UNIVERSIDAD SAN FRANCISCO DE QUITO USFQ

Colegio de Ciencias e Ingenierías

Aplicación de la Constante de Lebesgue al Aprendizaje de Máquinas

Proyecto de investigación

Axel Leonardo Oviedo Collahuazo

Licenciatura en Matemáticas

Trabajo de titulación presentado como requisito
para la obtención del título de Licenciado en Matemática

Quito, 15 de diciembre de 2015

UNIVERSIDAD SAN FRANCISCO DE QUITO USFQ

COLEGIO DE CIENCIAS E INGENIERÍAS

HOJA DE CALIFICACIÓN

DE TRABAJO DE TITULACIÓN

Aplicación de la Constante de Lebesgue al Aprendizaje de Máquinas

Axel Leonardo Oviedo Collahuazo

Calificación:

Nombre del profesor, Título académico: Luca Guzzardi , Ph.D.

Firma del profesor: _____

Quito, 15 de diciembre de 2015

Derechos de Autor

Por medio del presente documento certifico que he leído todas las Políticas y Manuales de la Universidad San Francisco de Quito USFQ, incluyendo la Política de Propiedad Intelectual USFQ, y estoy de acuerdo con su contenido, por lo que los derechos de propiedad intelectual del presente trabajo quedan sujetos a lo dispuesto en esas Políticas.

Asimismo, autorizo a la USFQ para que realice la digitalización y publicación de este trabajo en el repositorio virtual, de conformidad a lo dispuesto en el Art. 144 de la Ley Orgánica de Educación Superior.

Firma del estudiante: _____

Nombres y apellidos: Axel Leonardo Oviedo Collahuazo

Código: 00108104

Cédula de Identidad: 171854317-4

Lugar y fecha: Quito, diciembre de 2015

Resumen

En aprendizaje de máquinas se distingue entre dos formas de errores. El error E_{in}, que es el error que se minimiza para buscar la función objetivo, y el error E_{out} que se requiere sea bajo. En literatura, se encuentran muchas investigaciones que relacionan los dos errores. Estos dependen de la complejidad de la función objetivo, la cantidad de ruido y la cantidad de puntos. En cambio, no se encuentran relaciones entre los errores y la distribución de puntos. Este es un primer tentativo de trazar dicha relación.

Palabras clave: sobreajuste, interpolación de Lagrange, constante de Lebesgue, nodos de Chebyshev, aprendizaje de máquinas, función de regresión.

Abstract

In Machine Learning we distinguish between two types of error. The error E_{in}, which is the error that is minimized to look for the target function, and the error E_{out} that is required to be low. In literature, we find many investigations that relate these two errors. These depend on the complexity of the target function, the amount of noise and the amount points. However, there are not relations between the errors and the points distribution. This is a first attempt to trace this relation.

Key words: overfit, Lagrange Interpolation, Lebesgue constant, Chebyshev nodes, Machine Learning, regression function.

Índice de contenidos

Índice de Figuras

1. INTRODUCCIÓN

En el aprendizaje de máquinas, se tiene como objetivo encontrar una función ideal para predecir eventos en base a un conjunto de datos en los cuales conocemos la respuesta de los eventos. De esta manera, vamos a tener dos tipos de errores: E_{in} y E_{out}.

El error E_{in} es el error que existe entre el valor dado por la función encontrada y los datos usados. Así por ejemplo, supongamos que se tienen datos $V = \{(x_0,\, y_0), (x_1,\, y_1), \dots, (x_{k-1},\, y_{k-1}), (x_k,\, y_k)\}$ y que la función encontrada (en base a estos datos) por un método de aprendizaje es g, entonces se puede calcular $E_{in}\,(g) = \sum_{i=0}^{k} \frac{(g(x_i)-y_i)}{k}^2$.

Por otro lado, el error E_{out} es aquel que nos dice que tanto se desvía esta función encontrada para predicciones de eventos futuros. Una forma de aproximarnos a este error E_{out} es mediante el error de datos no usados en el método de aprendizaje. De esta manera siguiendo el ejemplo anterior, si tenemos un conjunto de datos $W = \{(x_{w_0},\, y_{w_0}), (x_{w_1},\, y_{w_1}), \dots, (x_{w_N},\, y_{w_N})\}$ tal que $V \bigcap W = \emptyset$, entonces una aproximación a este error puede ser $E_{out}\,(g) = \sum_{i=0}^{N} \frac{\left(g(x_{W_i})-y_{W_i}\right)^2}{N}$.

En el aprendizaje de máquinas, se tienen muchas herramientas para acercarse a una función que nos ayude a predecir (o se aproxime a) datos y eventos que no hayamos visto todavía. Sin embargo, estas herramientas son más útiles si se sabe dónde se debe buscar, es decir seleccionar oportunamente el conjunto de funciones (espacio de las hipótesis) donde buscar la función deseada. De esta manera, al usar un método de aprendizaje, necesitamos un conjunto de funciones de hipótesis entre las cuales buscaremos una función adecuada en base al método usado y a nuestros datos. Así por ejemplo, al usar el método de mínimos cuadrados podemos hacer una regresión lineal en base al conjunto de datos, si nuestro conjunto de funciones hipótesis son lineas. De igual forma, al aumentar nuestro conjunto de funciones de hipótesis a funciones cuadráticas, podemos modificar el método para tener una regresión cuadrática.

Uno de los problemas más importantes a tomar en cuenta es el sobreajuste,

el cual está fuertemente ligado al conjunto de funciones hipótesis, y en el cual buscaremos a una función apropiada.

1.1. Sobreajuste

El sobreajuste se da cuando ajustamos nuestra función a los datos de entrada más de lo que se puede garantizar, lo cual pasa cuando el espacio de funciones es demasiado "grande". Como resultado de esta acción puede ocurrir que se reduce el error en nuestros datos de entrada E_{in}, pero aumenta el error de datos posteriores (o datos que no fueron usados para el aprendizaje) E_{out}.

En [1] nos muestran este fenómeno. Imaginemos que se corre dos veces el algoritmo de mínimos cuadrados, para polinomios de grado 10 y para grado 2, sobre un mismo conjunto de datos. Sean g_{10} y g_2 sus resultados respectivos, se considera como medida del sobreajuste a:

$$E_{out}(g_{10}) - E_{out}(g_2)$$

Dado que el objetivo del aprendizaje de máquinas es aproximarse a una predicción de eventos posteriores, diferentes a los datos usados en el método de aprendizaje, el error E_{out} es un elemento importante que se debe tomar en cuenta. En nuestro caso se busca la manera de reducirlo.

De esta forma, los siguientes gráficos muestran como cambia el sobreajuste cuando se varía el ruido de los datos, la complejidad de la función buscada y el número de datos; y por lo tanto, la diferencia del E_{out} para la aproximación a un polinomio de segundo grado y el E_{out} para la aproximación a un polinomio de grado diez.

Aquí f es nuestra función objetivo, σ^2 es el nivel de ruido, N es el número de datos de entrada y Q_f es el grado de la función objetivo (véase [1]).

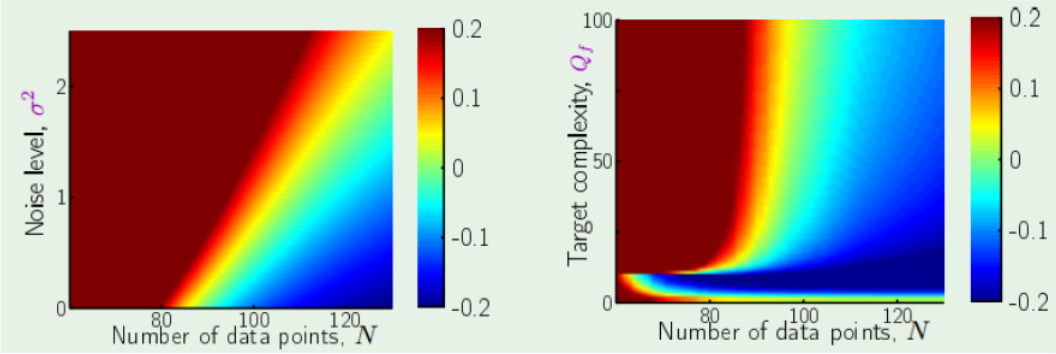

Figura 1: En la figura izquierda se muestra el sobreajuste en base al número de datos usados y la cantidad de ruido. En la figura derecha, se muestra el sobreajuste en base a la complejidad Q_f de la función buscada y el número de datos usados para el aprendizaje.

A medida que se intensifica el color azul la gráfica nos muestra que:

$$E_{out}(g_{10}) < E_{out}(g_2)$$

mientras que cuando se intensifica el color rojo tenemos que:

$$E_{out}(g_{10}) > E_{out}(g_2)$$

Por esto, en base al número de datos que se posea, al ruido que exista y a la complejidad de la función (que se puede proponer a priori, en base a información teórica) se puede ver que en las zonas azules será más apropiado usar un conjunto de funciones hipótesis más "grande" (en el caso del ejemplo son polinomios de grado 10). Mientras que para las zonas rojas es mejor tener al conjunto de funciones hipótesis más "pequeño" (igual al conjunto de polinomios de segundo grado en el ejemplo).

Como se mencionó anteriormente, el sobreajuste se da cuando se ajusta nuestra función a los datos más de lo que se puede garantizar. Al aumentar (en el ejemplo presentado) el grado de los polinomios del conjunto de funciones hipótesis, damos más libertad al método de ajustarse a nuestros datos; lo cual va a reducir nuestro E_{in} pero aumentar nuestro E_{out}. Es decir, hay que ser cuidadosos al elejir el conjunto de funciones hipótesis, ya que si es muy "grande" nos puede producir el problema del sobreajuste.

De esta manera, tenemos un marco referencial en base al sobreajuste que nos

ayuda a elegir un conjunto de funciones hipótesis apropiado. Este se basa en el número de datos usados, su ruido y la complejidad de la función buscada. Sin embargo, la intuición nos dice que el conjunto de datos (o una parte de él) puede tener una característica que nos daría un mejor resultado al modificar un poco el método usado. La característica a tomar en cuenta en el presente trabajo es la distribución de los datos.

Por lo tanto, buscaremos cómo afecta la distribución de los datos al error (E_{out}) de la función encontrada. Para esto abordaremos el problema usando los polinomios de Lagrange.

1.2. Interpolación de Lagrange

La interpolación puede ser vista como un problema de aprendizaje con condiciones fuertes (ya que fijan los datos de entrada para los cuales el error es cero). Por lo que es útil para abordar nuestro problema de una manera más sencilla.

El problema de la interpolación se lo puede plantear de la siguiente manera:

Buscamos una función f (en un espacio dado) tal que $f(x_i) = y_i$, dado un conjunto de puntos (x_0, y_0), (x_1, y_1), ... , (x_i, y_i), ... , (x_{k-1}, y_{k-1}), (x_k, y_k) donde si $x_j = x_h$ entonces $j = h$. La razón por la cual no se pueden tener datos $x_j = x_h$ con $y_j \neq y_h$ es porque no existe una función con puntos así, ya que para un valor de x tendría más de un valor de y.

El problema de la interpolación es ampliamente estudiado en el análisis numérico y del cual podemos usar muchos de sus resultados. Un resultado clásico es la interpolación de Lagrange, la cual cumple con la condición $f(x_i) = y_i$, con f en un espacio de polinomios.

Los polinomios de Lagrange tienen la propiedad de que dado un conjunto de puntos distintos entre si (x_i, y_i), son los polinomios de menor grado en los cuales para cada x_i corresponde un valor de y_i.

Estos polinomios se los define como la siguiente combinación lineal:

$$L(x) = \sum_{j=0}^{k} y_j . l_j(x)$$

donde las bases polinómicas de Lagrange (véase por ejemplo [2]) $l_j(x)$ son las siguientes:

$$l_j(x) = \prod_{\substack{0 \leq m \leq k \\ m \neq j}} \frac{(x - x_m)}{(x_j - x_m)}$$

En la figura 2, podemos ver como se ajusta el polinomio de Lagrange a un polinomio de grado 12 al aumentar el número de puntos usados.

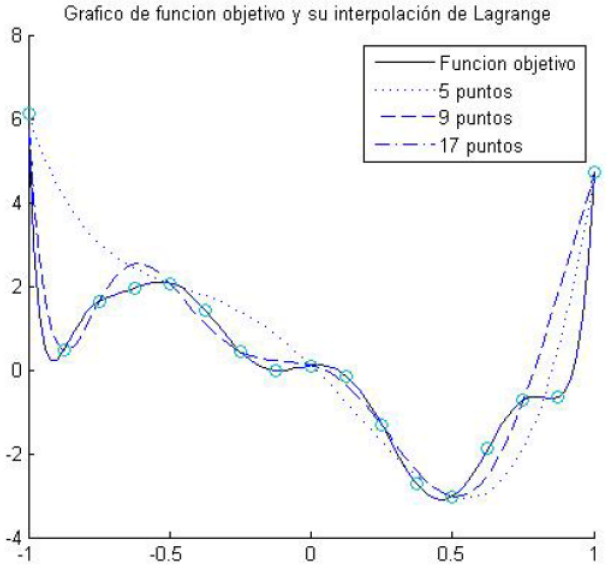

Figura 2: Se muestran tres interpolaciones de Lagrange utilizando 5, 9 y 17 puntos para una función de grado 12. Es claro que al utilizar 17 puntos el polinomio de Lagrange es idéntico a la función buscada.

Por otro lado, los polinomios de Lagrange son susceptibles al fenómeno de Runge en el cual al aumentar el grado del polinomio, utilizando puntos equidistantes, el polinomio oscila en los extremos provocando un error de aproximación a la función objetivo.

En el gráfico, se observa que el polinomio de Lagrange oscila en los extremos evidentemente para $n = 17$.

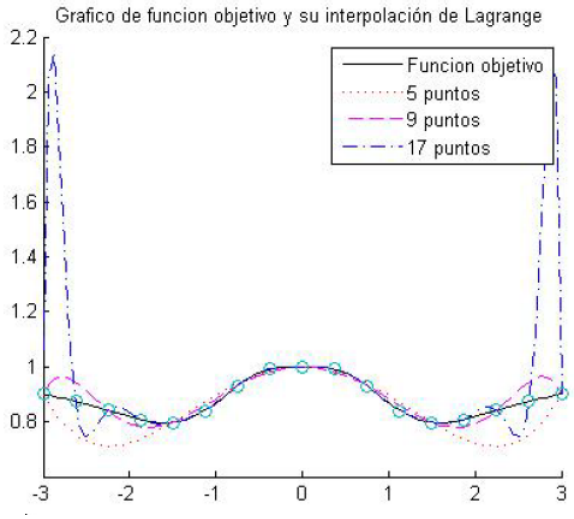

Figura 3: Se grafica los polinomios de Lagrange, con 5, 9 y 17 puntos de interpolación equidistantes, para aproximarse a la función $f(x) = 1 + 1/e^{x^2} - \frac{1}{x^2+1}$. Se puede ver que al utilizar más puntos tenemos oscilaciones en los extremos del intervalo.

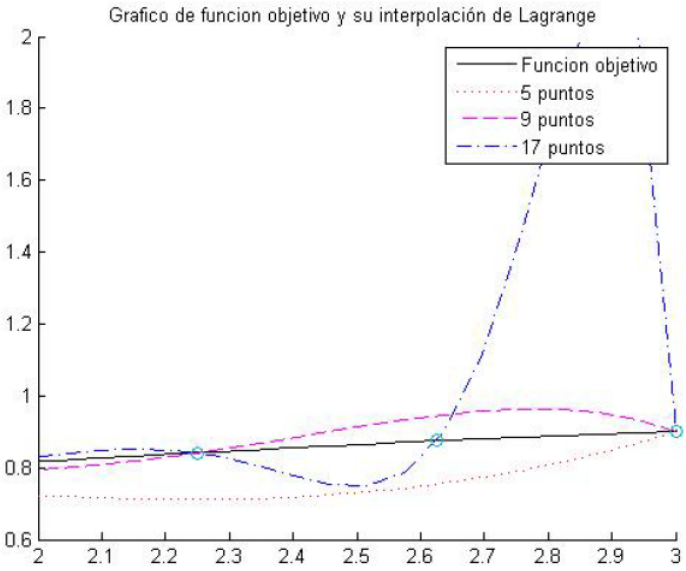

Figura 4: Al ampliar la figura anterior en el intervalo [2,3], podemos ver como aumenta la amplitud de las oscilaciones al aumentar el número de puntos.

Para poner este fenómeno en el contexto del aprendizaje de máquinas, se presenta el siguiente caso. Imaginemos que la función buscada es una función constante, pero los datos presentan un pequeño ruido aleatorio normal. Si estos puntos se encuentran distribuidos equidistantemente podemos ver como los polinomios de Lagrange oscilan en los extremos, pese a que los datos provienen de una función constante.

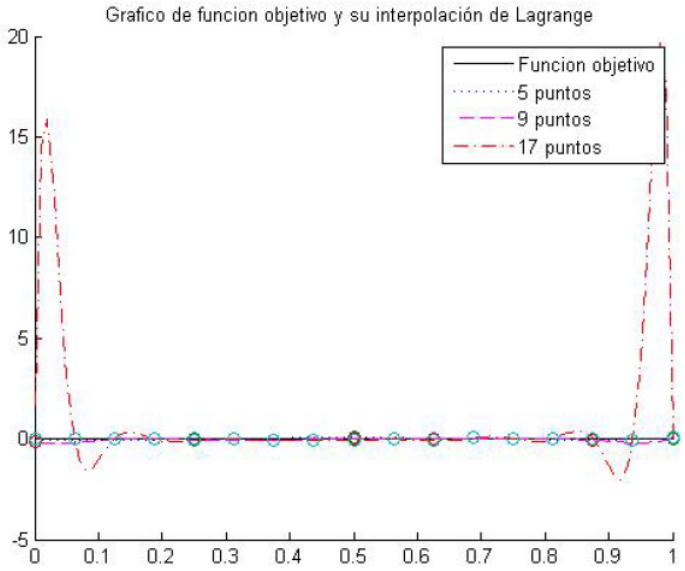

Figura 5: La función objetivo es $f(x) = 0$. Los puntos son equidistantes y presentan un pequeño ruido (aleatorio normal de media cero y desviación estándar 0.05) . Se aprecia como nos produce el fenómeno de Runge.

Este fenómeno se puede evitar o reducir al utilizar una mejor (en el contexto del fenómeno de Runge) distribución de puntos, la cual estará ligada a una menor velocidad de crecimiento de la constante de Lebesgue.

1.3. Constante de Lebesgue

Consideremos para cada $n \in \mathbb{N}$ un set de n nodos $x_{k,n}$ para $k = 1, 2, ..., n$ en el intervalo de $[-1, 1]$ de tal forma que

$$-1 \leq x_{n,n} < x_{n-1,n} < ... < x_{2,n} < x_{1,n} \leq 1$$

y ahora, construimos la matriz triangular infinita X:

$$X = \{x_{k,n} : \ k = 1, 2, ..., n; \ n \in \mathbb{N}\}$$

Ahora, dado $f \in C[-1, 1]$, podemos construir el polinomio de Lagrange de grado $n - 1$ para la función f, en base a los nodos dados en X. De esta manera, el polinomio de Lagrange puede ser modificado y ser escrito en base a n, X y f, de la siguiente manera:

$$L_{n-1}(X,\ f)(x) = \sum_{i=1}^{n} f(x_{i,n}) \cdot l_{i,n}(X,\ x)$$

donde $f(x_{i,n}) = y_{i,n}$, y

$$l_{i,n}(X,\ x) = \prod_{\substack{1 \leq m \leq n \\ m \neq i}} \frac{(x - x_{m,n})}{(x_{i,n} - x_{m,n})}$$

Es decir, $L_{n-1}(X,\ f)(x)$ es el polinomio de Lagrange de grado $n-1$ que utiliza n nodos (dados por X) y sus valores de f para aproximarse a esta función.

Ahora, sea $||f\ ||$ la norma uniforme del espacio de funciones continuas entre -1 y 1 :

$$||f\ || = \max_{-1 \leq x \leq 1} |f(x)|$$

y definiendo la función de Lebesgue como:

$$\lambda_n(X,\ x) = \max_{||f\ || \leq 1} |L_{n-1}(X,\ f)(x)| = \sum_{i=1}^{n} |l_{i,n}(X,\ x)|$$

tenemos que la constante de Lebesgue (véase [2]) será:

$$\Lambda_n(X) = \max_{||f\ || \leq 1} ||L_{n-1}(X,\ f)(x)|| = \max_{-1 \leq x \leq 1} \lambda_n(X,\ x)$$

Si bien la constante de Lebesgue tendrá un crecimiento a medida que se aumenta n, va a ser constante para un conjunto de nodos fijos y por tanto un n fijo.

Para X con nodos equidistantes tenemos que el crecimiento de la constante de Lebesgue, cuando $n \to \infty$, tiene la expansión asintótica (véase en [2]):

$$\Lambda_n(X) \backsim \frac{2^n}{e \cdot n \cdot \log(n)}$$

Esto nos da una constante de Lebesgue que crece muy rápidamente a medida

que n crece. Debido a este crecimiento ocurre el fenómeno de Runge, dado que en un sentido $\Lambda_n(X)$ mide el máximo del polinomio. Entonces, si el máximo del polinomio depende de $\Lambda_n(X)$, que en si depende de la distribución de los nodos usados en X, ¿Existen distribuciones que reducen $\Lambda_n(X)$ (y consecuentemente el fenómeno de Runge)?

A continuación veremos una distribución que reduce este crecimiento de la constante de Lebesgue que como nos da un crecimiento logarítmico (véase en [4]).

$$\Lambda_n(T) \backsim \frac{2}{\pi}\log(n+1)+1$$

Donde T es la matriz X con entradas distribuidas de la forma de nodos de Chebyshev que se presentan a continuación.

1.4. Nodos de Chebyshev

Los polinomios de Chebyshev son una familia de polinomios ortogonales que son definidos de forma recursiva. Los n-nodos de Chebyshev son las raices de un polinomio de Chebyshev de grado$n$.

Los polinomios de Chebyshev pueden ser de dos tipos, los cuales conforman familias diferentes. Las raíces para los polinomios del primer tipo se obtienen de la siguiente manera (véase en [2]):

$$x_k = \cos\left(\pi\frac{(2k-1)}{2n}\right)$$

para $k = 1, 2, ..., n$.

De manera similar, las raíces para los polinomios del segundo tipo son (véase en [5]):

$$x_k = \cos\left(\frac{k}{n+1}\pi\right)$$

para $k = 1, 2, ..., n$.

Ambas distribuciones nos dan un crecimiento logarítmico en la constante de Lebesgue, pero en el presente trabajo haremos uso de los nodos de Chebyshev

de segundo tipo.

A continuación se muestra como la interpolación de Lagrange con 9 puntos de entrada cambia drásticamente al cambiar la distribución de los puntos de equidistantes a nodos de Chebyshev del segundo tipo.

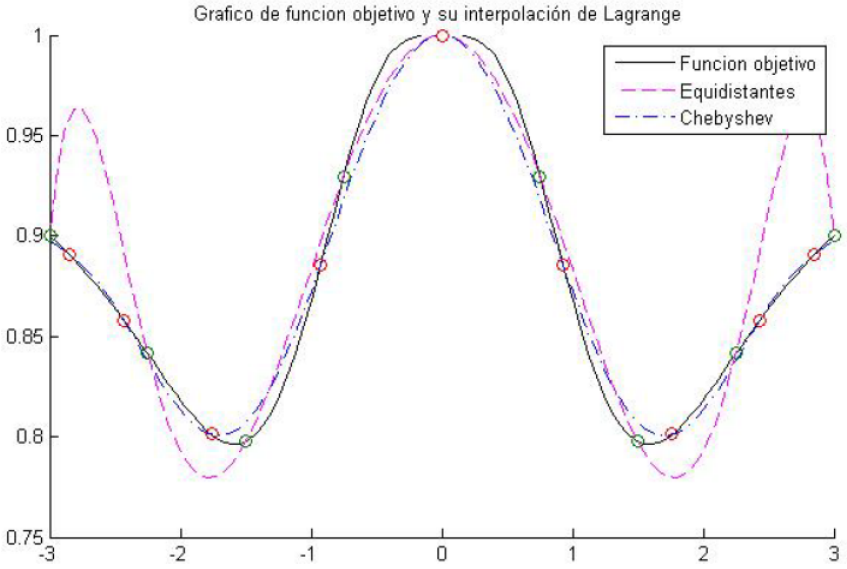

Figura 6: De la función $f(x) = 1 + 1/e^{x^2} - \frac{1}{x^2+1}$ se utilizan nueve puntos equidistantes y nueve nodos de Chebyshev. Se grafican sus respectivas interpolaciones de Lagrange. Al enfocarnos en el intervalo [2,3] del eje x de la figura, se nota como se reduce el fenómeno de Runge.

Como se muestra, el cambio es claro entre la distribución de puntos de Chebyshev y los nodos equidistantes. Sin embargo, la intuición nos dice que para distribuciones cercanas a la Chebyshev también tendremos una mejora respecto a la distribución equidistante de nuestros nodos.

2. DESARROLLO DEL TEMA

El incremento en la constante de Lebesgue (para polinomios de Lagrange) utilizando puntos distribuidos equidistantemente, es mucho más rápido que para puntos distribuidos de la forma de los nodos de Chebyshev. Además, los nodos de Chebyshev en la interpolación de Lagrange nos reduce el fenómeno de Runge, es decir las oscilaciones en los extremos.

En base a esto, se sabe que al tener una distribución en los datos de entrada de x, para la interpolación de Lagrange de una función f, igual a la de nodos de Chebyshev tendremos que nuestro E_{out} será menor, a si los datos en x se distribuirán equidistantemente. De igual manera, la intuición nos dice que para una distribución aproximada a la distribución de los nodos de Chebyshev, nuestro E_{out} será menor que para una distribución equidistante (o aproximada).

El error E_{out} es de suma importancia para el aprendizaje de máquinas. Por esta razón, este será el objeto de estudio el cual analizaremos como cambia al alejarnos de una distribución de Chebyshev y acercarnos a una distribución de puntos equidistantes.

Para establecer claramente este concepto de alejamiento y acercamiento de una distribución de puntos a otra, vamos a utilizar el concepto de homotopía (Apéndice A). De esta manera se propone lo siguiente:

$H_n : \{1, 2, ..., n\} \times [0, 1] \to [0, 1]$ definida de la siguiente forma:

$$H_n(i, t) = C_i^{(n)} + r(E_i^{(n)} - C_i^{(n)})$$

Donde $C_i^{(n)}$ es el i-ésimo (en orden ascendente) nodo de Chebyshev trasladado al intervalo [0,1] de un polinomio de Chebyshev de grado n, y $E_i^{(n)}$ es el i-ésimo nodo del intervalo [0,1] dividido en $n - 1$ partes iguales.

Utilizando esto tenemos una transformación continua de los nodos de Chebyshev a los puntos equidistantes en base a t.

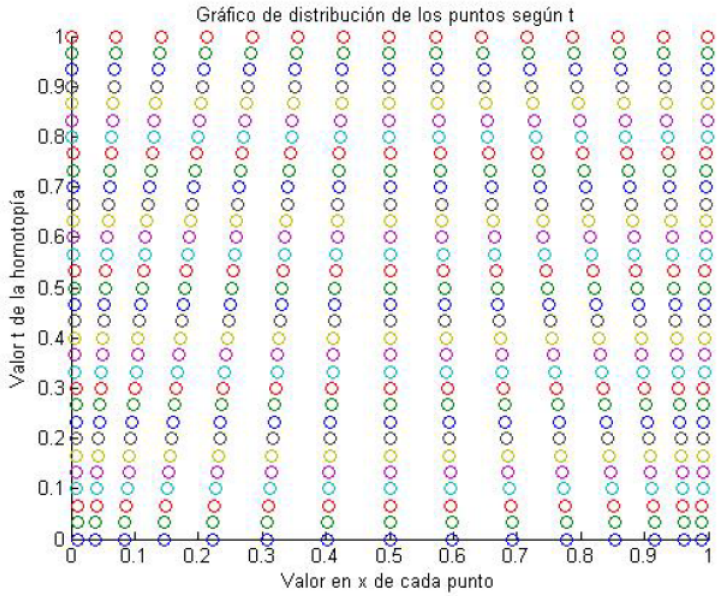

Figura 7: En el gráfico se muestra como están distribuidos 15 nodos al cambiar el valor de t. Se puede observar como los nodos cambian paulatinamente de la distribución de nodos de Chebyshev a nodos equidistantes.

Usando Matlab, programamos las funciones para obtener las diferentes distribuciones de puntos en base a H_n. De igual manera, programamos las funciones para obtener los coeficientes de el polinomio de Lagrange dado un set de datos. Además, se construye la función que nos retorna los coeficientes de una función objetivo dado el grado de complejidad Q. La función objetivo la establecemos como una combinación lineal (con pesos aleatorios) de los polinomios de Legendre hasta el Q-ésimo polinomio de Legendre (véase Anexo A). De igual manera, se programa la función que retorne el error dados los coeficientes de la función objetivo y los coeficientes del polinomio de interpolación de Lagrange. (véase Anexo B)

Como se puede ver, la función en Matlab e_out (Anexo B) tiene como entradas los coeficientes del polinomio (targetcoef) al que buscamos una aproximación y al cual nombraremos f, los nodos en x (xtrain) y el intervalo $[a, b]$. Fijando el intervalo en $[0, 1]$, tenemos que la función depende de f y de los nodos en x. Dado que la complejidad de f varía al aumentar el grado del polinomio Q, tenemos que e_out dependerá de Q. De igual manera, los nodos en x variarán en cantidad n y de acuerdo a su distribución en $[0, 1]$ dados por t (con

valores entre cero y uno) en la homotopía H_n. Es por esto que nuestro e_out estará determinado por tres variables: Q, $n \in \mathbb{N}$ y $t \in [0, 1]$. Ahora fijando la variable Q o la variable n en un valor constante, tenemos una función de dos variables que la podemos visualizar gráficamente utilizando una escala de color, donde el color representa el valor de e_out.

En cada uno de los gráficos que se muestran a continuación, se divide al eje x del gráfico en 500 partes (de 0 a 1) para observar como cambia el error descrito anteriormente según cambia la distribución (de los datos de entrada) dada por la homotopía. Para el siguiente gráfico se fija el grado Q de complejidad de f en 30:

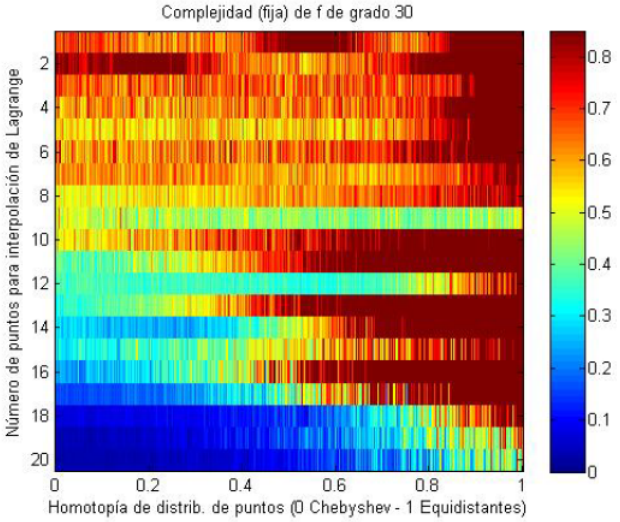

Figura 8: En el gráfico se muestran los datos al fijar la complejidad de f en $Q = 30$. Se varía n hasta veinte puntos (usados para la interpolarización). El color abarca un error del 0 al 0.85.

Ahora fijando la cantidad de puntos n en 10 y variando la complejidad de la función hasta $Q = 50$ obtenemos lo siguiente:

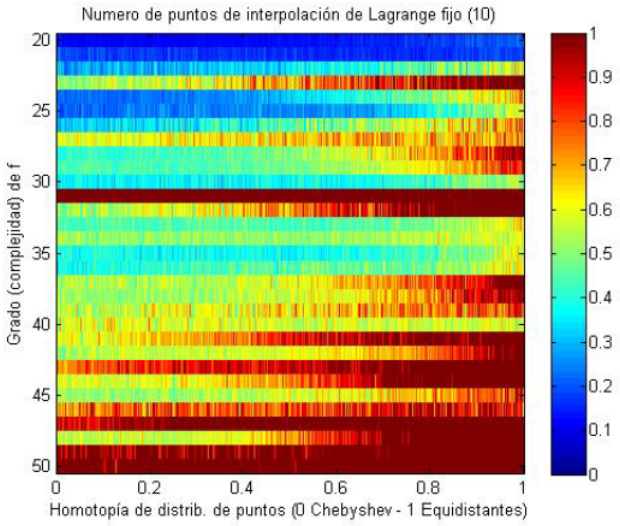

Figura 9: En el gráfico presente se fija el número de puntos usados en la interpolación de f en $n = 10$. Se varía Q de veinte hasta un grado de complejidad de cincuenta. El color abarca un error del 0 al 1.

Se puede observar en los dos gráficos que al seguir una línea horizontal la complejidad de la función buscada y el número de puntos usados para la interpolación están fijos, es decir solo varia t, en base a esto se comprueba como las distribuciones dadas por la homotopía en base a t cambian nuestro error.

En los dos gráficos se observa que nuestro error crece (relativamente en forma gradual) a medida que la distribución de los nodos se aproximan a una distribución equidistante.

3. CONCLUSIONES

De los gráficos obtenidos podemos concluir con varias cosas. En primer lugar, se evidencia que al tener distribuciones aproximadas a una distribución de Chebyshev, el error (E_{out}) se reduce en contraste a distribuciones aproximadas a la distribución equidistante. Se observa que teniendo una misma cantidad de datos y una misma función objetivo, el método de interpolación de Lagrange nos retorna diferente error (E_{out}) al variar la disposición de los nodos. Este error se incrementa a medida que se aleja de la distribución de nodos de Chebyshev y se acerca a una distribución equidistante (usando como medida de cercanía a t entre 0 y 1). Podemos ver que al tener una distribución similar a la de Chebyshev podemos tener una mejor aproximacón a funciones más complejas sin necesidad de tener más puntos.

Uno de los problemas al momento de realizar el actual proyecto se puede evidenciar en el siguiente gráfico, donde para una función de grado 30 se tiene un aumento drástico en el error después de 25 puntos. Se presume que el error se debe a una aproximación númerica dada por el programa Matlab. La forma de comprobar este problema sería elaborar el proyecto con programas de mayor presición.

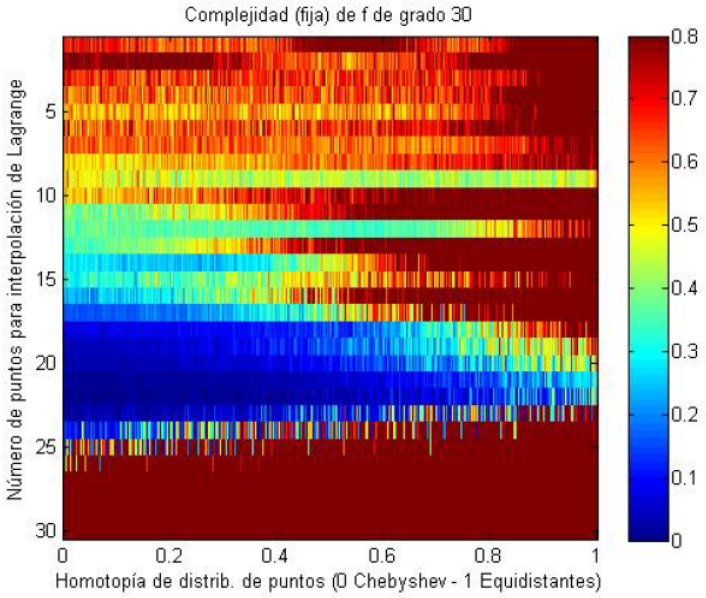

Figura 10: La función f se fija en una complejidad $Q = 30$ y se grafica hasta $n = 30$. Se puede ver que cerca de 25 nuestro error cambia drásticamente.

En las siguientes figuras se aprecia de una forma más marcada el cambio en el error al variar t de 0 a 1.

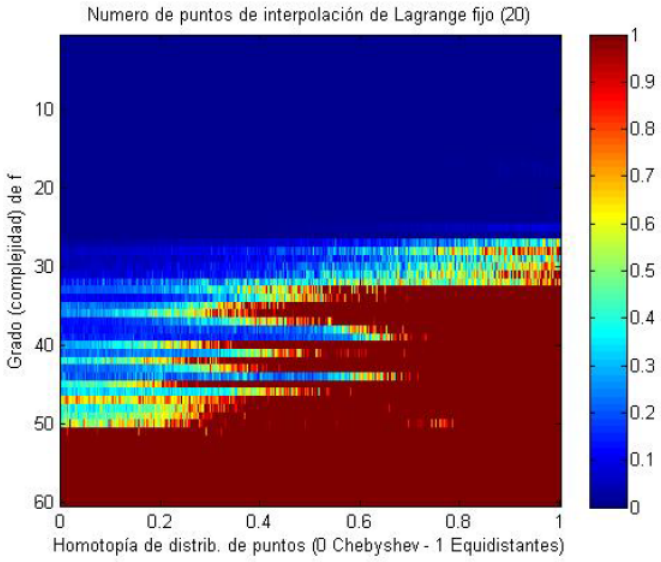

Figura 11: Se fija $n = 20$ y variamos Q de 1 a 60. El color abarca un error de 0 a 1.

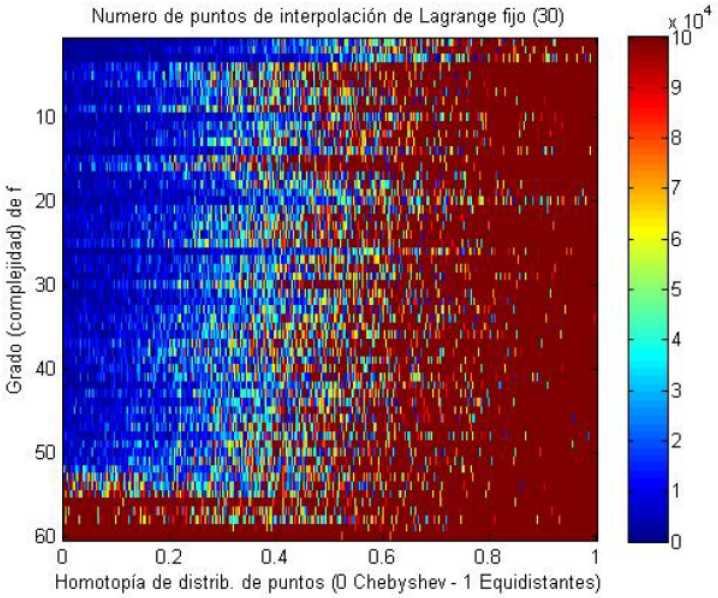

Figura 12: Se fija $n = 30$ y variamos Q de 1 a 60. El color abarca un error de 0 a 100000.

En la siguiente figura vemos que el error es extremadamente alto, razón por la cual se divide el gráfico en rojo y azul. Sin embargo, el hecho que se debe notar es que existe una inclinación para la frontera entre los dos colores. Esto claramente es debido al cambio en la distribución de los puntos de interpolación.

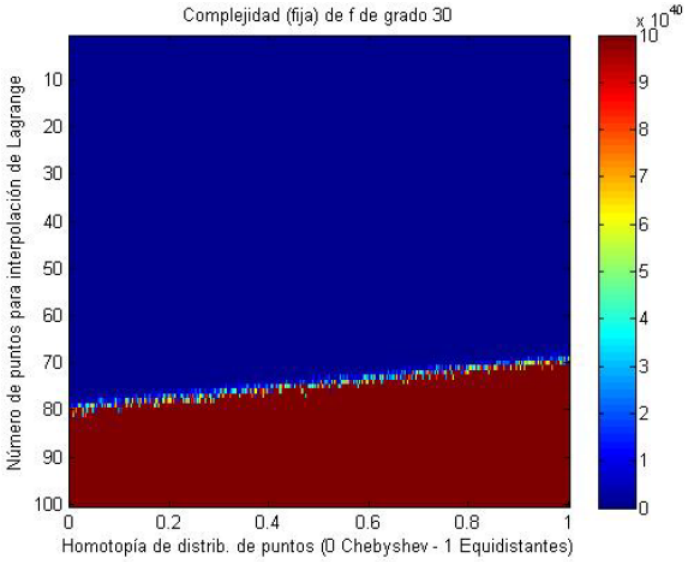

Figura 13: Se fija $Q = 30$ y variamos Q de 1 a 100. El color abarca un error de 0 a 10^{41}.

De esta manera, mediante el problema de interpolación podemos ver que se cumple la característica que buscábamos. Sin embargo, para tener una mejor aplicación al aprendizaje de máquinas (y como siguiente paso) es necesario

relacionar el método de mínimos cuadrados a la característica de la distribución de los datos, que fue estudiada en el presente trabajo mediante la interpolación . Este método nos permitirá tener una mayor generalización de las aplicaciones de las distribuciones de los datos en el problema de aprendizaje, dado que ya no estará restringido a la condición de interpolación $f(x_i) = y_i$.

Intuitivamente, una aplicación se podria dar al aplicar la interpolación de Lagrange posterior al método de los mínimos cuadrados, usando datos distribuidos de manera similar a la distribución de nodos de Chebyshev. Es decir, sea $g(x)$ la función resultante del algoritmo de los mínimos cuadrados para un conjunto de datos $(x_0,\, y_0)$, $(x_1,\, y_1)$, ... , $(x_i,\, y_i)$, ... , $(x_{k-1},\, y_{k-1})$, $(x_k,\, y_k)$. Buscamos un subconjunto C de estos datos (en x) que se aproxime a una distribución de nodos de Chebyshev. Y finalmente, interpolamos por Lagrange $L(C,\, g)(x) = \sum_{i=1}^{n} g(x_{C_i,}) \cdot l_i(x_{C_i})$. De esto, se debe buscar si el procedimiento propuesto mejora el resultado del algoritmo de los mínimos cuadrados.

Si se logra obtener una aplicación (o relación) del cambio en la distribución de los puntos en el método de mínimos cuadrados, esto da lugar a varias aplicaciones en el aprendizaje de máquinas. Una de estas puede ser la separación adecuada de los datos, en el sentido de usar los datos con distribución más aproximada a los de Chebyshev, entre los conjuntos de datos de aprendizaje, de prueba y validación.

Referencias

[1] ABU-MOSTAFA, Y., MAGDON-ICMAIL, M., LIN, H., *Learning from Data*, AMLBook, 2012

[2] BURDEN, R., FAIRES, J., *Numerical Analysis*, Thomson Brooks/Cole, Belmont, CA, 2005.

[2] SMITH, S., Lebesgue constants in polynomial interpolation, *Annales Mathematicae et informaticae,* **33**, (2006), 109–123 .

[4] TREFETHEN, L., *Approximation Theory and Aproximation Practice*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 2013.

[5] MASON, J., HANDSCOMB, D., *Chebyshev Polynomials*, Chapman and Hall/CRC, Boca Ratón, 2003.

[6] AYALA, R., DOMÍNGUEZ, E., QUINTERO, A., *Elementos de la Teoría de Homología Clásica*, Universidad de Sevilla, Sevilla, 2002.

[7] ARFKEN, G., WEBER, H., *Mathematical Methods for Physicists*, Elsevier Academic Press, San Diego, 2005.

ANEXO A: Homotopía y Polinomios de Legendre

Homotopía

Dos aplicaciones continuas $f, g : X \to Y$ se dicen homotópicas si existe otra aplicación continua $H : X \times [0,1] \to Y$ tal que (véase en [6])

$H(x,0) = f(x)$

$H(x,1) = g(x)$

Polinomios de Legendre

En ecuaciones diferenciales ordinarias tenemos que los polinomios de Legendre son las soluciones a las ecuaciones diferenciales de Legendre (véase en [7]):

$\frac{d}{dx}\left[(1-x^2)\frac{d}{dx}P_n(x)\right] + n(n+1)P_n(x) = 0$

donde $P_n(x)$ es el n-ésimo polinomio de Legendre.

Los polinomios de Legendre pueden ser obtenidos de manera recursiva mediante (véase en [7]):

$P_n(x) = (\frac{2n-1}{n})(x)P_{n-1}(x) - (\frac{n-1}{n})P_{n-2}(x)$

Debido a su incremento en complejidad y grado del polinomio, a sus múltiples aplicaciones en ramas de la Física y a su obtención de manera recursiva, estos polinomios son de utilidad para tomarlos como nuestra función buscada en la interpolación.

ANEXO B: Funciones MATLAB usadas

```
function [ y ] = valuepol(coef, a)
%La función calcula el valor de f(a), donde f es un polinomio dado por
los
%coeficientes de coef, y a es un número real.
y=0;
    for x=1:(length(coef)-1)
        y=y+power(a,length(coef)-x)*coef(x);
    end
    y=y+coef(length(coef));
end


function [ coef ] = basepolycoef(trainset,j)
%Función que calcula los coeficientes (descendentes en cuanto al grado)
de
%la base para j-ésimo dato para los polinomios de lagrange en base a los
%datos en x (trainset)
p=[];
    for x=1:length(trainset)
        if x~=j
            k=-1*trainset(x);
            p=vertcat(p,k);
        end
    end
coef=[1;p(1)];
m=trainset(j)+p(1);
    for x=2:length(p)
        m=m*(trainset(j)+p(x));
        d=coef(length(coef));
        k=1;
        for y=2:length(coef)
            c=coef(y)+p(x)*k;
            k=coef(y);
            coef(y)=c;
        end
        coef=vertcat(coef,d*p(x));
    end
    for x=1:length(coef)
        coef(x)=coef(x)*(1/m);
    end
end

function [ coeflagrange ] = polylagrange(xtrain, ytrain)
%Función que calcula los coeficientes del polinomio de lagrange
retornados
%en forma descendente (en cuanto al grado)
coef=[];
coeflagrange=[];
%Para un solo dato de entrada la función retorna una función constante
if length(xtrain)==1
    coeflagrange=ytrain;

else
    %Polinomio de lagrange para dos puntos o más
    for x=1:length(xtrain)
        coef=basepolycoef(xtrain,x);
        for y=1:length(coef)
```

```
            coef(y)=coef(y)*ytrain(x);
        end
        if x==1
            coeflagrange=coef;
        else
            for y=1:length(coef)
                coeflagrange(y)=coeflagrange(y)+coef(y);
            end
        end
    end
end
end


function [ A ] = polylegendre(n)
%La función presente nos retorna una matriz en la que cada fila son los
%coeficientes (del menor grado al mayor) de los polinomios de Legendre
%hasta el n-ésimo polinomio de Legendre
Lj=[0;1];
Lk=[-1/2;0;3/2];
Ln=[0;0;0];
A=[0,0,0;0,1,0;Lk'];
    %Polinomio de legendre n=1
    if n==1
        A=[0,0;0,1];
    %Polinomio de legendre n=2
    elseif n==2
        Ln=Lk;
    else
        %Algoritmo para calcular los polinomios de legendre de forma
        %recursiva
        for x=3:n
            Ln(1)=Lj(1)*(-1)*(x-1)/x;
            for y=2:length(Lj)
                Ln(y)=((2*x-1)*Lk(y-1)-(x-1)*Lj(y))/x;
            end
            Ln(length(Ln))=Lk(length(Lk)-1)*(2*x-1)/x;
            Ln=vertcat(Ln,Lk(length(Lk))*(2*x-1)/x);
            Lj=Lk;
            Lk=Ln;
            A=[A zeros(x,1)];
            A=vertcat(A,Ln');
        end
    end
end


function [ coef ] = targetfunction(n)
%La función nos retorna los coeficientes (del coeficiente de mayor grado
%al coeficiente constante) de un polinomio de grado n en base a una
%combinación lineal de los polinomios de Legendre hasta el n-ésimo
%polinomio de Legendre usando como pesos números aleatorios de una
%normal estándar
A=polylegendre(n);
p=normrnd(0,1,[n+1,1]);
coef=[];
coef=(A')*p;
co=coef;
    for x=1:n+1
        coef(x)=co(n+2-x);
```

```
    end
end


function [ p ] = puntosdistr(n,a,b,tipo)
%La función presente nos retorna un set de números en el intervalo [a,b]
p=[];
    if tipo==1
        for x=1:n
            k=(cos((2*x-1)*pi/(2*n))+1)*(b-a)/2+a;
            p=vertcat(p,k);
        end
    elseif tipo==2
        for x=1:n
            k=(cos((x)*pi/(n+1))+1)*(b-a)/2+a;
            p=vertcat(p,k);
        end
    elseif tipo==3
        p=vertcat(p,a);
        for x=2:(n)
            p=vertcat(p,((b-a)/(n-1)+p(x-1)));
        end
    else
        for x=1:n
            p=vertcat(p,(b-a)*rand+a);
        end
    end
p=sort(p);
end


function [ p ] = meddistrpuntos(r,a,b,n)
%La siguiente distribución nos retorna los n-puntos en el intervalo
[a,b],
%dados por la Homotopía de la distribución de los nodos de Chebyshev a la
%distribución equidistante en un t=r
p=[];
e=puntosdistr(n,a,b,3);
ch=puntosdistr(n,a,b,2);
    for x=1:n
        k=ch(x)+r*(e(x)-ch(x));
        p=vertcat(p,k);
    end
end


function [ e ] = e_out(targetcoef, xtrain, a, b)
%La función presente nos retorna el error de salida de la interpolación
de
%Lagrange dado el set de puntos xtrain y los coeficientes de la función
%buscada.
e=0;
ytrain=[];
%Calculo de ytrain en base al polinomio buscado.
    for x=1:length(xtrain)
        ytrain=vertcat(ytrain,valuepol(targetcoef, xtrain(x)));
    end
%Interpolación de Lagrange
g=polylagrange(xtrain, ytrain);
    %Cálculo del error para 200 puntos aleatorios en el intervalo (a,b)
```

```
    for x=1:200
        m=rand*(b-a)+a;
        k=valuepol(targetcoef,m)-valuepol(g,m);
        e=e+abs(k);
    end
    e=e/200;
end


function [ C ] = fixedgraph(n, m, tipo, Q, a, b)
%Función que nos calcula el error para diferentes valores de t (aquí
%llamada r)y para diferentes valores de complejidad de la función o del
%número de puntos usados (fijando el uno y variando el otro)
xtrain=[];
r=0;
f=[];
C=zeros(m-n+1,501);
%Para un determinado número Q de puntos fijos
if tipo==1
    for x=n:m
        f=targetfunction(x);
        for y=1:501
            xtrain=meddistrpuntos(r,0,1,Q);
            C(x,y)=e_out(f, xtrain, 0, 1);
            r=r+1/500;
        end
        r=0;
    end
%Gráfico con el error mostrado por el color entre a y b
imagesc([0,1],[n,m],C,[a,b])
xlabel('Homotopía de distrib. de puntos (0 Chebyshev - 1 Equidistantes)')
ylabel('Número de puntos para interpolación de Lagrange')
title('Numero de puntos de interpolación de Lagrange fijo ')
%Para una determinada complejidad Q de la función objetivo
else
    f=targetfunction(Q);
    for x=n:m
        for y=1:501
            xtrain=meddistrpuntos(r,0,1,x);
            C(x,y)=e_out(f, xtrain, 0, 1);
            r=r+1/500;
        end
        r=0;
    end
%Gráfico con el error mostrado por el color entre a y b
imagesc([0,1],[n,m],C,[a,b])
xlabel('Homotopía de distrib. de puntos (0 Chebyshev - 1 Equidistantes)')
title('Complejidad (fija) de f ')
ylabel('Grado (complejidad) de f')
end
end
```