Estimación de un modelo de aprendizaje con regresión logística utilizando métodos de optimizacion numérica

1 Introducción

Tomando como referencia el trabajo de investigación realizado por Colubri et al. (2019), se busca entrenar un modelo de regresión logística que permite pronosticar la supervivencia a enfermedad por virus del ébola. Para tal propósito, se utilizan datos recolectados por el Cuerpo Internacional de Medicina (IMC, por sus siglas en inglés) durante 2014 y 2016 en Liberia y Sierra Leona. El fin del presente proyecto es evaluar el desempeño de diferentes métodos de optimización numérica en un contexto de *machine learning* utilizando métodos cómputo simples y en paralelo. En particular, se mide el desempeño tanto en términos del uso y tiempo de recursos computacionales para lograr convergencia, como con relación a métricas de desempeño del modelo. Finalmente, se realiza un análisis de los resultados en torno a la pregunta planteada inicialmente y el modelo de regresión asociado.

Este reporte está organizado como se menciona a conitinuación. La sección 2 menciona los objetivos a cumplir, e introduce el problema a resolver. La sección 3 introduce literatura relacionada al problema. La sección 4 describe el conjunto de datos a utilizar. La sección 5 describe la implementación de los métodos numéricos en un contexto de *machine learning*. La sección 6 describe la implementación y algoritmos utilizados para solucionar el problema. La sección 7 discute los resultados obtenidos. Finalmente, la sección 8 presenta las conclusiones.

2 Objetivos y problema a resolver

El objetivo principal de este trabajo es evaluar el rendimiento de los métodos de optimización mencionados en la sección 6. Dado lo anterior, definimos rendimiento en términos de métricas de desempeño del modelo, y también en métricas de desempeño durante el proceso de computo. A continuación, se describe el planteamiento del problema a resolver.

Problema de clasificación

Como es resaltado por Murphy (2012), el problema de regresión logística es una generalización del problema de regresión lineal, convirtiendolo hacia un problema de clasificación, siempre y cuando la variable de respuesta sea de carácter binario (i.e. $y \in \{0, 1\}$), y por tanto se pueda asumir que sigue una distribución Bernoulli. En este caso, dicha variable de respuesta corresponde a si el paciente muere a casua del virus del ébola (1), o no (0).

Dado lo anterior, utilizamos:

$$Pr[y \mid \boldsymbol{x}, \boldsymbol{w}] = Ber(y \mid \mu(x)) \tag{1}$$

donde la media, se define en términos de la probabilidad de que el paciente muera, es decir, $\mu(x) = E[y \mid x] = p(y = 1 \mid x)$. Donde el conjunto de variables explicativas está representado por x.[1]

Por otro lado, para la realización del computo de la media, se utiliza la función sigmoide, $\sigma(x)$, la cual garantiza que dada una combinación lineal de las variables explicativas, con los parámetros del modelo (i.e. $\beta^T x$), se cumpla que $0 \leq \mu(x) \leq 1$:

$$\mu(x) = \sigma\left(\beta^T x\right) \tag{2}$$

donde σ está definida por:

$$\sigma(x) = \frac{1}{1 + \exp(-x)} = \frac{e^x}{e^x + 1} \tag{3}$$

Los resultados asociados a la ecuación (2) están dados en términos de probabilidades. Así, es necesario definir un umbral clasificatorio para definir si el modelo predice que el paciente fallezca, o no. En este ejercicio, como se hace usualmente, y como realizan Colubri et al. (2019), se toma como umbral el valor de 0.5. Es decir:

$$\hat{y} = \begin{cases} 0, & \text{si } \sigma(\hat{\beta}^T x) < 0.5 \\ 1, & \text{si } \sigma(\hat{\beta}^T x) \geq 0.5 \end{cases} \tag{4}$$

Los parámetros asociados a las variables regresoras, $\hat{\beta}$, son estimados con el conjunto de datos de entrenamiento.

Función de pérdida

La función de pérdida asociada a este problema es la log-verosimilitud negativa, o entropía cruzada, definida de las siguiente forma:

$$LVN(\beta) = -\sum_{i=1}^{N} [y_i log \mu_i + (1 - y_i) log(1 - \mu_i)] \tag{5}$$

Dada la definición de (5), se tiene que este problema no tiene solución analítica. Es por esta razón que para minimizar la función de pérdida en torno a β es requerido utilizar métodos de optimización numérica.

Para los algoritmos a utilizar, es imprecindible la utilización del gradiente y la matriz hessiana asociados a (5), los cuales se definen a continuación:

$$\nabla LVN = \frac{d}{d\boldsymbol{\beta}} f(\boldsymbol{\beta}) = \sum_i (\mu_i - y_i) x_i = X^T(\mu - y) \tag{6}$$

$$\nabla^2 LVN = \frac{d}{d\boldsymbol{\beta}} g(\boldsymbol{\beta})^T = \sum \left(\nabla_\beta \mu_i\right) x_i^T = \sum \mu_i(1 - \mu_i) x_i x_i^T = \boldsymbol{X}^T \boldsymbol{S} \boldsymbol{X} \tag{7}$$

donde $\boldsymbol{S} \triangleq diag(\mu_i(1 - \mu_i))$. Adicionalmente, como resalta Murphy (2012), dado que $\boldsymbol{H}$ es positiva definida, entonces (5) tiene un mínimo global que puede ser alcanzable utilizando métodos de optimización.

En las siguientes secciones se expone la literatura relacionada con el problema a resolver y se describe el conjunto de datos.

[1] Las variables independientes son detalladas en las sección 4.

3 Revisión de Literatura

Como se mencionó al principio, el presente trabajo se apoya principalmente en el artículo *Machine Learning Models from 2014-2016 Ebola Outbreak-Data-harmonization Challenges, Validation, Strategies and mHealth Applications*, cuyo objetivo principal fue diseñar una app que, dado los síntomas de un paciente con ébola, calcula la probabilidad de muerte del paciente y recomienda un tratamiento personalizado. Lo anterior, es resultado de la implementación de diferentes modelos de regresión logística, mediante los cuáles se identificaron los principales factores asociados a la muerte de una persona con ébola (case fatality rate, CFR).

3.1 Internal Medical Corps (IMC): Conjunto de datos de Entrenamiento

Considerando la situación sanitaria del 2014-2016, la app fue focalizada en regiones con mayor afectación y menor posibilidad de acceso a atención médica. Así, los modelos se entrenaron con datos provenientes de cinco unidades de tratamiento del ébola (ETU, por sus siglas en inglés), localizadas en Sierra Leona y Liberia. La base de datos constaba con 470 observaciones (correspondientes a los pacientes atendidos), con variables como temperatura, escala de bienestar, time to presentantion (TTP), triaje, entre otras; además, a partir de la reacción en cadena de polimerasa, se calculó el cycle threshold (CT).

3.2 Análisis bivariado y exploratorio

Con estos datos se procedió con un análisis bivariado y exploratorio entre la variable dependiente (muerte o no muerte)y el resto de las variables, identificando la asociación entre cada uno de los posibles regresores y la variable dependiente. Para evaluar la asociación entre las variables numéricas y la variable dependiente, se realizaron *point biserial correlation*; para evaluar la asociación entre dos variables categóricas, se realizaron pruebas chi cuadradas con corrección de Yates.

El resultado de este primer análisis fue que más de 50 por ciento de los pacientes que murieron por ébola reportaron síntomas de pérdida de apetito, fiebre, astenia, dolor musculoso esquelético, dolor de cabeza y diarrea. Además, se identificó prevalencia de triajes notablemente diferente entre los resultados (mortalidad y no mortalidad de los pacientes). Finalmente, observando el *p-value* se identificó que pocas variables se asocian significativamente con el resultado del paciente: CT, edad, ictericia (jaundice) –con un p-value menor a 0.05; conjuntivitis, confusión, disnea, dolor de cabeza y sangrado – con un p-value menor a 0.15.

3.3 Imputaciones Múltiples

Un problema suscitado al trabajar con estos datos, fue la cantidad de valores faltantes (22 por ciento respecto del total de potenciales predictores). Con base en lo anterior, se tomaron las siguientes acciones: se eliminaron los predictores potenciales que tuvieran más de 50 por ciento de valores faltantes; se evaluó la idoneidad de imputar los valores faltantes mediante la prueba Little's MCAR, cuya hipótesis nula establece que los datos faltantes se debe totalmente al azar y se implementa probando si las medias de las variables de interés, a través de distintos grupos no cambian. Resultado del test, se aceptó que los datos faltantes se debían completamente al azar y se realizó una imputación múltiple con la función aregImpute del paquete de R Hmisc. En resumen, la imutación consistió en la generación de: una distribución predictiva bayesiana a partir de los datos conocidos y un número N de conjuntos de datos imputados. Cada valor faltante en la i-ésima imputación se predijo a partir de un modelo aditivio ajustado en una muestra de bootstrap con reemplazo de los datos originales.

3.4 Selección de Variables

En búsqueda de un modelo parsimonioso, los autores ejecutaron un Elastic Net Regularization, misma que combina las normas $L1$ y $L2$ de las regresiones de Lasso y Ridge, para la selección del subconjunto de características relevantes y no redundantes. Los criterios de selección de los regresores candidatos no redudantes y relevantes fueron los siguientes: se seleccionaron las variables con un coeficiente positivo en al menos 50 por ciento de las regresiones penalizadas; se eliminaron las variables con p-value bajo y se incorporaron las variables que tuvieron una asociación débil en el análisis bivariado. El resultado fue el conjunto de variables usadas en el modelo que denominaron *parsimonioso*.

Además, las variables seleccionadas fueron evaluadas por sesgo; luego, se aplicaron splines cúbicos restringidos para modelar las relaciones no lineales entre CFR y edad y CFR y temperatura corporal, donde los valores reales de edad y temperatura fueron la entrada del procedimiento de ajuste del spline, tal como se implementó en el paquete Hmisc.

3.5 Modelos de Regresión Logística: Modelo Parsimonioso y Modelos Alternativos

Derivado del análisis y procedimiento anterior, se implementaron varios modelos: el modelo parsimonioso – edad del paciente, edad del paciente al cubo, temperatura corporal, temperatura corporal al cubo, amarillamiento, sangrado, disnea, disfagia, TTP y CT x TTP– y tres modelos adicionales que se emplearon cuando los datos de triaje eran limitados: parsimonioso sin temperatura –idéntico al parsimonioso pero con eliminación de la temperatura corporal–, sólo clínico –ictericia, sangrado, disnea, disfagia, debilidad y diarrea– y el modelo mínimo –TC y edad. Cada modelo final se obtuvo ajustando N copias del modelo, en cada conjunto de datos imputados, y promediando esas copias en un solo modelo usando 'fit.mult.impute' de Hmisc.

Una predicción se clasificó como muerte cuando la puntuación del modelo superó el umbral de 0.5. Los intervalos de confianza (IC) de todas las estimaciones de rendimiento se calcularon utilizando la transformación de Fisher. Las razones de probabilidad (OR) de la regresión logística se convirtieron en razones de riesgo (RR).

3.6 Resultados

El modelo parsimonioso representó de manera adecuada las probabilidades reales de muerte, con cierta subestimación para pacientes de bajo riesgo de muerte.

Los modelos mínimos y solo clínicos resultaron menos calibrados, con el primero subestimando las probabilidades reales tanto en el lado de bajo y alto riesgo, y el segundo sobreestimando las probabilidades reales para una amplia gama de riesgos por encima del 60 por ciento.

Los índices de validación de todos modelos, junto con sus intervalos de confianza (al 95 por ciento), se superponen. La especificidad y R2 fue mayor en el modelo parsimoniosos. Mientras que la sensibilidad fue ligeramente mayor en el modelo mínimo. El modelo clínico muestró un rendimiento consistentemente menor.

Para el caso del modelo parsimonioso, la clasificación de todas las variables por su importancia, medida por el estadístico Wald $\tilde{\chi}^2$ – prueba estadística paramétrica que se utiliza para poner a prueba el verdadero valor del parámetro basado en la estimación de la muestra–, están dadas en el siguiente orden: CT, edad del paciente, ictericia, hemorragia, temperatura corporal y la interacción $CT - TTP$.

En términos de los odds ratios del modelo parsimonioso, se concluyó que la presentación de ictericia o sangrado se asocia con más del doble de las probabilidades de muerte, aunque su preva-

lencia es baja (al 5 por ciento). Finalmente, al transformar los riesgos de probabilidad en razones de riesgo, se identificó una estimación del 4 por ciento de aumento en el riesgo asociado con la aparición de esos síntomas.

3.7 Validación externa: KGH y GOAL

Los resultados fueron validados con dos bases de datos –provenientes del Kenema Government Hospital (KGH) y GOAL–; el primero constaba con 106 casos con ébola y una tasa de fatalidad (CFR, por sus siglas en inglés) de 73 por ciento; el segundo constaba de 158 casos con ébola, con una tasa de fatalidad de 60 por ciento.

Al igual que la base de datos de entrenamiento, ambas bases de datos empleadas durante la validación, contaban con registros incompletos, razón por la cuál se les aplicaron también imputaciones múltiples.

En general, los resultados de la validación fueron los siguientes: todos los modelos, con excepción del modelo clínico, exhibieron un rendimiento consistente en términos del área bajo la curva (AUC) y precisión general, con diferencias análogas en términos de especificidad y sensibilidad. El modelo parsiomonioso presentó mayor especificidad, mientras que el modelo mínimo fue más sensible. El impacto de la imputación fue intrascendente.

4 Conjunto de Datos

Por la restricciones de uso de la base de datos de *entrenamiento* original (además de una serie de procedimientos díficiles de seguir como protocolos de investigación y evidencia de aprobación por nuestro protocolo de investigación por parte de un Comité de ética independiente), optamos por trabajar con una de las dos bases de datos que los autores emplearon para *validar* sus modelos: KGH. La base de datos en mención, como se mencionó en la sección anterior, consta de 106 casos positivos de pacientes con ébola y un CFR global de 73 por ciento. Originalmente, previo al tratamiento de los datos, la base tenía únicamente 44 registros de triaje, 58 registros de carga viral, con un total de 78 valores faltantes en todo el data set.

Para harmonizar los datos, los autores transformaron la carga viral en CT, conforme con la curva estándar qPCR:

$$log_{(carga\ viral)} = m * CT + c_0 \tag{8}$$

Los resultados de la transformación no dependieron del origen geográfico de los datos IMC. Sin embargo, el origen geográfico tuvo un impacto significativo ($p < 0.0001$) en la distribución de CT a través de los sitios.

Para corregir dicho sesgo analítico y garantizar que los resultados fueran internamente consistentes en cada sitio y, por lo tanto, comparables entre los sitios, se normalizaron los valores de CT de cada sitio, mediante el escalado de características (restando la media y dividiendo por la desviación estándar).

Como las diferencias en CT también podrían provocar diferencias en los comportamientos de búsqueda de atención, entonces se normalizaron los valores de la CT dentro de los sitios también contribuiría a reducir el efecto de este posible factor de confusión -una variable o factor que distorsiona la medida de asociación entre otras dos variables.

Nosotros, para fines del presente trabajo, empleamos una de las versiones imputadas de esta base de datos, dispuesta en el siguiente sitio: ebola-imc-public, misma que cuenta con 11 variables.

Así, para evaluar el rendimiento de los métodos de optimización mencionados en la sección 5, realizaremos dos regresiones logísticas: el modelo de regresión logística clínicas (ictericia, sangrado,

disfagia, debilidad y diarrea), junto con el CT y el modelo de regresión logística simple (CT y edad como regresores). Esto principalmente se debe al limitado insumo inicial (la base de datos dispuesta al público), mismo que contiene un sub-conjunto reducido de las variables originales empleadas por los autores para el entrenamiento del modelo.

Tipo	Nombre	Descripción
Variables Numéricas	CT	El cycle threshold (CT) es una variable que se calcula a partir de una relación médica bien conocida (qPCR) y la carga viral (una expresión númerica de la cantidad de virus dado un volúmen de fluido que normalmente se correlaciona con la severidad de una infección viral activa).
	TEMP	Temperatura corporal del paciente. Toma valores de 36 a 39.9
	AGE	Edad del paciente. Toma valores de 1 a 73.
Variables Categóricas	HEADCH	Presencia o no dolores de cabeza. Toma valores valores 1 o 0, dependiendo de si el paciente presenta o no dolores de cabeza.
	BLEED	Presencia o no de sangrado. Toma valores valores 1 o 0, dependiendo de si el paciente presenta o no sangrado.
	DIARR	Presencia o no de diarrea. Toma valores valores 1 o 0, dependiendo de si el paciente presenta o no diarrea.
	VOMIT	Dificultad para comer, conocido como disfagia, término técnico para describir el síntoma consistente en dificultad para la deglución (problemas para tragar). Esta dificultad suele ir acompañada de dolores, a veces lancinantes (disfagia dolorosa u odinofagia . Toma valores valores 1 o 0, dependiendo de si el paciente presenta o no de disfacia.
	PABD	Presencia o no de PADB*.
	WEAK	Presencia o no de debilidad o fatiga general.Toma valores valores 1 o 0, dependiendo de si el paciente presenta o no debilidad.
	JAUN	Condición en la cuál la piel, los ojos y los miembros mucosos que vuelven amarillos debido a altos niveles de bilirubina. Toma valores valores 1 o 0, dependiendo de si el paciente presenta o no ictericia.
	OUT	Muerte o no muerte del paciente. Toma valores 1 o 0. Dependiendo de si el paciente muere o no muere.

Tabla 1: Variables del Dataset Kenema Government Hospital (KGH): Versión Imputada 50. *: PABD se refiere a Donación de sangre autóloga -personas en donde los trasplantes y las transfusiones la misma persona es a la vez donante y receptora- preoperatoria (*Preoperative Autologous Blood Donation*).

5 Algoritmos de optimización

Retomando lo descrito en la sección 2, para cumplir el objetivo de minimizar la entropía cruzada (ecuación (5)) utlizaremos cuatro algoritmos de optimización numérica para encontrar el vector $\hat{\beta}$. Los algoritmos a utilizar serán: el método de máximo descenso, método de Newton, método BFGS (Broyden, Fletcher, Goldfarb y Shanno), y SGD[2]. En particular, se aplican algunas variantes a los mismos, y se comparan las iteraciones para converger, además del tiempo de ejecución.

En lo que sigue de esta sección utilizaremos la notación de Nocedal and Wright (2006), en la cual se plantea que en un método de optimización se inicia con β_0 y el algoritmo genera una secuencia de iteraciones $\{\beta_k\}_{k=0}^{\infty}$ que se detienen cuando no sea se pueda progresar más en el proceso, o sea

[2]Por sus siglas en inglés, *Stochastic Gradient Descent*.

haya alcanzado una solución con precisión suficientemente buena. Así, los algoritmos incorporan un proceso según el cual la decisión entre moverse de β_k a β_{k+1} depende de haber alcanzado un valor menor en la función objetivo.

5.1 Método de máximo descenso

El método de máximo descenso es el método de búsqueda de línea más sencillo, y plantea que la forma de encontrar el mínimo de una función $f(\beta_k)$ es por medio de la búsqueda iterativa de β_k siguiendo la dirección negativa del gradiente asociado, i-.e. $-p^T \nabla f_k$[3]. Cabe anotar que, como es resaltado por Nocedal, este supuesto está basado en el Teorema de Taylor, el cual establece que:

$$f(\beta_k + \alpha_k p) \approx f(\beta_k) + \alpha_k p^T \nabla f_k \tag{9}$$

donde α_k es el paso del descenso o tasa de aprendizaje en la iteración k. Así, en (9) la función f disminuye a una tasa de paso, multiplicada por el gradiente y el paso, y una dirección p, es decir: $p^T \nabla f_k$. Por otro lado, dado que el gradiente es negativo se cumple que:

$$f(\beta_k + \alpha_k p) < f(\beta_k) \tag{10}$$

Adicional a lo anterior, es de vital importancia adicionar un componente al problema, correspondiente a la elección de la tasa de aprendizaje, α, en la medida que, valores muy altos de la misma, pueden impedir que haya convergencia hacia el mínimo y el algoritmo se estanque, o que su convergencia sea demasiado lenta.

Así, como es presentado por Nocedal and Wright (2006) y Murphy (2012), el último componente que se requiere adicionar al problema de optimización planteado, y que a su vez lo convierte en un método de búsqueda de línea o minimización de línea [4], es elegir α tal que se minimice:

$$\varphi(\alpha) = f(\beta_k + \alpha_k p) \tag{11}$$

En particular, una condición necesaria para obtener el óptimo del problema anterior es que $\varphi'(\alpha) = 0$, lo que a su vez implica que $\varphi'(\alpha) = p^T \nabla f_k$, y por tanto $\nabla f_k = 0$. Lo anterior garantiza que se ha llegado a un punto estacionario, correspondiente a un punto mínimo de la función f_k. En general, las iteraciones para llegar a lograr esta convergencia dependen del algoritmo utilizado y del punto inicial del mismo, β_0.

En el método de descenso de gradiente, la forma de actualizar el vector de parámetros es como se define a continuación:

$$\beta_{k+1} = \beta_k - \alpha \nabla f_k \tag{12}$$

De esta forma dado un β_0 inicial elegido aleatoriamente, la actualización de los pesos (12), continúa hasta que $\|\nabla f_k\| \approx 0$ o hasta que un número definido de iteraciones se alcance.

Cabe adicionar que, como mencionan Nocedal and Wright (2006) , una de las ventajas del método de descenso de gradiente es que solo requiere el cálculo del gradiente, sin embargo, también puede tener un desempeño bastante lento para problemas difíciles.

[3] donde ∇f_k es el gradiente de la función f con respecto a β_k.

[4] En inglés *line search*.

5.2 Método de Newton

El método de Newton es parte la familia de métodos de optimización de segundo orden, en la medida que tienen en cuenta la curvatura de la función a optimizar, la cual es incorporada por la matriz Hessiana, $\nabla^2 f$. Lo anterior, implica un mayor costo en términos de cómputo con respecto al algoritmo de máximo descenso. Sin embargo, la existencia de dicha curvatura, en la mayoría de los problemas, permite lograr niveles de convergencia mucho más rápido.

En particular, en este método, la forma de actualizar el vector de parámetros es de la siguiente manera:

$$\beta_{k+1} = \beta_k - \alpha_k \nabla^2 f_k^{-1} \nabla f_k \tag{13}$$

Como es anotado por Nocedal, el método de Netwton requiere que $\nabla^2 f_k$ sea una matriz definida positiva, lo que se cumple en caso de que también sea estrictamente convexa. Cabe mencionar que, dependiendo del problema, en ocasiones es computacionalmente costoso computar de forma exacta la matriz hessiana de f_k, por lo que hacer aproximaciones en el proceso iterativo puede ser útil para resolver el problema de minimización, como se menciona en la siguiente subsección, con el método BFGS.

5.3 Método BFGS

El método de propuesto por Broyden, Fletcher, Goldfarb and Shanno (BFGS) es un algoritmo quasi-newtoniano que aborda una solución a los problemas en los cuales es muy costoso el cómputo de la matriz Hessiana de f_k. Como es descrito por Murphy, este método iterativamente computa una aproximación de $\nabla^2 f_k$ a partir del gradiente, ∇f_k.

Dado lo anterior, la aproximación de la matriz Hessiana se define como $H_k \approx \nabla^2 f_k$, y se calcula con una función de rango 2 dada por:

$$H_{k+1} = H_k + \frac{w_k (w_k)^T}{(w_k)^T z_k} - \frac{H_k z_k (H_k z_k)^T}{(z_k)^T H_k z_k} \tag{14}$$

$$z_k = \beta_{k+1} - \beta_k \tag{15}$$

$$w_k = \nabla f_{k+1} - \nabla f_k \tag{16}$$

Como es mencionado por Murphy (2012), por lo general el algoritmo se inicializa con $H_o = I$.

5.4 Método de descenso de gradiente estocástico - SGD[5]

El método SGD impone un problema diferente a los planteados anteriormente. En particular, en este caso se quiere minimizar la pérdida en términos esperados. Es decir:

$$f(\beta) = E[f(\theta, z)]$$

Donde la ecuación (5.4) implica calcular el valor esperado de la función de pérdida sobre datos futuros que son muestreados de los datos originales. Dado que estos datos de entrada son de carácter aleatorio, esto implica que estamos realizando optimización aleatoria.

En nuestro caso, podemos definir una función de riesgo empírico de la siguiente forma:

[5]Por sus siglas en Inglés, *Stochastic Gradient Descent*

$$L_{emp} = \frac{1}{m}\sum_{i=1}^{m} y_i log(\mu_i) + (1-y_i) log(1-\mu_i) \tag{17}$$

donde la media corresponde a $\mu_i = (1+e^{-\beta^T x_i})^{-1} = \sigma(\beta^T x_i)$.
Por otro lado, el gradiente de la función de riesgo esta dado por:

$$\nabla L = \frac{dL}{d\mu_i} = \frac{1}{m}\sum_{i=1}^{m} x_i(\mu_i - y_i) \tag{18}$$

En cada iteración i se aplica el método de descenso del gradiente. Dado que durante el proceso iterativo se hacen varios muestreos, se calcula un promedio con todas las iteraciones para poder calcular $\hat{\beta}$:

$$\hat{\beta} = \frac{1}{m}\sum_{i=1}^{m} \beta_i \tag{19}$$

5.5 Método de descenso de gradiente estocástico en paralelo

Para paralelizar el método del descenso del gradiente estocástico se utilizó una estrategia directa. En particular, se dividen los datos de entrada de manera aleatoria entre el número de núcleos que tenemos en el equipo en el cual se ejecuta el algoritmo utilizando la librería *dask* de *python*. Se manda a cada *worker* el *lote*[6] de datos que debe de procesar. Posteriormente, se hace el cálculo como si fuera una lote cualquiera entrenando en un proceso normal. Al hacer la actualización del vector $\hat{\beta}$ se juntan todas las direcciones de descenso y se hace la actualización del paso.

6 Implementación

Con el fin de generar un entorno aislado que permita evaluar el desempeño del proceso de entrenamiento en término de recursos computacionales, se utilizará una instancia EC2 de Amazon Web Services (AWS). Adicionalmente, para tener un control del entorno virtual asociado, se creará una imagen de docker asociada al repositorio donde se encontrarán todos los códigos de este proyecto.[7]

La implementación de este ejercicio se realizará por medio de código escrito en lenguaje Python. En particular, la totalidad del ejercicio se realizó por medio de la escritura de módulos propios, que en su gran mayoría hacen uso de las librerías: numphy, scipy, y sci-kit learn.

En la sección 2 se definieron computos que son utilizados por la totalidad de algoritmos implementados, en particular, a continuación se enlistan las ecuaciones y sus funciones asociadas. Estas funciones aparecen en código de Python en la lista de código 1 de los apéndices. Adicionalmente, se incorpora la función "normaliza", la cual se utiliza para normalizar la dirección de descenso.

- Cálculo del promedio de una distribución Bernoulli: [ecuación (2)], función: *calcmu(X, beta)*
- función que calcula el sigmoide [ecuación (3], función: *sigmoide(z)*

[6] Uno de los subconjuntos de los datos seleccionados de manera aleatoria.

[7] La documentación para ejecutar el ambiente de docker, se encuentra en el repositorio de Github asociado a este proyecto, en este archivo. La documntación relacionada con AWS se encuentra en la siguiente carpeta.

- función de clasificación [ecuación (4], función: *clasifica(X, betahat,limit)*
- cálculo de la entropía cruzada [ecuación (5)], función: *f(X,y,beta)*
- gradiente de la entropía cruzada [ecuación (6)], función: *gradientef(X,y,beta)*
- Hessiana de la entropía cruzada [ecuación (14)], función: *hessianaf(X,y,beta)*
- normalización de un vector, función: *normalize(x)*

6.1 Algoritmos

De acuerdo a lo presentado en la sección 5, a continuación se plantea el algoritmo para cada método y su código asociado disponible en las listas de la sección de apéndices (sección 8).

Inicialmente, dado que todos los métodos estudiados comparten el uso de una dirección de descenso, p_k^T, se definió una función que calcule este componente de acuerdo al método especificado. La lista de Código 2 presenta la implementación en Python.

6.1.1 Método de máximo descenso

En la lista de código 1 se presenta el pseudo-algoritmo asociado al método de máximo descenso presentado en la sección 5:

Algorithm 1: Método de máximo descenso

Result: Devuelve $\hat{\beta}_0$
Definir MaxIter y conv. Se inicializa β_o aleatorio;
while *MaxIter*> *iter* **do**
  if $\|\nabla f_k\| >$ *conv* **then**
    Evaluar $\nabla f_k(\beta_k)$;
    Actualizar α_k con condiciones de Armijo;
    $\beta_{k+1} \leftarrow \beta_k - \alpha_k \nabla f_k$;
  end
  detenerse;
end

Como se puede observar en la lista de Código 2, en la línea 3, para este método se ajusta la dirección de descenso correspondiente al gradiente, lo cual es acorde con lo presentado en el algoritmo 1.

6.1.2 Método de Newton

En la lista de código 1 se presenta el pseudo-algoritmo asociado al método de Newton presentado en la sección 5:

Algorithm 2: Método de Newton

Result: Devuelve $\hat{\beta}_0$
Definir MaxIter y conv. Se inicializa β_o aleatorio;
while *MaxIter*> *iter* **do**
  if $\|\nabla f_k\| >$ *conv* **then**
    Evaluar $\nabla f_k(\beta_k)$;
    Evaluar $\nabla^2 f_k(\theta_k)$;
    Resolver d^k en $(\nabla^2 F(\beta^k))d^k = -\nabla F(\beta^k)$;
    Actualizar α_k con condiciones de Armijo;
    $\beta_{k+1} \leftarrow \beta_k + \alpha_k \nabla^2 f_k^{-1} \nabla f_k$;
  end
  detenerse;
end

Como se puede observar en la lista de Código 2, en la línea 5, para este método se ajusta la dirección de descenso dependiendo del gradiente y la Hessiana del problema como la solución a un sistema de ecuaciones, lo cual es acorde con lo presentado en el algoritmo 2.

6.1.3 Método de BFGS

Algorithm 3: Método de BFGS

Result: Devuelve $\hat{\beta}_0$
Definir MaxIter y conv. Se inicializa β_o aleatorio y $H_0 = \boldsymbol{I}$;
while *MaxIter*> *iter* **do**
  if $\|\nabla f_k\| >$ *conv* **then**
    Evaluar $H_{k+1} = H_k + \frac{w_k (w_k)^T}{(w_k)^T z_k} - \frac{H_k z_k (H_k z_k)^T}{(z_k)^T H_k z_k}$;
    Evaluar $z_k = \beta_{k+1} - \beta_k$;
    Computar $w_k = \nabla f_{k+1} - \nabla f_k$;
    Actualizar α_k con condiciones de Armijo;
    $\beta_{k+1} \leftarrow \beta_k + \alpha_k \nabla^2 f_k^{-1} \nabla f_k$;
  end
  detenerse;
end

Como se puede observar en la lista de Código 2, en la línea 5, para este método se ajusta la dirección de descenso dependiendo del gradiente y la hesiana del problema, lo cual es acorde con lo presentado en el algoritmo 3.

Por otro lado, de lo mencionado inicialmente, se tiene que estos métodos son de búsqueda de línea,lo que implica que también se resuelve un problema de minimización que incopora un ajuste de la tasa de aprendizaje, α, durante el proceso iterativo. Así, para lo anterior, se utilizan las condiciones de Armijo, expuestas en Nocedal and Wright (2006) para realizar dicho ajuste:

$$f(\beta_k + \alpha p_k) \leq f(\beta_k) + c_1 \alpha_k \nabla f_k^T p_k \tag{20}$$

para una constante $c_1 \in (0, 1)$. La ecuación (20) implica que la disminución en la función f es proporcional tanto a la tasa de aprendizaje α_k como al gradiente de la función. El proceso iterativo es presentado en la lista de código 4 de los apéndices.

Una vez se han presentado el procedimiento para calcular la dirección de descenso, y la tasa de aprendizaje, es factible presentar el algoritmo principal del método de descenso de gradiente (disponible en la lista de código) que calcula $\hat{\beta}_0$ para cada uno de los métodos.

En este sección se presentaron los pseudo-algoritmos y el código asociado a cada implementación, embargo, cabe anotar que los productos intermedios fueron omitidos, y se encuentran en los módulos de este proyecto. A modo de ejemplo, a continuación se ejemplifica la salida de la implementación del método de descenso de gradiente para el método de máximo descenso, Newton, BFGS, SGD, y SGD en paralelo.

```
================================================================
Método de máximo descenso
================================================================
Iteración: 500 gradiente: 2.5045446E-01, alpha: 7.1790E-02
Iteración: 1000 gradiente: 1.9681746E-02, alpha: 5.7264E-03
Iteración: 1500 gradiente: 3.6459555E-03, alpha: 1.0611E-03
Iteración: 2000 gradiente: 4.4262414E-04, alpha: 1.2901E-04
Iteración: 2500 gradiente: 2.3181024E-05, alpha: 6.7516E-06


** Resultados Finales
* Total de iteraciones: 2964
* Norma del gradiente de f: 9.756061172576794e-07
* beta_hat:
array([-10.36242902,  16.25102342,  -3.90446862,   0.60600272,
         0.77744891,   4.55476501,  -0.91897329,   3.3866912 ,
         3.76095218,   3.9617985 ,   4.98358393,   6.21113331])


* Error de clasificación: 4.55%
================================================================
CPU times: user 410 ms, sys: 26 ms, total: 436 ms
Wall time: 419 ms
```

Figura 1: Implementación del método de descenso de gradiente en Python

```
================================================================
Método de Newton
================================================================
Iteración: 20 gradiente: 4.3186382E-01, alpha: 1.0000E+00
Iteración: 40 gradiente: 6.6389532E-04, alpha: 7.0697E-03
Iteración: 60 gradiente: 5.1340283E-05, alpha: 5.6392E-04
Iteración: 80 gradiente: 2.6953093E-06, alpha: 2.9513E-05


** Resultados Finales
* Total de iteraciones: 87
* Norma del gradiente de f: 1.1609129137769218e-06
* beta_hat:
array([-10.36243506,  16.25103288,  -3.90446995,   0.60600623,
         0.77744971,   4.55476817,  -0.91897444,   3.3866927 ,
         3.76095382,   3.96180007,   4.98358595,   6.21113592])


* Error de clasificación: 4.55%
================================================================
CPU times: user 125 ms, sys: 7.72 ms, total: 132 ms
Wall time: 41.9 ms
```

Figura 2: Implementación del método de Newton en Python

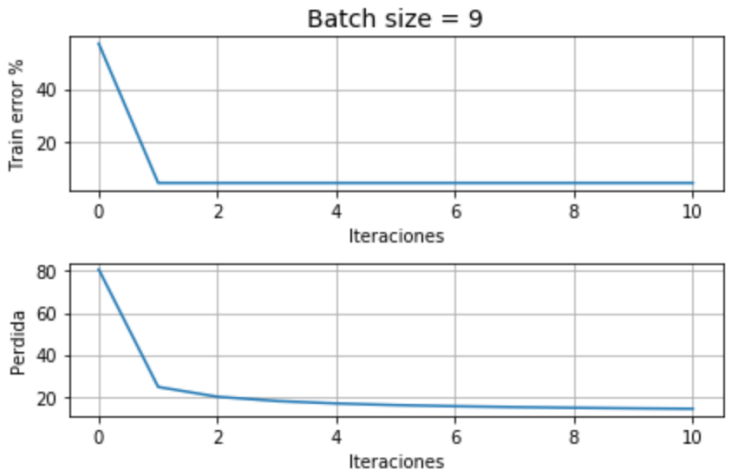
```
================================================================
Método de BFGS
================================================================
Iteración: 10000 gradiente: 7.0627661E-02, alpha: 2.0276E-02
Iteración: 20000 gradiente: 3.6716352E-03, alpha: 1.0611E-03
Iteración: 30000 gradiente: 1.9548109E-04, alpha: 5.5533E-05
Iteración: 40000 gradiente: 2.3716148E-05, alpha: 6.7516E-06


** Resultados Finales
* Total de iteraciones: 47837
* Norma del gradiente de f: 1.1624129428409921e-06
* beta_hat:
array([-10.36240975,  16.25097655,  -3.90446056,   0.60599614,
         0.77744804,   4.55475336,  -0.91896656,   3.38668298,
         3.76094818,   3.96179347,   4.98357503,   6.21112378])


* Error de clasificación: 4.55%
================================================================
CPU times: user 8.45 s, sys: 63.9 ms, total: 8.51 s
Wall time: 8.58 s
```

Figura 3: Implementación del método de BFGS en Python

```
Nº DE INTERACIONES:  100000
```

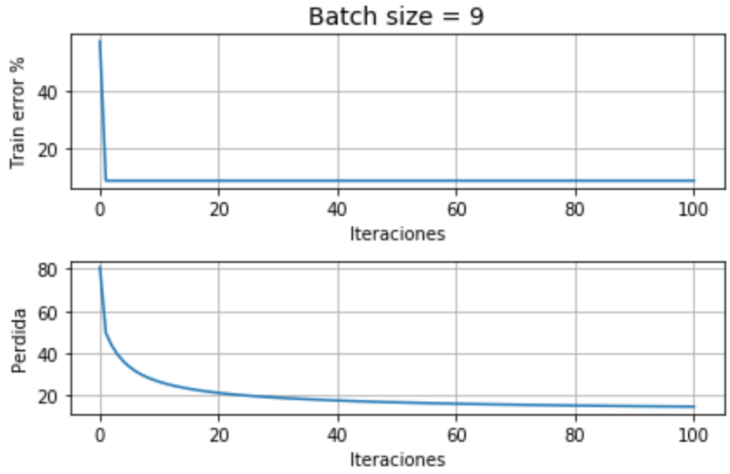



```
beta_hat= [-5.51625653  9.41234684 -2.34521111 -0.25427645  0.59709283
2.51511279
 -0.01347701  2.10364421  2.07589005  1.96217798  2.23616146  3.42584893]
Error de clasificacion= 9.09 %


CPU times: user 9.07 s, sys: 482 ms, total: 9.56 s
Wall time: 10.7 s
```

Figura 4: Implementación del método de SGD en Python. Tamaño de lote: 9

```
Nº DE INTERACIONES:  10000
```




```
beta_hat= [-4.91055688  8.77979807 -2.17311365 -0.32696179  0.58071703
2.325793
  0.04479777  2.00969346  1.83257881  1.70586735  1.89491893  3.07499889]
Error de clasificacion= 9.09 %


CPU times: user 16min 37s, sys: 1min 5s, total: 17min 43s
Wall time: 19min 30s
```

Figura 5: Implementación del método de SGD paralelo en Python utilizando 100 épocas. Tamaño de lote: 9

7 Resultados

7.1 Desempeño de los algoritmos

Los algoritmos fueron ejecutados en una máquina MacBook Pro (13-inch, 2019, Four Thunderbolt 3 ports), con procesador Quad-Core Intel Core i5 a 2.4 GHz, memoria RAM de 8 GB 2133 MHz LPDDR3, y con un sistema operativo macOs Catalina versión 10.15.4. Los resultados del entrenamiento del modelo se presentan en la tabla 2

		Método				
Variable		Máximo descenso	Newton	BFGS	SGD[2]	SGD paralelo[4]
Error de clasificación		4.55%	4.55 %	4.55 %	9.09 %	9.09%
$\|\|\nabla f_k\|\|$		1.12×10^{-6}	1.05×10^{-6}	1.18×10^{-6}	-	-
Criterio de parada		Convergencia	Convergencia	Convergencia	Iteraciones	Épocas (100)
Iteraciones		2,964	87	47,837	100,000	10,000
Tiempo	Usuario	410 ms	125 ms	8.45 s	7.54 s	11 min 54s
	Sistema	26 ms	7.72 ms	63.9 ms	1.01 s	42.3 s
	Total	436 ms	132 ms	8.51 s	8.62 s	12 min 36 s

Tabla 2: Desempeño de los algoritmos utilizados. **Nota: (1)** Los tiempos de ejecución están denotados en segundos (s) y milisegundos (ms). **(2)** Los tiempos reportados para el algoritmo SGD y SGD en paralelo son el promedio de una evaluación para el número de batches entre 9 y 10. **(3)** El criterio de máximo de iteraciones fue fijado en 10^5 y el de convergencia es 10^{-7}.

A modo general, para los algoritmos que no incorporan optimización estocática, tres conclusiones grandes se pueden hacer sobre lo presentado en la tabla 2:

- El algoritmo más eficiente en términos de computo (que incluye iteraciones y tiempo de cómputo) fue el método de Newton. Este método se detuvo por el criterio de convergencia establecido en tan solo 87 iteraciones. Por otro lado, el método más ineficiente en tiempo de computo e iteraciones es el BFGS, el cual logró convergencia luego de 47,837 iteraciones.
- Los algoritmos estudiados se detuvieron por el criterio de parada de convergencia.
- Aunque divergen en tiempo de cómputo, todos los algoritmos estudiados logran alcanzar una precisión de 95.45% y un $\hat{\beta}_0$ idéntico.

Por otro lado, en cuanto a los algoritmos que utilizan optimización estocática, cabe aclarar que los vectores estimados $\hat{\beta}$ no fueron iguales. Sin embargo, los valores no distan mucho en sus valores. En particular, una diferencia importante se da en cuanto al signo del parámetro asociado con
Para SGD se obtuvo:

$$\hat{\beta}_{SGD} = [\ -5.51 \quad 9.41 \quad -2.34 \quad -0.25 \quad 0.59 \quad 2.51 \\ -0.013 \quad 2.103 \quad 2.07 \quad 1.96 \quad 2.23 \quad 3.42\ ]$$

Por otro lado, para SGD en paralelo se obtuvo:

$$\hat{\beta}_{SGD_{paralelo}} = [\ -4.91 \quad 8.77 \quad -2.17 \quad -0.32 \quad 0.58 \quad 2.32 \\ 0.044 \quad 2.009 \quad 1.83 \quad 1.7 \quad 1.89 \quad 3.07\ ]$$

destacamos los siguientes resultados:

- Los algoritmos no convergieron hacia un mismo nivel de parámetros, sin embargo, lograron niveles de precisión similares.
- Los signos de los elementos para $\hat{\beta}$ son consistentes con los encontrados con los algoritmos que no usan optimización estocástica.
- El resultado esperado del algoritmo de computo en paralelo era disimuuir el tiempo de computo logrado.

7.2 Modelo de regresión logística: interpretación

Como se mencionó anteriormente, el problema de regresión logística es una extensión del problema de regresión lineal, aplicada hacia un problema de clasificación. En este sentido, la regresión logística modela las probabilidades de clasificación de una variable dependiente dentro de dos posibles categorías, 0 y 1.

Retomando los resultados de lo expuesto en la sección 2, y considerando las ecuaciones 3 y 4 , tenemos que la probabilidad de que un paciente muera a causa del virus del ébola para este modelo se define de la siguiente forma:

$$P\left(y=1\right) = \frac{e^{\beta^T x}}{1+e^{\beta^T x}} \tag{21}$$

Note que la ecuación (21) es equivalente a:

$$P\left(y=1\right)=\frac{e^{\beta_0+\beta_1 x_1+\ldots+\beta_p x_p}}{1+e^{\beta_0+\beta_1 x_1+\ldots+\beta_p x_p}} \tag{22}$$

En (22) cabe aclarar que para este ejercicio, no estamos considerando el término equivalente al intercepto en el caso de regresión lineal (i.e $\beta_0 = 0$). Por otro lado, dado lo presentado en (22), a diferencia de un modelo de regresión tradicional, en este caso los pesos no afectan de forma lineal las probabilidades. Por lo tanto, para lograr interpretabilidad en cuanto a las mismas, es necesario despejar el componente lineal de la ecuación. El resultado de este procedimiento se define en la literatura como el *logit,*[8] y corresponde al logaritmo del cociente entre la probabilidad de morir y la probabilidad de sobrevivir al virus (*ratio de probabilidades*):

$$\lambda=\log\left\{\frac{P\left(y=1\right)}{1-P\left(y=1\right)}\right\}=\beta_1 x_1+\ldots+\beta_p x_p \tag{23}$$

Como es posible observar en (23), esta ecuación corresponde a una línea recta, dependiente del conjunto de covariables, x.

Es importante mencionar otra diferencia fundamental entre el modelo de regresión logística y el modelo de regresión lineal tradicional. En este último, se asume homocedasticidad (los errores estándar de los coeficientes siguen una distribución normal con media 0 hay varianza constante). Este no es el caso en el modelo logístico, dada la naturaleza dicotómica de la variable de salida del mismo.

La interpretación de los resultados del modelo de regresión logística se realiza en términos del valor del ratio de probabilidades, que corresponde al operador exponencial de (23). Dicho operador se lee como el indicador de variación en el ratio de probabilidades como resultado del cambio en una unidad en la predicción.

Dado que, como se presentó en la sección 7.1, el valor de $\hat{\beta}$ convergió entre los modelos, a continuación se realiza su interpretación. Los resultados de la computación de dicho vector de parámetros, asociado al ratio de probabilidades, son los siguientes:

$$\begin{aligned} ratio\,probs = \exp(\log\{&-10.95x_1+18.88x_2-4.42x_3+1.14x_4+1.40x_5+4.76x_6-0.94x_7+\\ &3.006x_8+3.73x_9+4.45x_{10}+5.20x_{11}+6.38x_{12}\}) \end{aligned} \tag{24}$$

donde $x_1,\ldots,x_{12}$ corresponden a: *cycle threshold*, temperatura, dolor de cabeza, sangrado, diarrea, vómito, presencia de PABD, debilidad, edad hasta 22 años, edad entre 23 y 36 años, edad entre 37 y 45 años, y edad mayor a 45 años, respectivamente.

Dado que el ratio de probabilidades, (24), corresponde a la probabilidad de morir dividida sobre la probabilidad de sobrevivir, podríamos ver como varía esta medida cuando se cambian los valores de las covariables (x). Para lo anterior, se debe calcular un valor de referencia con el cual se pueda comparar. En este sentido, tomaremos dicho valor para el caso en el cual la persona no tiene ninguna complicación de salud adicional al virus del ébola, y su edad se encuentra en un rango de 0 años a 22 años. Es decir que en (24), se tiene $x_i = 0$ para $i = 3,4,5,6,7,8$ y $x_9 = 1$:

$$ratio\,probs=\exp\left(\log\left\{-10.95x_1+18.88x_2+3.73\right\}\right) \tag{25}$$

Si tomamos los valores promedio de *cycle treshold,* $\bar{x}_1$, y de temperatura, $\bar{x}_2$, tenemos lo siguiente:

[8]Note que λ es creciente entre $-\infty$ e ∞ a medida que $P\left(y=1\right)$ crece entre 0 y 1.

$$
\begin{aligned}
ratio\, probs_0 &= \exp\left(\log\left\{-10.95\bar{x}_1 + 18.88\bar{x}_2 + 3.73\right\}\right) \\
&= \exp\left(\log\left\{-10.95(25.72) + 18.88(37.25) + 3.73\right\}\right) \\
&= 425.75
\end{aligned} \tag{26}
$$

Por otro lado, si quisiéramos analizar, por ejemplo, cual es el efecto de que la persona tenga sangrado sobre la probabilidad de que la persona muera por presencia del virus del ébola, debemos calcular el ratio de probabilidades ahora con $x_4 = 1$:

$$
\begin{aligned}
ratio\, probs_1 &= \exp\left(\log\left\{-10.95\bar{x}_1 + 18.88\bar{x}_2 + 1.14 + 3.73\right\}\right) \\
&= \exp\left(\log\left\{-10.95(25.72) + 18.88(37.25) + 1.14 + 3.73\right\}\right) \\
&= 426.89
\end{aligned} \tag{27}
$$

Ahora, si calculamos el ratio entre $ratio\, probs_0$ y $ratio\, probs_1$, tenemos:

$$
\frac{ratio\, probs_1}{ratio\, probs_0} = \frac{426.89}{425.75} = 1.0026 \tag{28}
$$

Puesto que (28) es mayor a 1, esto indica que la probabilidad de la variable dependiente incrementa. Es decir, para una persona con edad de 0 años a 22 años que tiene ébola[9], tener sangrado, afecta positivamente su probabilidad de morir. Este análisis se puede hacer de manera similar para las demás variables independientes del modelo. Como se mencionó anteriormente, podemos verificar la dirección de la inferencia de las covariables, más no su efecto lineal sobre las probabilidades.

8 Conclusiones

En este ejercicio se construyeron seis algoritmos de optimización numérica para resolver el problema de regresión logística para estimar la probabilidad de muerte de pacientes de ébola para un problema con pocos datos. En general, los métodos que no incorporan optimización estocástica (método de descenso, Newton, y BFGS) tuvieron el mejor desempeño en términos de tiempo de computo y ocnvergencia. Por otro lado, los algoritmos que incorporan optimización estocástica tuvieron un bajo desempeño en términos de tiempo de cómputo y convergencia. Cabe aclarar que los anteriores resultados deben ser revisados y evaluados con un problema que incorpore una mayor cantidad de datos. Para un caso como el que se describe anteriormente, se espera superioridad tanto de SGD como de SGD en paralelo.

References

Colubri, A., Hartley, M.-A., Siakor, M., Wolfman, V., Felix, A., Sesay, T., Shaffer, J. G., Garry, R. F., Grant, D. S., Levine, A. C., et al. (2019). Machine-learning prognostic models from the 2014–16 ebola outbreak: Data-harmonization challenges, validation strategies, and health applications. *EClinicalMedicine*, 11:54–64.

Murphy, K. P. (2012). *Machine learning: a probabilistic perspective*. MIT press.

Nocedal, J. and Wright, S. (2006). *Numerical optimization*. Springer Science & Business Media.

[9] Y que tiene niveles de *cycle treshold* y temperatura iguales al promedio muestral.

Appendices

```
def calc_mu(X,beta):
    # función que calcula el sigmoide de X^T*beta
    a = np.matmul(beta,np.transpose(X))
    mu = sigmoide(a)
    return mu

def sigmoide(z):
    # función que calcula el sigmoide de z
    sig = 1/(1+ np.exp(-z))
    return sig

def clasifica(X, beta_hat,limit=0.5):
    # función que clasifica a 1 si la combinación lineal definida es mayor a limit
    mu = calc_mu(X,beta_hat)
    yhat = mu
    yhat[mu<limit] = 0
    yhat[mu>=limit] = 1
    return yhat

def f(X,y,beta):
    # función que calcula la Log-verosimilitud negativa
    prob = calc_mu(X,beta)
    lvn = -sum(y*np.log(prob)+(1-y)*(np.log(1-prob)))
    return lvn
def gradiente_f(X,y,beta):
    # función que calcula el gradiente evaluado
    mu = calc_mu(X,beta)
    grad = np.matmul(np.transpose(X), mu-y)
    return grad

def hessiana_f(X,y,beta):
    # función que calcula el gradiente evaluado
    mu = calc_mu(X,beta)
    S = np.diag(mu*(1-mu))
    hes = np.matmul(np.transpose(X),np.matmul(S,X))
    return hes

def normalize(x):
    # función que normaliza el vector x
    norm = x/np.sqrt(sum(x*x))
    return norm
```

Lista de Código 1: Funciones base de los algoritmos

```python
def descent_direction(X, y, beta, method="max",H=None):
    if(method == "max"):
        pk = gradiente_f(X,y,beta)

    elif(method == "newton"):
        grad = gradiente_f(X,y,beta)
        hess = hessiana_f(X,y,beta)
        pk = np.linalg.solve(hess,grad)

    elif(method=="bfsg"):
        pk = np.matmul(H,gradiente_f(X,y,beta))

    return - normalize(pk)
```

Lista de Código 2: Función para especificar el método de descenso de acuerdo al método

```python
def calc_lr(X, y, beta, lr, pk, c1=10**(-4), tao=0.5, reset_lr=False):
    # Calcula la tasa de aprendizaje en cada iteración utilizando conds. de Armijo
    # Inicializamos
    tao = 0.9
    max_iter = 100
    iter = 0
    # Inicializa lr
    if reset_lr==True: lr = 1
    # Evaluaciones periódicas
    grad = gradiente_f(X,y,beta)
    eval_f = f(X,y, beta)
    # Primera iteracion
    f_x =  f(X,y, beta + lr*pk) #en nocedal es phi(alpha)
    f_x1 = eval_f + c1 * lr *  np.dot(grad,pk) # en nocedal es l(alhpa)
    while ((f_x > f_x1) & (iter < max_iter)):
        lr = lr*tao
        f_x =  f(X,y, beta + lr*pk)
        f_x1 = eval_f + c1 * lr *  np.dot(grad,pk)
        iter+=1
    return lr
```

Lista de Código 3: Función para actualizar la tasa de aprendizaje con las condiciones de Armijo

```
def gradient_descent(X, y, lr=1, tol=10**(-7), max_iter=10**5, method="max",
reset_lr=False, verbose_n=1000):
    # Inicializa
    iteraciones=0
    H = None
    dims = X.shape[1]
    tol = tol*dims

    # Inicializamos beta aleatoria
    beta = np.random.normal(1,3,dims)
    if method =="bfsg": H = np.identity(dims)

    # Primera iteracion
    pk =  descent_direction(X, y, beta, method,H)
    beta_new = beta + lr*pk
    if method == "bfsg": H=calc_H(X,y,beta,beta_new,H)

    # Condición de paro.
    while ((np.linalg.norm(gradiente_f(X,y,beta_new)) > tol) & (iteraciones < max_iter)):
        iteraciones+=1 #contador de ciclo

        beta = beta_new
        pk =  descent_direction(X,y,beta,method,H)
        lr = calc_lr(X, y, beta, lr, pk, reset_lr = reset_lr)

        beta_new = beta + lr*pk

        if method == "bfsg": H=calc_H(X,y,beta,beta_new,H)
```

Lista de Código 4: Programa principal de descenso de gradiente