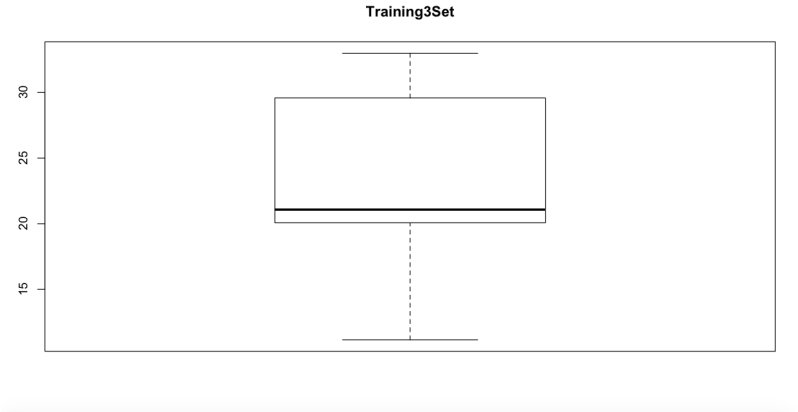

Figura 22: Metodo de Newton

Ahora veremos como se comporta cuando los datos estan standarizados.

Newton-Raphson	TrainingSet1	TrainingSet2	TrainingSet3	TrainingSet4	TrainingSet5
Media	25.63053	19.95975	22.97354	23.81599	29.49619
Desviación	6.086877	12.75822	8.596659	16.09104	15.82916
Coeficiente de Variación	0.2374854	0.6391974	0.3741984	0.6756402	0.5366511

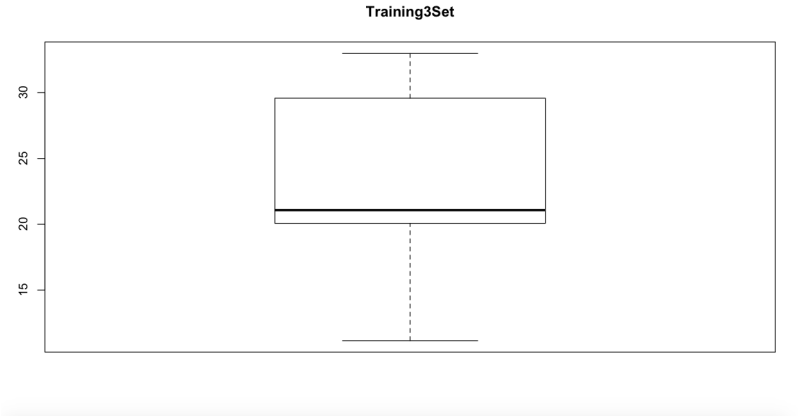

Figura 23: Boxplot del TrainingSet 3

Y finalmente veremos como se comporta con datos normalizados entre 0 y 1

Newton-Raphson	TrainingSet1	TrainingSet2	TrainingSet3	TrainingSet4	TrainingSet5
Media	25.63053	19.95975	22.97354	23.81599	29.49619
Desviación	6.086877	12.75822	8.596659	16.09104	15.82916
Coeficiente de Variación	0.2374854	0.6391974	0.3741984	0.6756402	0.5366511

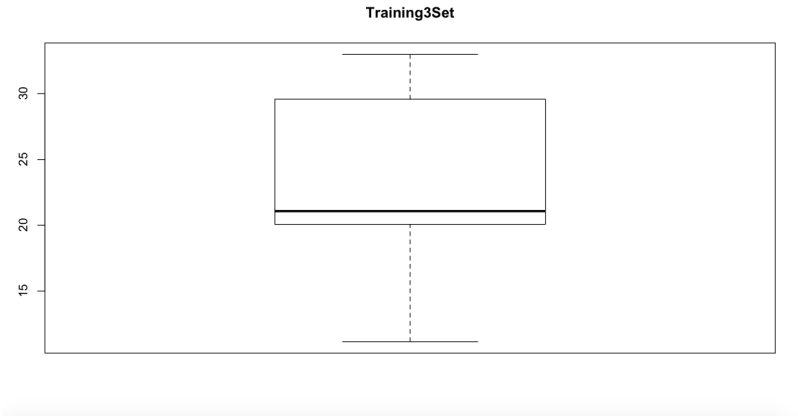

Figura 24: Boxplot del TrainingSet 3

Luego de ocupar el algoritmo Newton-Raphson, inmediatamente notamos algo muy interés ante, que el error cuadrático medio son los mismos, ya sea que los datos estén normalizados, estandarizados o sin haber hecho algún tipo de intervención.

1.3. Locally Weighted Regression

Uno de los problemas de la regresión lineal es que siempre lo que intenta es modelar una recta para toda la Data que tengamos, esto esta bien siempre cuando la Data un comportamiento linear y no tiene mucho ruido, pero cuando no es linear, se tiende a reajustar la Data de entrenamiento.
El algoritmo Locally Weighted regression intenta arreglar este problema asignadoles pesos a la Data de entrenamiento, los Datos que están mas cerca de la Data que intentamos predecir tendrán un peso mayor que los que están mas lejos, de ahí viene el nombre Locally. Un punto que hay que tener presente es que este algoritmo tiene un costo computacional bien alto, debido que cada vez que queramos predecir un punto, necesitamos toda la data nuevamente, tenemos que hacer eso debido a lo que queremos es hacer una regresión lineal cercano al punto de interés.
Este algoritmo es muy similar a los anteriores, la gran diferencia es que tenemos que calcular una matriz de pesos, en la que en su diagonal están los pesos de cada dato.

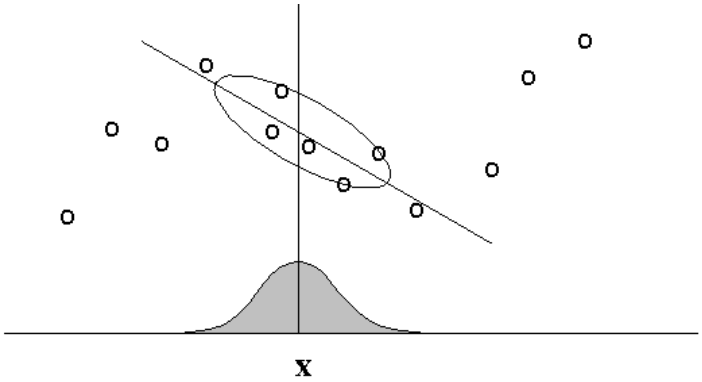

Figura 25: Metodo de Newton

Ahora siguiendo la misma idea que el gradiente descendente, en vez de minimizar nuestra función de del error cuadrático medio, minimizaremos esta:

$$J(\beta) = \frac{1}{2}\sum_{m=1}^{M} w_m (f(x_m) - y_m)^2$$

Que es lo mismo, solo que va multiplicada por el peso que se le asigna a cada dato, para calcular dichos usaremos el siguiente kernel, hay otros, pero cabe recalcar, que es fundamental para este algoritmo:

$$w_m = e^{\frac{(-(x_m - x)^T (x_m - x)}{2\tau^2}}$$

Donde τ es el ancho de Banda. Notar que los pesos dependen de los datos de algún Testing Set, sin ellos este algoritmo no funcionara. Como lo hicimos anteriormente podemos escribir este algoritmo de manera matricial.

$$\beta = (X^T W X)^{-1} X^T W Y$$

Ahora lo que haremos igual que los algoritmos anteriores, es estudiar el algoritmo, pero en vez de hacer variar α, variaremos τ, del mismo modo nos pondremos en los 3 contextos, donde los datos son estandarizados, normalizados y sin algún tipo de intervención. Comenzaremos con los últimos mencionados.

TrainingSet1	100	1000	10000	100000	1000000
Media	22.64876	19.23192	19.22298	19.22290	19.22290
Desviación	11.158203	4.667806	4.566184	4.565168	4.565158
Coeficiente de Variación	0.4926629	0.2427114	0.2375377	0.2374859	0.2374854

TrainingSet1	100	1000	10000	100000	1000000
Media	21.10724	14.97182	14.96983	14.96982	14.96982
Desviación	11.152416	9.561295	9.568589	9.568667	9.568668
Coeficiente de Variación	0.5283692	0.6386193	0.6391915	0.6391974	0.6391974

Observamos que hacer variar nuestro bandwith al principio tiene una influencia significarte en los valores entregados en el error cuadrático medios, pero ya en los 2 últimos bandwidth el cambio es mínimo, solo es observable en la desviación, en este caso elegimos como ganador 1000000.

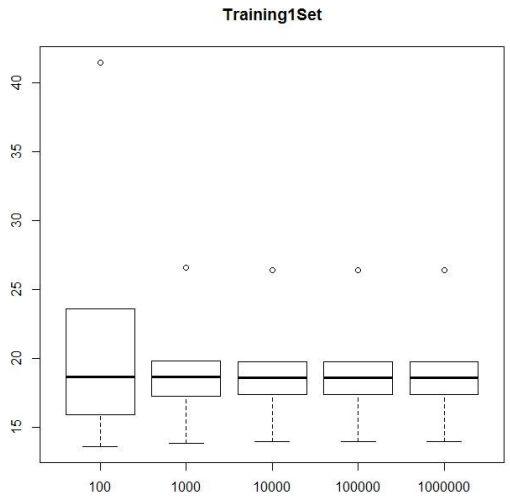

Figura 26: Boxplot del TrainingSet 1

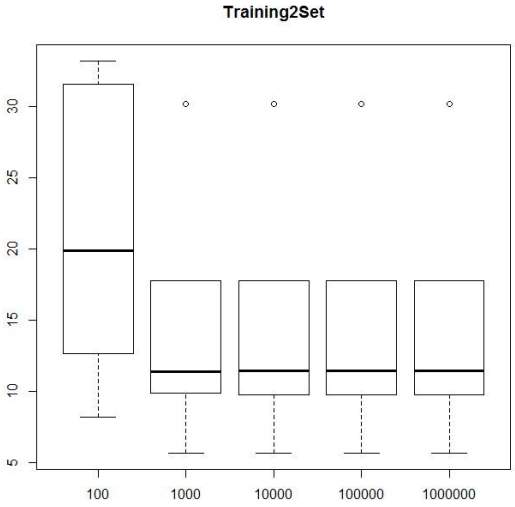

Figura 27: Boxplot del TrainingSet 2

Ahora veremos como se comporta el bandwidth con los datos estandarizados.

TrainingSet1	100	1000	10000	100000	1000000
Media	19.22292	19.22290	19.22290	19.22290	19.22290
Desviación	4.565200	4.565158	4.565157	4.565157	4.565157
Coeficiente de Variación	0.2374873	0.2374854	0.2374854	0.2374854	0.2374854

TrainingSet2	100	1000	10000	100000	1000000
Media	14.96978	14.96982	14.96982	14.96982	14.96982
Desviación	9.568645	9.568667	9.568668	9.568668	9.568668
Coeficiente de Variación	0.6391976	0.6391974	0.6391974	0.6391974	0.6391974

Observamos que luego de estandarizar los datos, al hacer variar el bandwidth no afecta en la media y el cambio de la desviación es bajo. No podemos elegir un ganador, debido a que con tres bandwidth llegamos al mismo error cuadrático medio y la misma desviación.

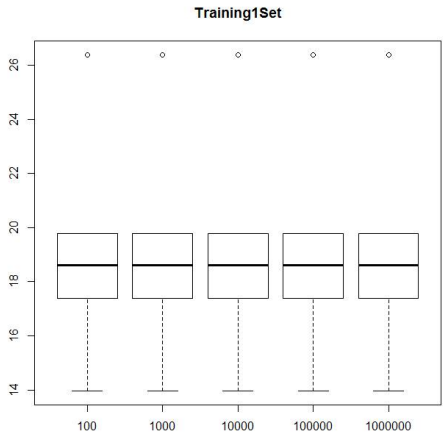

Figura 28: Boxplot del TrainingSet 1

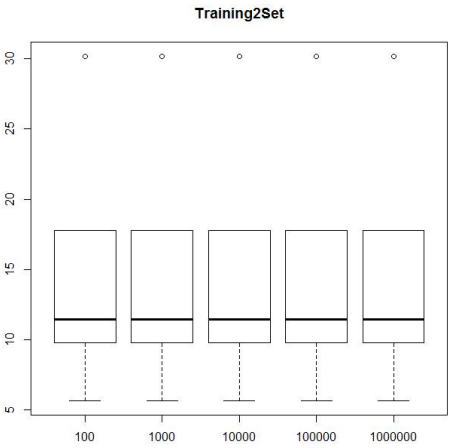

Figura 29: Boxplot del TrainingSet 2

Finalmente observaremos que sucede cuando los datos son normalizado en el rango 0 y 1.

Training1Set	100	1000	10000	100000	1000000
Media	19.22292	19.22290	19.22290	19.22290	19.22290
Desviación	4.565200	4.565158	4.565158	4.565158	4.565158
Coeficiente de Variación	0.2374873	0.2374854	0.2374854	0.2374854	0.2374854

Training2Set	100	1000	10000	100000	1000000
Media	14.96978	14.96982	14.96982	14.96982	14.96982
Desviación	9.568645	9.568668	9.568668	9.568668	9.568668
Coeficiente de Variación	0.6391976	0.6391974	0.6391974	0.6391974	0.6391974

Observamos que al normalizar los datos, llegamos al mismo resultado que si hubiésemos estandarizados los datos. Acá al igual que en caso anterior no podemos elegir un ganador, debido a que llegamos a tres bandwidth con la misma media y desviación.

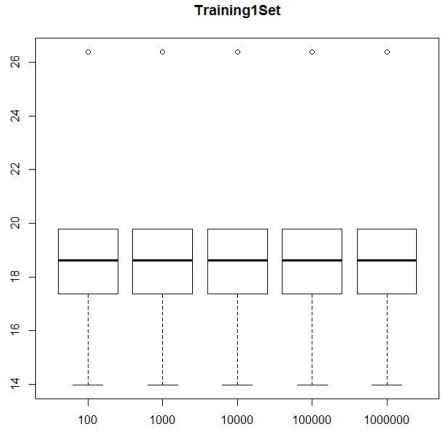

Figura 30: Boxplot del TrainingSet 1

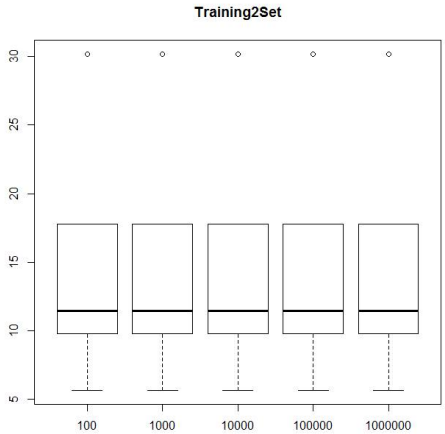

Figura 31: Boxplot del TrainingSet 2

Como en datos normalizados y estandarizados no podemos llegar a una decisión concluyente, concluiremos que el mejor bandwitdth que obtuvimos en los datos sin ninguna intervención, que fue 1000000.

Luego de haber visto como se comportaban los algoritmos bajo distintos contextos, usando los α o τ correspondientes, o para el caso de los algoritmos sin algún valor de ajuste, volveremos a utilizar estos algoritmos, pero esta vez los entrenaremos sobre los Training Set, luego mediremos el error cuadrático medio tras estimar los targets del mismo Training set y de un Testing Set.

Training Set	Batch	Online	Newton-Raphson	Locally Weight
Media	309.8028	328.1712	14.81462	14.81462
Desviación	19.83523	24.45678	1.746402	1.746402
Media (Standard)	17.79835	77.40158	14.81462	14.81462
Desviación (Standard)	6.524362	15.72179	1.746402	1.746402
Media (Normalizado)	194.8366	219.9336	14.81462	14.81462
Desviación (Normalizado)	13.09829	16.14001	1.746402	1.746402

Testing Set	Batch	Online	Newton-Raphson	Locally Weight
Media	309.7607	310.1051	23.26980	23.26980
Desviación	74.74610	89.78506	7.623893	7.623893
Media (Standard)	36.54481	79.89400	32.44852	32.44852
Desviación (Standard)	19.569919	17.85889	17.080320	17.080320
Media (Normalizado)	230.7128	245.3310	97.79643	97.79643
Desviación (Normalizado)	90.17993	82.81698	68.333786	68.333786

Los resultados obtenidos muestran el error cuadrático medio y desviación obtenidos sobre todos los training set y testing set, podemos observar que existe un aumento en el error cuadrático medio en el caso de Batch y Online, lo mismo pasa con la desviación, observamos también que se obtiene en general un error cuadrático menor cuando los datos son estandarizados, otra cosa que sin duda hay que destacar es sobre por que Newton-Raphson y Locally Weight llegan a los mismos errores cuadráticos, como vimos anteriormente, Newton-Raphson en regresión lineal es equivalente a la ecuación normal, ahora en el caso de Locally Weigth, cuando calculamos el peso, al ser el bandwith tan grande, como es 1000000, al elevarlo al cuadrado queda 1000000000000, un numero suficientemente grande para hacer 0 al numerador que correspondería a la diferencia entre x_m y x, quedándonos finalmente e^0 siendo eso igual a 1, por lo tanto queda como resultado una matriz diagonal de puros 1, por lo tanto el termino que conocíamos como

$$(X^T W X)^{-1}$$

Seria equivalente a

$$(X^T X)^{-1}$$

Del mismo modo el termino de la ecuación matricial que era

$$X^T W Y$$

Queda como

$$X^T Y$$

Quedándonos como resultado

$$(X^T X)^{-1} X^T Y$$

Que es la ecuación normal, y que es la misma ecuación que utilizamos para Newton-Raphson, por lo que explicaría la igualdad de errores cuadráticos entre Newton y Locally.

Ahora mostraremos boxplot comparativos entre los resultados entre el TrainingSet y el TestingSet.

Boxplot del Algoritmo Gradiante Descendente Batch

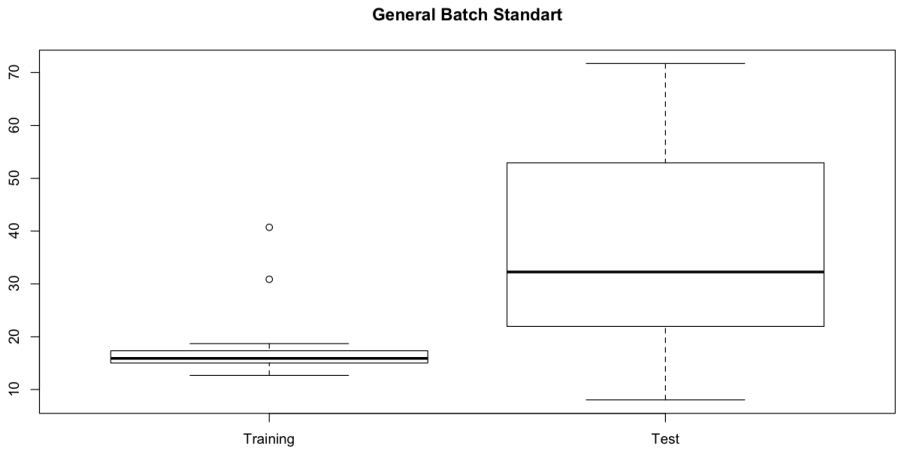

Figura 32: Boxplot Comparativo Gradiante Descendente Batch

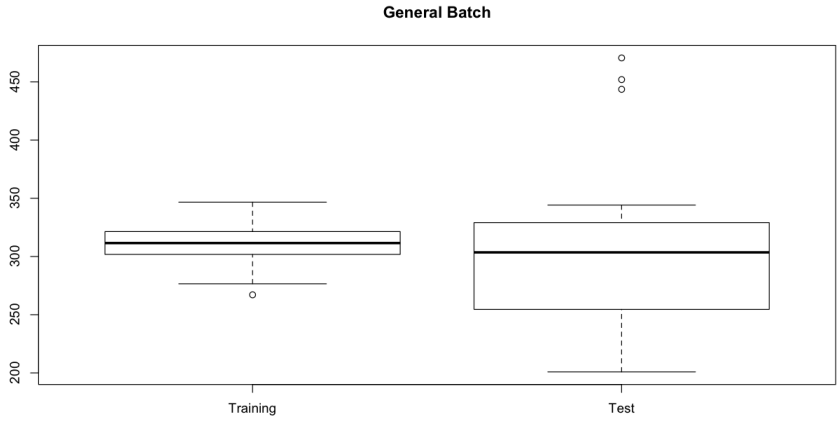

Figura 33: Boxplot Comparativo Gradiante Descendente Batch

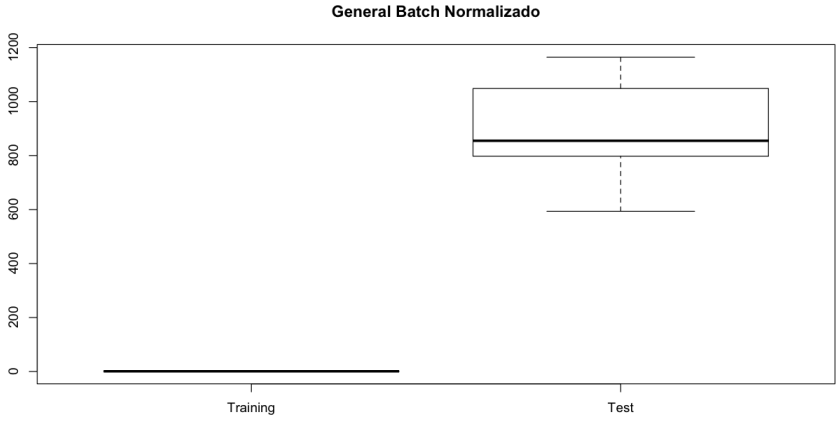

Figura 34: Boxplot Comparativo Gradiante Descendente Batch

Boxplot del Algoritmo Gradiante Descendente Online.

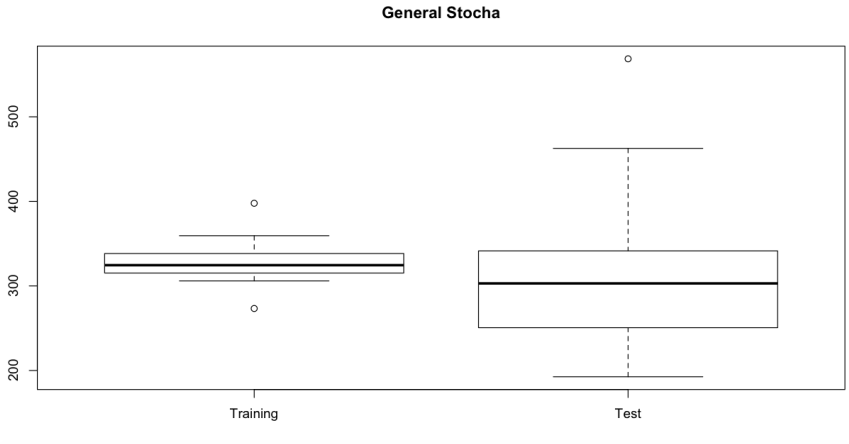

Figura 35: Boxplot Comparativo Gradiante Descendente Batch

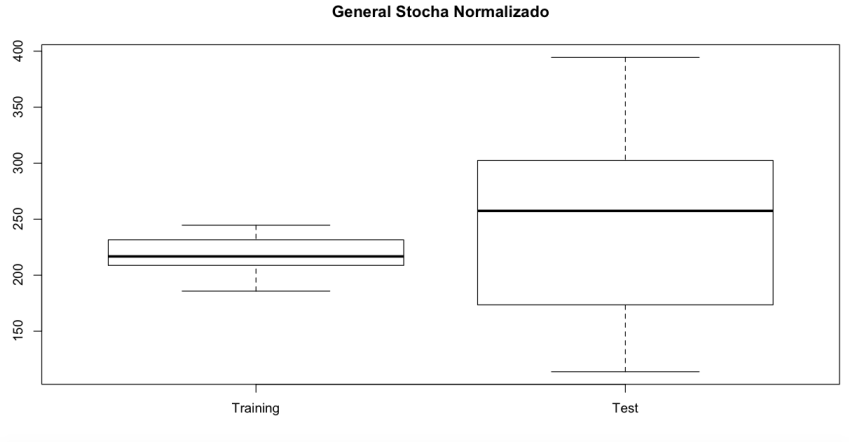

Figura 36: Boxplot Comparativo Gradiante Descendente Batch

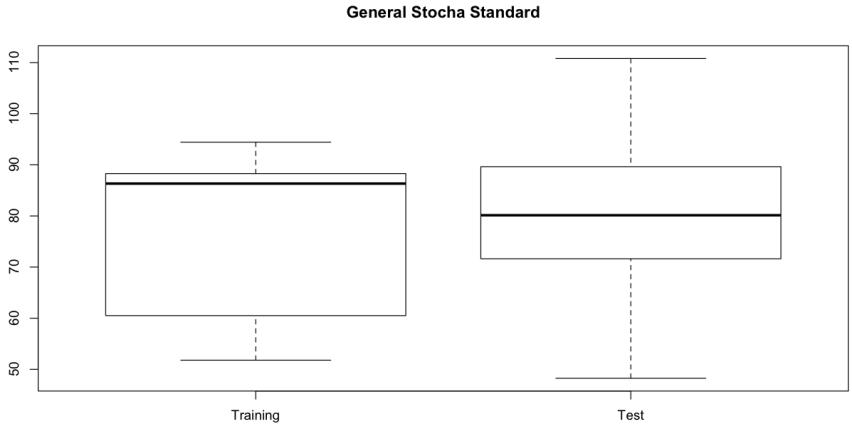

Figura 37: Boxplot Comparativo Gradiante Descendente Batch

Boxplot de Newton-Raphson

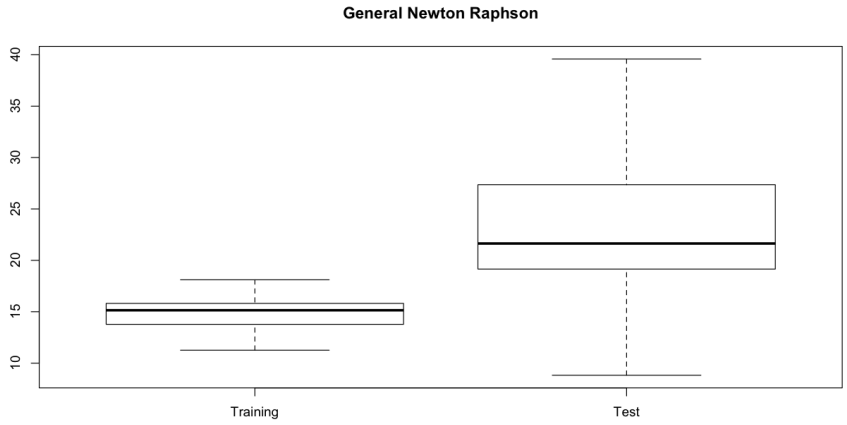

Figura 38: Boxplot Comparativo Newton-Raphson

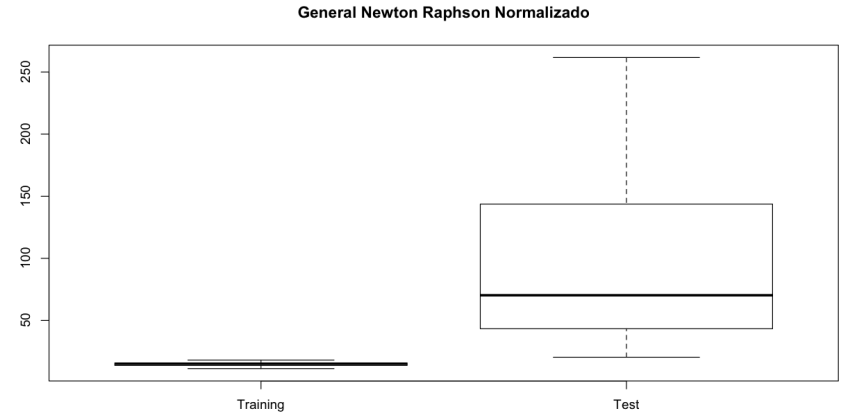

Figura 39: Boxplot Comparativo Newton-Raphson

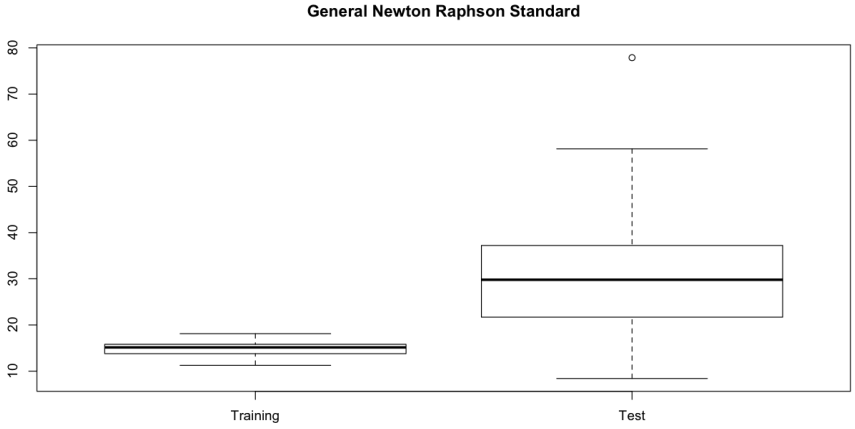

Figura 40: Boxplot Comparativo Newton-Raphson

Boxplot de Locally Weight

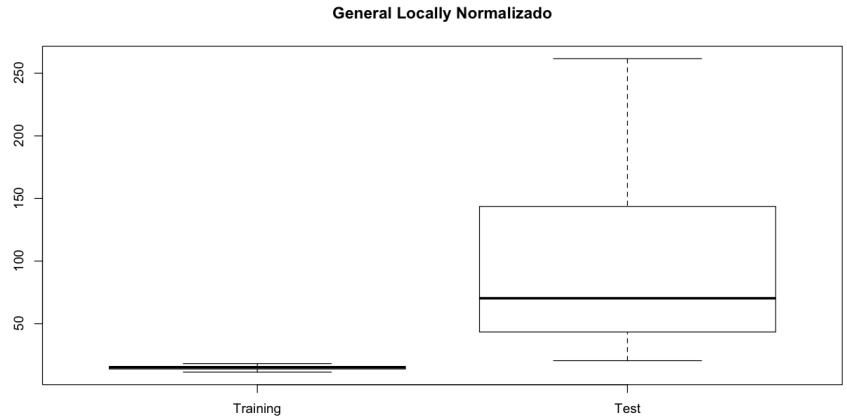

Figura 41: Boxplot Comparativo Locally Weight

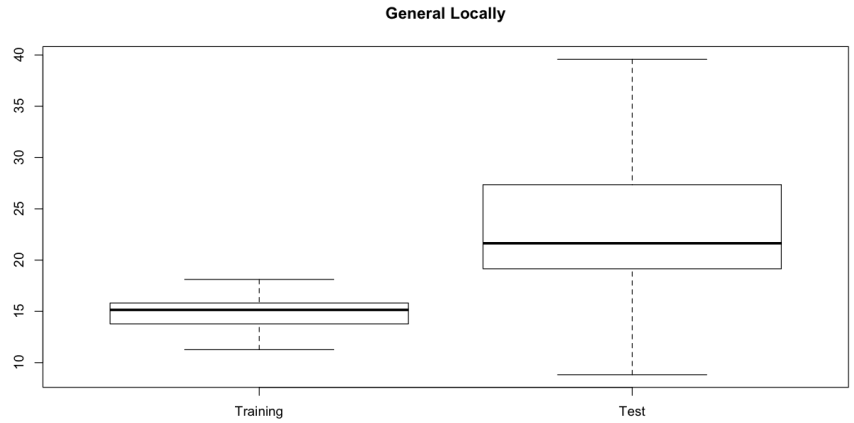

Figura 42: Boxplot Comparativo Locally Weight

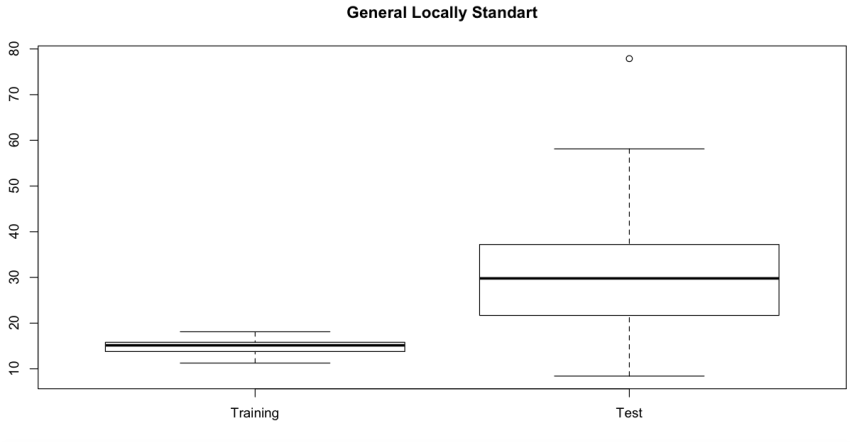

Figura 43: Boxplot Comparativo Locally Weight

Para finalizar regresión lineal, usaremos Locally Weight para predecir los resultados del Archivo Test 1, y escribiremos los targets, y los valores predichos, para observar a simple vista los resultados.

Predicción	Target
66.89541	74.473
37.60018	35.114
46.96031	46.659
21.92413	23.804
27.26564	31.436
47.36219	53.371
68.42010	64.534
55.30787	52.077
36.91234	40.449
37.31009	35.783
57.73985	50.765
37.97773	40.400
37.09700	33.174
37.74488	38.840
26.80491	29.924
56.37647	53.131
48.47087	39.241
31.75505	32.208
45.40899	45.328
29.73682	27.753
41.23068	43.671
45.38617	44.664
30.21512	24.442
44.01522	40.917
45.11082	41.999

2. Regrecion Logistica

En esta sección utilizaremos distintos algoritmos que están relacionados con regresión logística binaria, este tipo de regresión es utilizada para predecir el resultado de una variable categoría, y es de gran interés, debido a que muchas cosas en la realidad se modelan bajo este tipo de clasificación, la presencia o no de una enfermedad, la presencia o no de alguna partícula, la toma o el rechazo de de alguna decisión, etc. Utilizaremos una función llamada función logística, que básicamente es la base de este modelo.

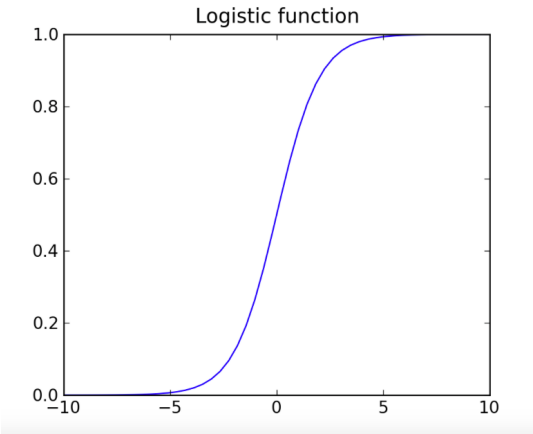

Figura 44: Grafica de la función Logistica

Esta función que se escribe como:

$$f_\beta(x) = \frac{1}{1+e^{-\beta^T x}}$$

Y tiene la característica, como podemos ver en la grafica, que cuando esta función es evaluada su imagen va entre los rangos 0 y 1. Ahora llamemos z a $\beta^T x$ y encontremos la derivada de la función.

$$\begin{aligned} f'_\beta(x) &= \frac{d}{dz}\frac{1}{1+e-z} = \frac{1}{(1+e^{-z})^2}(e^{-z}) \\ &= \frac{1}{(1+e^{-z})}(1-\frac{1}{(1+e^{-z})}) \\ &= f_\beta(x)(1-f_\beta(x)) \end{aligned}$$

Ahora, asumiremos los siguiente, que la probabilidad que alguna variable sea parte de la categoría del 1 o del 0, la podemos modelar con nuestra función logística. Quedaría lo siguiente:

$$\begin{aligned} \text{P(y = 1|x; } \beta) &= f_\beta(x) \\ \text{P(y = 0|x; } \beta) &= \text{1- } f_\beta(x) \end{aligned}$$

Ahora si asumimos que estos sucesos se distribuyen Bernoulli entonces tendremos la función de probabilidad:

$$\text{P(y|x; } \beta) = (f_\beta(x))^y(1-f_\beta(x))^{1-y}$$

Asumiendo ahora independencia entre cada vector perteneciente a un conjunto de vectores, por ejemplo, algún Training Set, podemos escribir la función de máxima verosimilitud de todos los vectores quedaría como:

$$\begin{aligned} L(\beta) &= \prod_{m=1}^{M} P(y_m|x_m;\beta) \\ &= \prod_{m=1}^{M}(f_\beta(x_m))^{y_m}(1-f_\beta(x_m))^{1-y} \end{aligned}$$

Ahora si deseamos dejarla como sumatoria, le aplicamos logaritmo, que básicamente es la función Log de máxima verosimilitud.

$$\begin{aligned} \ell(\beta) &= \sum_{m=1}^{M} y_m log(f_\beta(x_m)) + (1-y_m) log(1-f_b(x_m)) \\ &= \sum_{m=1}^{M} y_m \beta^T x_m - log(1-e^{\beta_T x_m}) \end{aligned}$$

Ahora si derivamos la función ℓ obtendremos :

$$\frac{\varphi}{\varphi\beta_i}\ell = y\frac{1}{f_\beta(x)} - (1-y)\frac{1}{1-f_\beta(x)})\frac{\varphi}{\varphi\beta_i}\beta^T x$$
$$= (y\frac{1}{f_\beta(x)} - (1-y)\frac{1}{1-f_\beta(x)})f_\beta(x)(1-f_\beta(x))\frac{\varphi}{\varphi\beta_i}\beta^T x$$
$$= (y(1-f_\beta(x)) - (1-y)f_\beta(x))x^i$$
$$(y - f_b(x))x^{(i)}$$

El objetivo de encontrar esta función de maxima verosimilitud, es debido a que es la función de costo que maximizaremos, y para maximizar necesitamos la derivada. En el fondo estamos buscando la máxima probabilidad de que el evento suceda, de esta manera al conocer la probabilidad de esta podemos discriminar ese suceso, elegir a que etiqueta corresponde, conoceremos como frontera el lugar de corte de la discriminación, es decir si a nuestra grafica de función logística, le trazamos la frontera justo en el 0.5 quedaría:

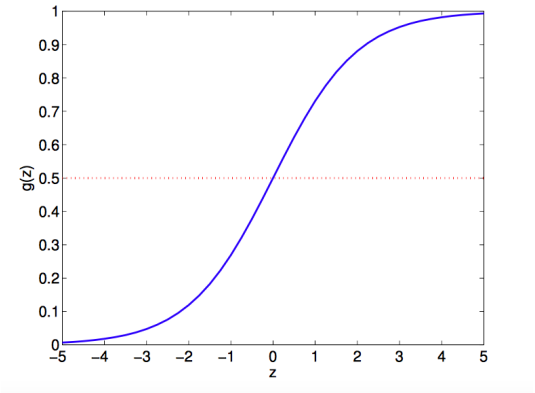

Figura 45: Grafica de la función logistica

Los puntos que se encuentren bajo la recta en 0.5 corresponderán hacer de la etiqueta del 0, y sobre esta de la etiqueta del 1.

2.1. Gradiente Descendente

Al igual que regresión, utilizaremos la regla de actualización del gradiente descendente, pero con un cambio, debido a que estamos maximizando cambia el signo de la regla de actualización, lo cual queda:

$$\beta_j^{\rho+1} = \beta_j^\rho + \alpha\frac{\varphi}{\varphi\beta_j}J(\beta)$$

La gran diferencia es que esta vez estaremos maximizando, debido que buscamos que la probabilidad sea máxima, y la segunda gran novedad es la función de costo que usaremos, que en este caso es la de Log máxima verosimilitud.

2.1.1. Gradiente Ascendente Online

Como obtuvimos la derivada de la función Log máxima verosimilitud, podemos utilizar nuestro algoritmo gradiente online, solo que esta vez es ascendente, por lo que existe un cambio de signo si lo comparamos con el descendiente online, la regla de actualización nos queda de la siguiente manera:

$$\beta_j^{\rho+1} = \beta_j^\rho + \alpha(f_\beta(x^{(i)}) - y^{(i)})x_j^{(i)}$$

Iremos actualizando hasta que converja. Al igual que hicimos en regresión lineal, observaremos como se comporta el algoritmo en presencia de datos estandarizados, normalizados y sin ninguna intervención, cabe destacar que el α cumple el mismo rol que en la sección anterior. Otra cosa importante es que acá no mediremos error cuadrático medio, si no que error rate, como es un problema de clasificación binaria, nuestra función predictora arrojara números 0 y 1, los compararemos con los targets, y mediremos la proporción entre los errores cometidos por la función con el total. Comenzaremos con datos sin alguna intervención.

TrainingSet 10	10(-4)	10(-3)	10(-3)	2*10(-3)	10(-5)
Media	0.5444444	0.5555556	0.4555556	0.4555556	0.6111111
Desviación	0.09938080	0.08784105	0.19002924	0.19002924	0.15214515
Coeficiente de Variación	0.1825362	0.1581139	0.4171374	0.4171374	0.2489648

TrainingSet 15	10(-4)	10(-3)	10(-3)	2*10(-3)	10(-5)
Media	0.3000000	0.3111111	0.4444444	0.4555556	0.4333333
Desviación	0.1739164	0.1278985	0.1416394	0.1638653	0.1068632
Coeficiente de Variación	0.5797213	0.4111023	0.3186887	0.3597044	0.2466075

Observamos que los error rates son bastante parecidos, y no parece ser que el α sea determinante a la hora de estimar parámetros, observamos de la tabla anterior que los error rates son bien variados.

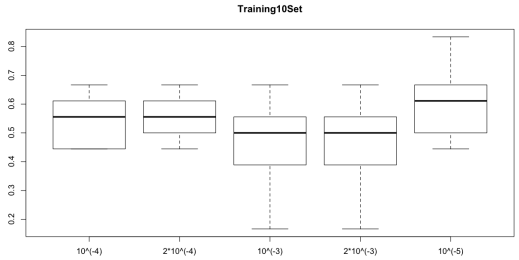

Figura 46: Boxplot del TrainingSet 10

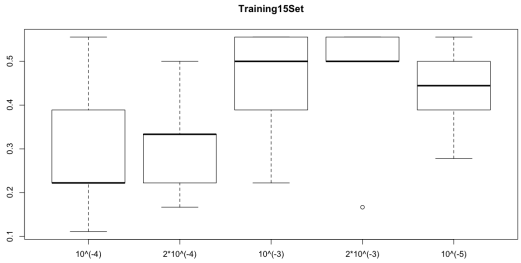

Figura 47: Boxplot del TrainingSet 15

El criterio que utilizaremos para elegir el ganador será los que tengan el menor error rate. Si 2 tienen el mismo error rate, considerare a ambos como ganadores. En el TrainingSet 10 empataron $2x10^{-3}$ y en el TrainingSet 15 gano 10^{-4}.

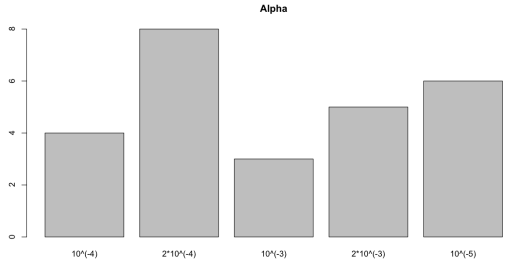

Figura 48: Diagrama de Barra de los distintos alpha

Training3Set	10(-4)	2*10(-4)	10(-3)	2*10(-3)	10(-5)
Media	0.1416394	0.1416394	0.1416394	0.1416394	0.1416394
Desviación	0.1666667	0.1666667	0.1666667	0.1666667	0.1666667
Coeficiente de Variación	0.8498366	0.8498366	0.8498366	0.8498366	0.8498366

Training6Set	10(-4)	2*10(-4)	10(-3)	2*10(-3)	10(-5)
Media	0.1278985	0.1278985	0.1278985	0.1278985	0.1278985
Desviación	0.1888889	0.1888889	0.1888889	0.1888889	0.1888889
Coeficiente de Variación	0.6771097	0.6771097	0.6771097	0.6771097	0.6771097

Ahora veremos que sucede con los datos estandarizados.

Observamos que el α cuando los datos estandarizados tiene una baja influencia en la calidad de la predicción, si bien el error rate es mucho menor que en la parte anterior, al hacer variar el α no influyo en el resultado final.

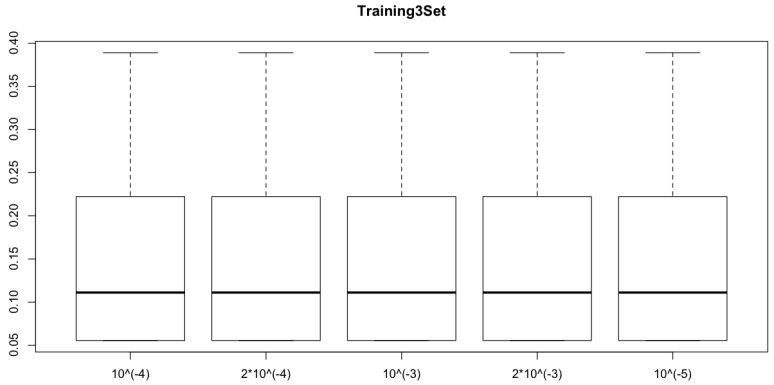

Figura 49: Boxplot del TrainingSet 3

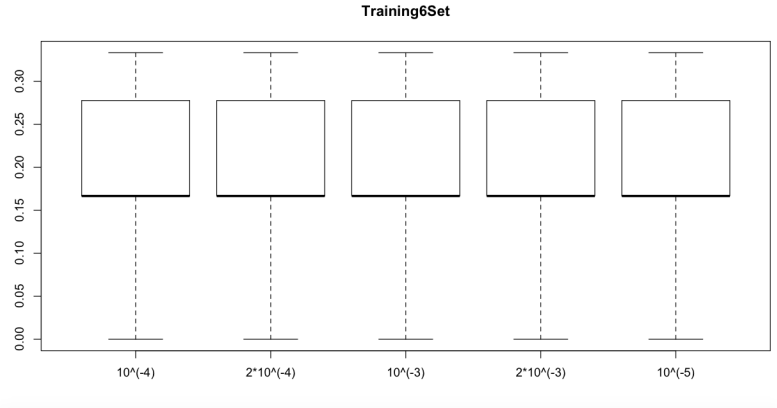

Figura 50: Boxplot del TrainingSet 6

En esta sección no podemos elegir un α ganador debido a que este fenómeno, donde los error rate son iguales, ocurrió para los 20 training Set.

Training3Set	10(-4)	2*10(-4)	10(-3)	2*10(-3)	10(-5)
Media	0.3353457	0.3353457	0.3353457	0.3353457	0.3353457
Desviación	0.1266862	0.1266862	0.1266862	0.1266862	0.1266862
Coeficiente de Variación	0.3353457	0.3353457	0.3353457	0.3353457	0.3353457

Training3Set	10(-4)	2*10(-4)	10(-3)	2*10(-3)	10(-5)
Media	0.3777778	0.3777778	0.3777778	0.3777778	0.3777778
Desviación	0.1490712	0.1490712	0.1490712	0.1490712	0.1490712
Coeficiente de Variación	0.3946002	0.3946002	0.3946002	0.3946002	0.3946002

Ahora nos queda hacer lo mismo con los datos normalizados, cabe recalcar que la normalización la haremos de -1 a 1, por la naturaleza de la función logística.

Nos damos cuenta que pasa el mismo fenómeno que cuando estandarizamos los datos, pero con un error rate mayor.

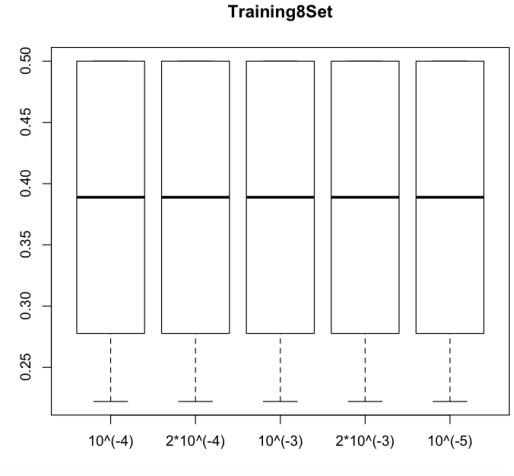

Figura 51: Boxplot del TrainingSet 3

Figura 52: Boxplot del TrainingSet 6

Al igual que los datos estandarizados se repitió el mismo fenómeno, en el que las medias de los error rate son iguales a lo largo de los 5 α, por lo que no podemos elegir un α ganador. Finalmente elegiremos el como ganador a $2x10^{-4}$ ya que tuvo la mayor cantidad de error rate bajos, en la sección de datos sin intervenir.

2.2. Newton-Raphson

En regresión logística también podemos usar Newton-Raphson, ya que la regla de actualización de este algoritmo nos pedía el gradiente de la función de costo, y el Hessiano, basta con encontrar estas 2.

El Gradiente de la función de costo es:

$$\nabla_\beta \ell(\beta) = X^T(Y - f_\beta(X))$$

Y el Hessiano es:

$$\text{H} = -X^T W X$$

Donde W es una matriz diagonal, donde cada elemento de su diagonal corresponde ser

$$W_{ii} = f_\beta(x_i)(1 - f_\beta(x_i)$$

Ahora si juntamos todo esto, la regla de actualización de Newton-Raphson, para regresión logística nos queda

$$\beta^{p+1} = \beta^p + (X^T W X)^{-1} X^T (Y - f_\beta(X))$$

Algo muy importante a considerar, es que debe existir la inversa del Hessiano

Ahora usando la versión de Newton-Raphson para regresión logística, correremos el five-fold, no con un fin de estudiar algún parámetro, si no para ver como se comporta este algoritmo en los 3 escenarios en los que hemos estado corriendo todos los algoritmos. Comenzaremos con los datos sin ninguna intervención

Newton Raphson	TrainingSet1	TrainingSet2	TrainingSet3	TrainingSet4	TrainingSet5
Media	0.07777778	0.08888889	0.1	0.07777778	0.06334308
Desviación	0.03042903	0.08425417	0.09128709	0.09296223	0.07777778
Coeficiente de Variación	0.3912304	0.9478594	0.9128709	1.195229	0.814411

Figura 53: Boxplot del TrainingSet2

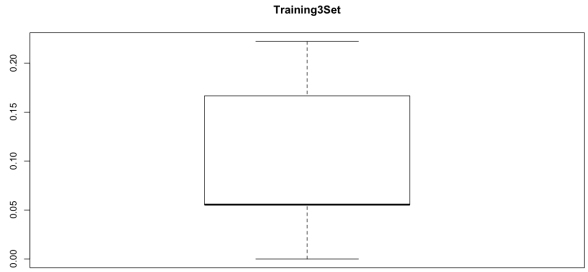

Figura 54: Boxplot del TrainingSet3

Ahora veremos como se comporta en presencia de datos estandarizados.

Newton Raphson	TrainingSet1	TrainingSet2	TrainingSet3	TrainingSet4	TrainingSet5
Media	0.2	0.2111111	0.2111111	0.2111111	0.2111111
Desviación	0.09296223	0.1204416	0.1730268	0.08240221	0.5705127
Coeficiente de Variación	0.4648111	0.5705127	0.8196006	0.3903262	0.09128709

Observamos que existió un aumento en el error rate comparado con el error rate obtenido en la sección de datos sin recibir algún tipo de intervención.

Figura 55: Boxplot del TrainingSet1

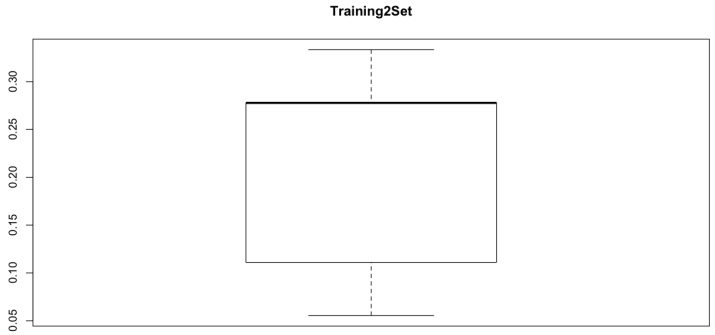

Figura 56: Boxplot del trainingSet 2

Newton Raphson	TrainingSet1	TrainingSet2	TrainingSet3	TrainingSet4	TrainingSet5
Media	0.2	0.1777778	0.2111111	0.1555556	0.2333333
Desviación	0.06334308	0.07243558	0.1383322	0.04648111	0.1266862
Coeficiente de Variación	0.3167154	0.4074502	0.6552579	0.2988072	0.5429407

Finalmente, veremos como se comporta este algoritmo en presencia de datos normalizados entre -1 y 1

Observamos que el error rate es bien parecido al error rate de datos estandarizados, pero mayor al error rate de los datos sin alguna intervención.

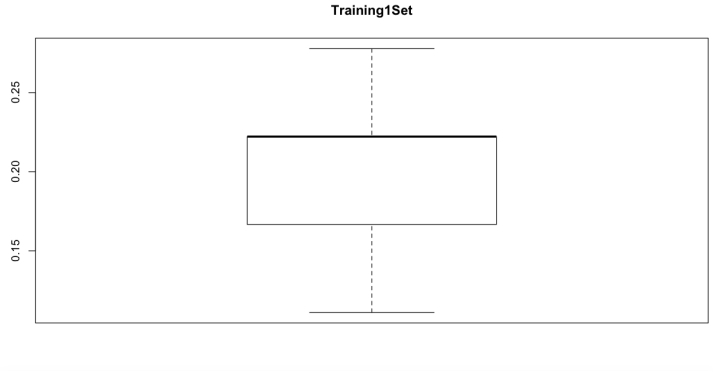

Figura 57: Boxplot del TrainingSet1

Figura 58: Boxplot del TrainingSet2

Pudimos observa que para el caso de gradiente ascendente online, si nosotros estandarizamos o normalizamos los datos, existe una mejora, ya que nuestro error rate disminuye considerablemente, en cambio, para Newton-Raphson al estandarizar o normalizar, lo que sucedió fue que empeoro, es decir aumento el error rate.

Finalmente, lo que haremos ahora, será estimar los parámetros con un training set, y luego intentar predecir le error rate de los targets del Training Set y de un Testing Set.

Training Set	Newton-Raphson	Online
Media	0.04722222	0.4138889
Desviación	0.01011628	0.05965844
Media (Standard)	0.1700000	0.1455556
Desviación (Standard)	0.02366652	0.02129651
Media (Normalizado)	0.2011111	0.4077778
Desviación (Normalizado)	0.01190170	0.02551612

Testing Set	Newton-Raphson	Online
Media	0.07166667	0.4683333
Desviación	0.03631409	0.12680279
Media (Standard)	0.1766667	0.1650000
Desviación (Standard)	0.3302241	0.05350302
Media (Normalizado)	0.2000000	0.4033333
Desviación (Normalizado)	0.06666667	0.07939832

En Newton-Raphson observamos el fenómeno contrario al que vimos en el five-fold, que el error crece al momento de estandariza o normalizar los datos, además observamos que el error rate entre las predicciones hechas en el Training y en el Testing son bastante cercanas, por lo que podríamos concluir que Newton-Raphson es un algoritmo bueno para cuando deseamos generalizar, por otro lado, el gradiente ascendente online, aumenta un poco su error al generalizar, y a diferencia de Newton, con datos estandarizados el error rate es mucho menor. Es claramente visible que el error rate de Newton es mucho menor que el error rate de gradiente ascendente Online.
Ahora mostraremos los boxplot comparativos entres los error rate del testing y training set.

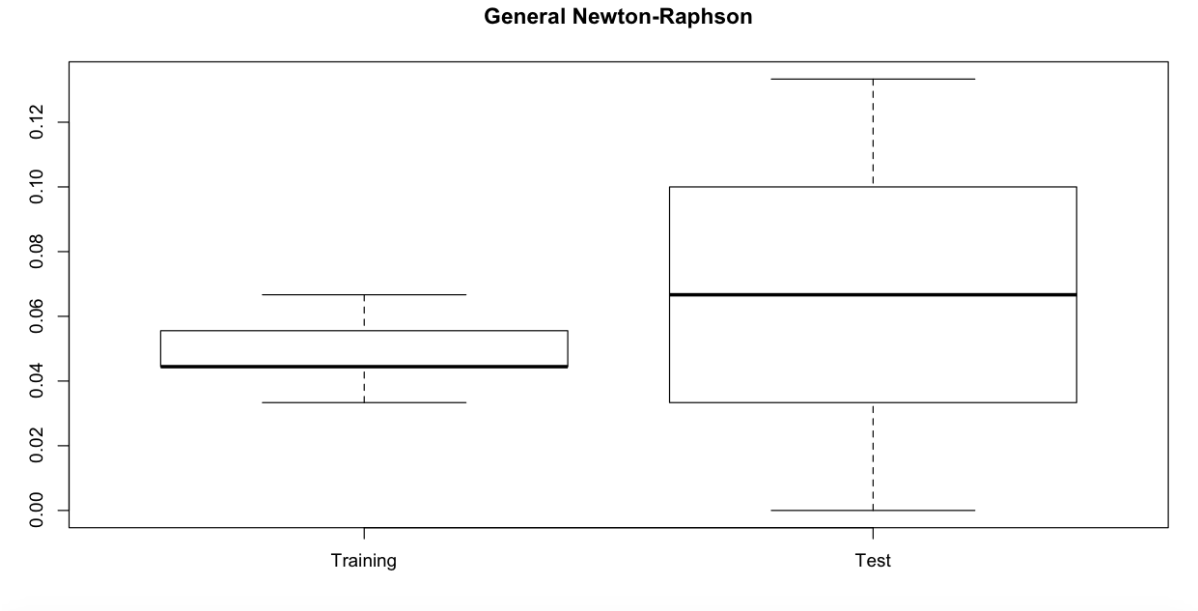

Figura 59: Boxplot Comparativo Newton-Raphson

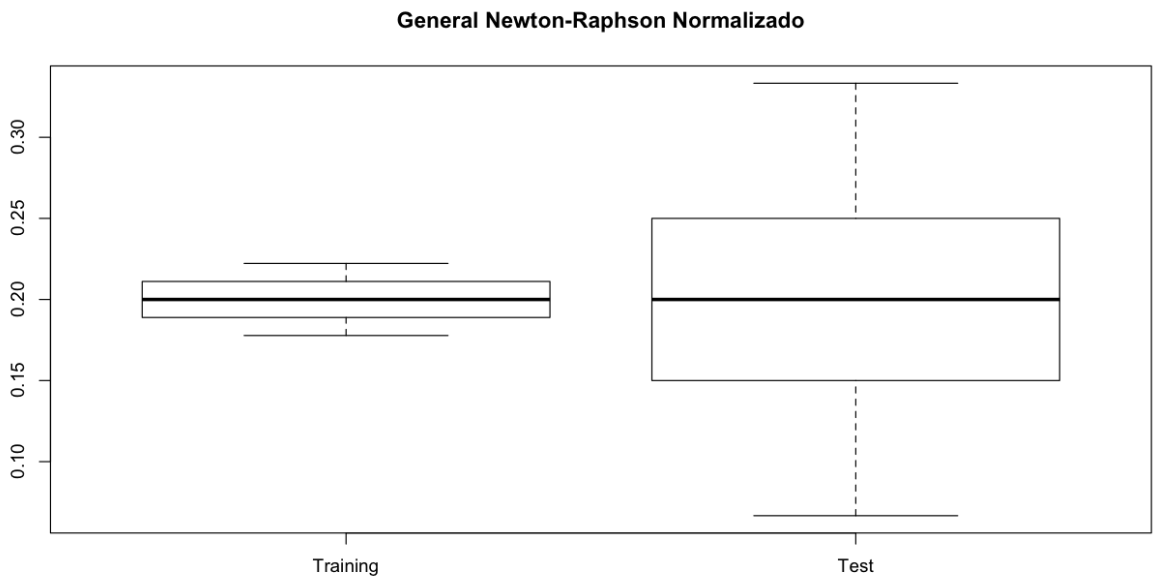

Figura 60: Boxplot Comparativo Newton-Raphson Normalizado

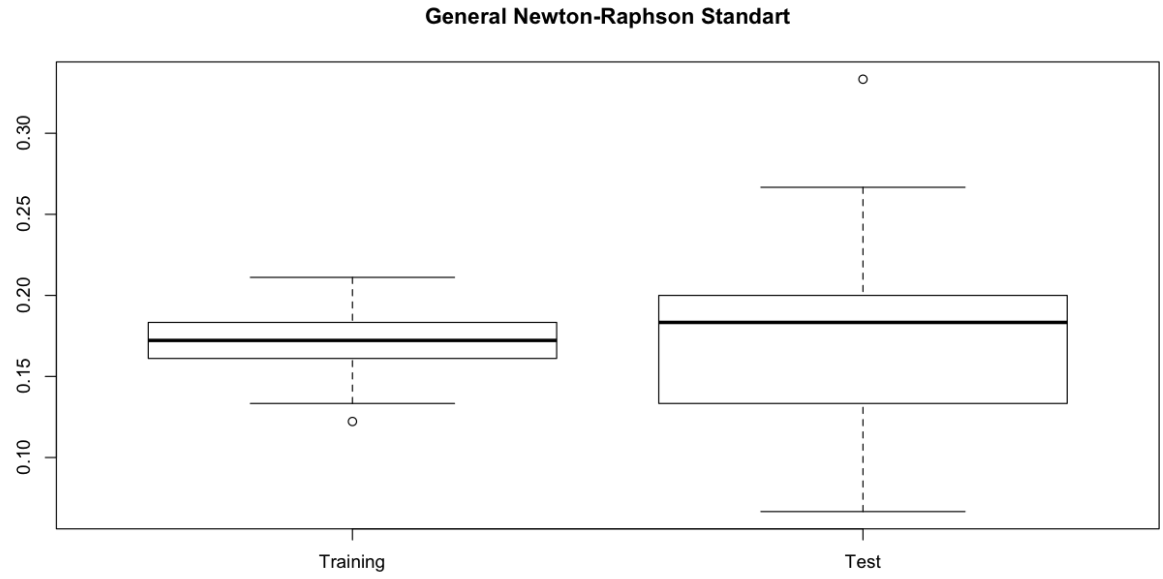

Figura 61: Boxplot Comparativo Newton-Raphson Estandar

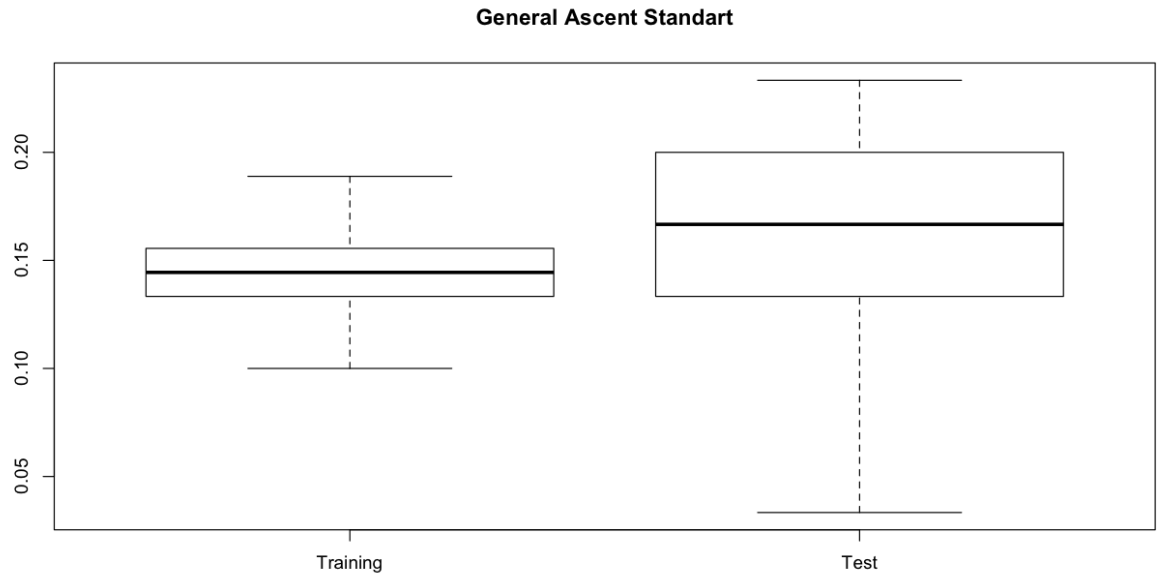

Figura 62: Boxplot Comparativo Gradiente Ascendente Online Standart

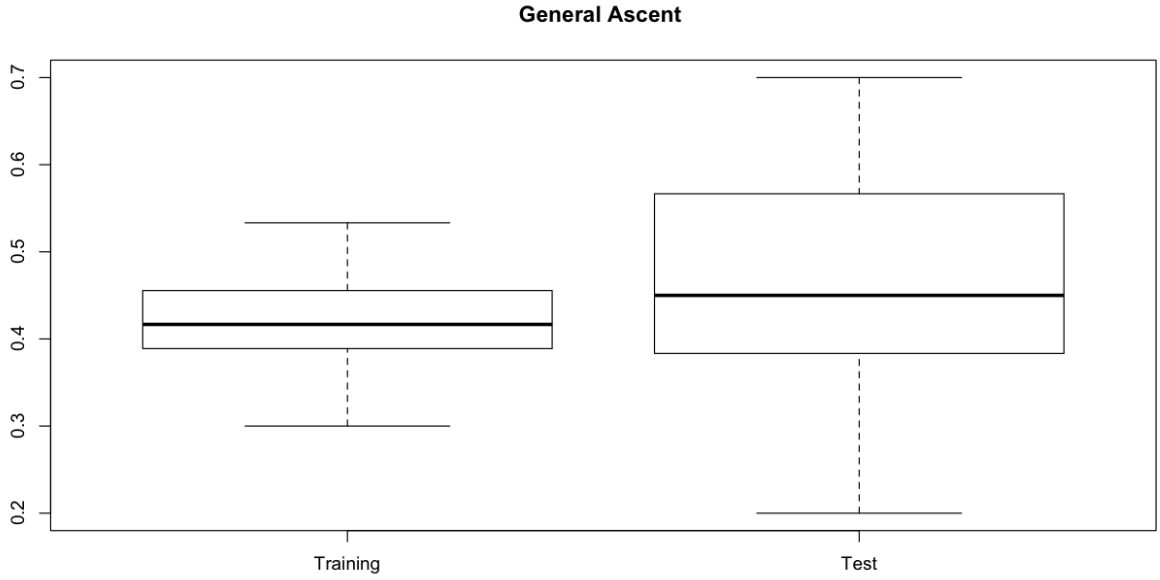

Figura 63: Boxplot Comparativo Gradiente Ascendente Online

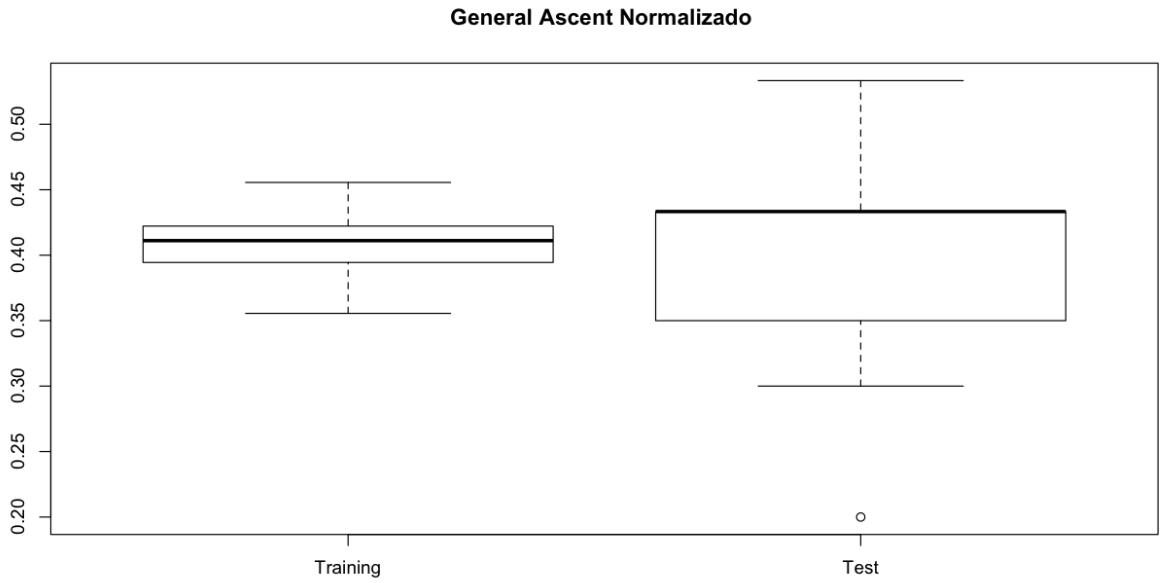

Figura 64: Boxplot Comparativo Gradiente Ascendente Online Normalizado

3. Conclusión

En el desarrollo de este trabajo, fuimos aprendiendo de primera mano sobre los algoritmos, al momento de implementarlos es donde uno los va conociendo de verdad, desde como programarlos hasta las cosas que se deben tener cuidado con ellos, como puede ser determinante que existe la inversa de la matriz diagonal como en el caso de Newton-Raphson, o el α de los algoritmos tipo gradiente descendente. Otra cosa importante es que se ejercito el tema de trabajo con matrices, y como estas pueden agilizar la programación.
Ahora respecto al trabajo, se observo de primera como afecta el trabajar con datos estandarizados, normalizados o sin algún tipo de manipulación, algunos de los algoritmos reaccionaban de mejor manera y otros simplemente no, también otra cosa determinante era el tipo de problema que estábamos abordando.