Instituto Tecnológico de Costa Rica
Escuela de Ingeniería Electrónica
Profesor: Dr. Pablo Alvarado Moya
EL 5857 Aprendizaje Automático
I Semestre 2024

Entrega: **21 de marzo, 2024**

Tarea 3

Regresión logística y optimización

I. Introducción

En esta tarea se pone en práctica el primer concepto básico de clasificación revisado en el curso: la regresión logística. Para poder aplicarla es necesario utilizar las técnicas de descenso de gradiente para el proceso de aprendizaje.

La solución de esta tarea requiere conocer y comprender la teoría asociada a los métodos de clasificación y optimización, para poder enfrentar los problemas típicos que aparecen a la hora de su programación.

El objetivo principal de implementar los métodos, es crear una *intuición* del comportamiento de estos métodos "básicos" y sus hiperparámetros, antes de empezar a trabajar con los métodos más complejos en las siguientes tareas y proyectos del curso.

Para probar los métodos se utilizará un conjunto de datos de tres especies de pingüinos en el archipiélago Palmer en la Antártica (figura 1).

Fotógrafo: Liam Quin
Adélie

Fotógrafo: Christopher Michel
Chinstrap

Fotógrafo: Georg Botz
Gentoo

Figura 1: Tres especies de pingüinos con las que se trabajará en las tareas 3 y 4.

Se utilizarán la longitud y profundidad del pico, la longitud de la aleta y la masa de cada espécimen como características de entrada. Las clases a utilizar en los experimentos de clasificación serán el sexo, la isla de proveniencia, o la especie de los pingüinos. Este conjunto de datos de Allison Horst, Alison Presmanes Hill y Kristen Borman es apto para experimentaciones iniciales en estas temáticas de clasificación, y es realista en el sentido de que trae defectos por datos faltantes y algunos datos perdidos (llamados *outliers*) y errores de etiquetación.

La invitación al repositorio de trabajo para esta tarea en el Github Classroom está en este enlace.

II. Procedimiento

II.1. Datos

La función en `loadpenguindata.m` importa los datos en el archivo `penguins_size.csv`, que corresponden al conjunto de datos original, y elimina los puntos inválidos.

Revise con detenimiento cómo funciona este importador de datos. En particular preste atención a cómo se detectan y se eliminan los datos faltantes, pues estrategias similares requerirá usted en puntos posteriores.

Revise además cómo se convierten las etiquetas de cada clase a la correspondiente codificación *one-hot*.

En las siguientes secciones usted realizará varios experimentos, para explorar y reforzar conceptos del aprendizaje automático.

II.2. Optimización

En la programación de los métodos de aprendizaje automático es necesario abstraer con frecuencia funcionalidad y estado de alguna instancia, lo que se implementa naturalmente con el concepto de *clase*, en el paradigma de programación orientada a objetos. Tómese por ejemplo, la clase `normalizer` usada para normalizar las columnas de una matriz. En su estado, esta clase almacena el método y los factores que se utilizan cuando se hace la normalización, de modo que posteriormente se pueda volver a aplicar exactamente el mismo mapeo de normalización a nuevos datos, o se pueda revertir la normalización utilizada.

En esta tarea usted utilizará el concepto de *clases* que utiliza GNU/Octave para aplicar el paradigma de orientación a objetos en su solución.

En esta sección usted deberá completar la implementación de la clase `optimizer`. Se implementa como clase, para poder reutilizar la configuración del optimizador, como el número de iteraciones, el método usado para mostrar el progreso del proceso de descenso de gradiente, la forma de visualizar el progreso del entrenamiento, o los parámetros como tasa de aprendizaje o constantes de filtros.

En ese código se utilizan técnicas de GNU/Octave para manipular los argumentos que el usuario especifica al optimizador de una forma flexible, utilizando pares de parámetros de la forma "`propiedad`", `valor`.

Además, se utiliza el paquete de derivación automática para facilitar el cálculo del gradiente de la función de pérdida. Lamentablemente, dicho paquete tiene algunos errores que pueden producir resultados incorrectos si se utilizan estrategias no vectorizadas en la implementación de las funciones de hipótesis, de pérdida, etc., pues puede inducirse a interrupciones en la cadena de derivadas. Otro problema de dicho paquete es que no se respetan las dimensiones de las matrices usadas en la derivación, lo que obliga a ciertos ajustes manuales. En esta tarea, dichos problemas se subsanan utilizando vectorizaciones estándar.

Revise los conceptos de *constructor* en el paradigma orientado a objetos e identifique el constructor en el código disponible para la clase `optimizer`. Identifique los otros métodos implementados.

Su tarea concreta en este punto es completar la clase para que realice el proceso de descenso de gradiente utilizando descenso por lotes ("`batch`"), y el descenso estocástico de gradiente con minilotes ("`sgd`"). Ya se brinda como ejemplo el método de descenso estocástico con `momentum`, y

usted debe completar el código de la clase para realizar los otros dos métodos (son pocas líneas que debe agregar, pero debe comprender bien ese código). (2 puntos)

El archivo `linreg_main.m` utiliza la clase `optimizer` para resolver un problema sencillo de regresión lineal, de modo que sirva como banco de prueba para el proceso de desarrollo de los métodos de descenso de gradiente faltantes.

II.3. Regresión logística

Para el problema de regresión logística se buscará clasificar el sexo de cada espécimen en el conjunto de datos. Se realizarán experimentos con 2, 3 ó 4 características de entrada, para revisar distintos aspectos de los procesos de aprendizaje.

1. Implemente en el archivo `logreg_hyp.m` la hipótesis $h_{\underline{\boldsymbol{\theta}}}(\underline{\boldsymbol{x}})$ correspondiente a la regresión logística. (2 puntos)

   La función de hipótesis debe recibir un vector columna $\underline{\boldsymbol{\theta}}$ con los parámetros y una matriz de diseño $\boldsymbol{X}$ con un dato en cada una de sus m filas.

   Debe asumir que quien llama a esta función ya se encargó de hacer las modificaciones necesarias en los datos para hacer posible el sesgo, y por tanto el número de columnas de $\boldsymbol{X}$ debe ser igual a la dimensión de $\underline{\boldsymbol{\theta}}$.

   Utilice `assert` para asegurarse que las dimensiones utilizadas son las esperadas, y en caso contrario que el programa termine con un error. También puede hacer el código robusto, en el sentido de que si el usuario entrega un vector fila o columna, internamente utilice un vector columna.

2. Implemente MSE (*mean squared error*) como la función de pérdida (o error) $J(\underline{\boldsymbol{\theta}})$ en el archivo `logreg_loss.m` para la regresión logística. (3 puntos)

   Las funciones de hipótesis y de pérdida reciben como argumentos el vector de parámetros $\underline{\boldsymbol{\theta}}$ (`theta`), los datos de entrada en la matriz de diseño $\boldsymbol{X}$, y las etiquetas de salida $\underline{\boldsymbol{y}}$ con un 1 si el dato correspondiente pertenece a la clase representada o 0 si no.

3. Pruebe sus funciones agregando código en el archivo `logreg_main.m` para hacer un clasificador que use regresión logística, usado las cuatro características disponibles como entrada para estimar el sexo del espécimen.

   Recuerde modificar los datos disponibles para que el modelo pueda utilizar sesgo. Además, recuerde normalizar los datos, particularmente en este caso que cada característica tiene un rango de valores distinto a los demás. (5 puntos)

4. Utilizando las cuatro características disponibles a la entrada, grafique la pérdida con cada método de optimización implementado, en función del número de iteración, como se ilustra en la figura 2. (2 puntos)

   Despliegue los valores de $\underline{\boldsymbol{\theta}}$ obtenidos en cada caso. Razone qué características parecen ser las más relevantes según esos valores. (1 punto)

5. Como *error empírico* se conoce la cuenta total de errores cometidos por un clasificador en una cantidad determinada de experimentos. Se puede calcular cuando se conoce qué es lo que

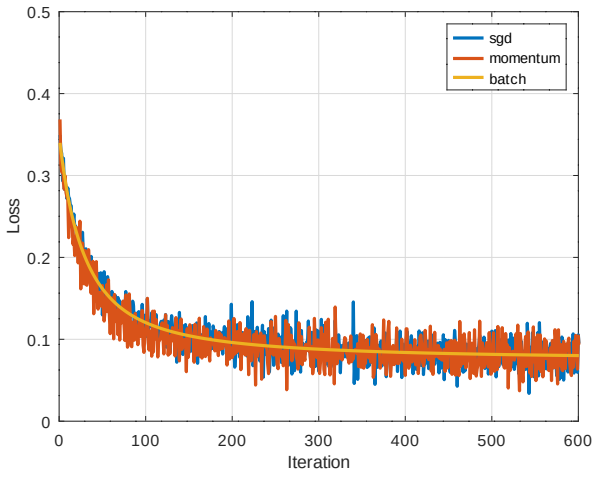

Figura 2: Pérdida en función de la iteración para los cinco métodos de optimización.

el clasificador debió haber producido como salida, es decir, si se conocen datos de referencia, o también llamados datos de verdad básica (en inglés *ground truth*).

Implemente una función para calcular el error empírico si se brindan como argumentos tanto la predicción hecha por el regresor logístico, como las etiquetas con las clases verdaderas. Esta función debe calcular tanto el número total absoluto de errores, como el error porcentual.

(5 puntos)

6. Complete el código en `logreg_main.m` para indicar con cada método de optimización, el error empírico alcanzado, usando por un lado el conjunto de entrenamiento, y por otro lado el conjunto de datos de prueba. (5 puntos)

   Muestre el número total de errores, el número total de experimentos realizados, y qué porcentaje de esos experimentos produjo una clase errónea.

   Por ejemplo:

   ```
   Errores de entrenamiento: 31 de 266 (11.654135%)
   Errores de prueba: 5 de 67 (7.462687%)
   ```

7. En este punto, se realizarán experimentos de clasificación utilizando solo dos características a la vez. Con esto se pretende verificar experimentalmente la observación intuitiva realizada por usted en el punto 4, con respecto a las características más relevantes de acuerdo a $\underline{\boldsymbol{\theta}}$.

   En `logreg_main.m` agregue código que automáticamente entrene el regresor logístico con dos características a la vez. Considerando que la matriz de diseño tiene cinco columnas, la primera correspondiente al término que permitirá el sesgo, y las otras cuatro las características en sí del conjunto de datos, hay entonces 10 posibles combinaciones de las características. Calcule el error empírico para cada caso con el conjunto de prueba Utilice la configuración del regresor que considere conveniente.

   Verifique si el mejor caso corresponde al predicho por usted en el punto 4 para dos características. (5 puntos)

   Para el siguiente punto, utilice las dos mejores características.

8. El caso con solo dos características permite graficar una superficie correspondiente a la salida del clasificador, correspondiente a $p(y = 1|\underline{\boldsymbol{x}})$.

   Agregue código en `logreg_main.m` para visualizar esa superficie, de modo similar al ilustrado en la figura 3. (3 puntos)

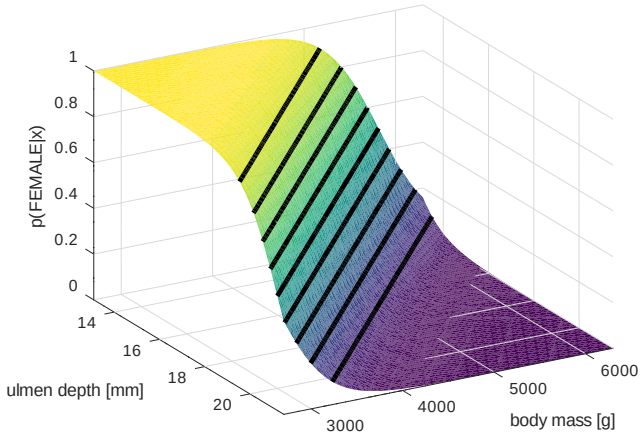

Figura 3: Superficie de salida $p(y|\underline{\boldsymbol{x}})$ en función de las dos características de entrada.

   Debe tener cuidado para poder utilizar la escala original de las características, a pesar de que el clasificador usó el conjunto de entrenamiento normalizado.

9. Este caso con solo dos características, permite ilustrar el concepto de frontera de decisión, es decir, el límite entre la región del espacio de entrada que el clasificador asigna a la clase $y = 1$, y la región asignada a la clase $y = 0$.

   La función `contour` de GNU/Octave permite graficar curvas de nivel de superficies, lo que puede usarse para dibujar dicha frontera de decisión.

   Muestre entonces en una nueva figura los puntos de prueba y la frontera de decisión, de forma similar a lo ilustrado en la figura 4. (3 puntos)

10. Ahora se realizará otro experimento más utilizando 3 características.

    Seleccione otra característica adicional a las dos utilizadas en los puntos anteriores. Justifique su elección. (1 puntos)

11. Entrene el regresor logístico con esas tres características. Visualice la trayectoria seguida en el espacio paramétrico donde *vive* $\underline{\boldsymbol{\theta}}$ para tres de los métodos de optimización, de forma similar a lo que se ilustra en la figura 5. (3 puntos)

12. Puntos extra. Agregue un archivo `logreg_main2.m` con la funcionalidad de los puntos 8 y 9, pero modificados de tal forma que que permita realizar fronteras de división no lineales. Puede utilizar los desarrollos hechos por usted en la tarea 2. (4 puntos)

Entregable: archivos de GNU/Octave, y README con instrucciones de cómo ejecutar el código. Debe mostrarse cada punto solicitado en figuras aparte, y en la salida de texto debe indicarse claramente los resultados solicitados o el paso calculado en el momento. Evite que su programa se

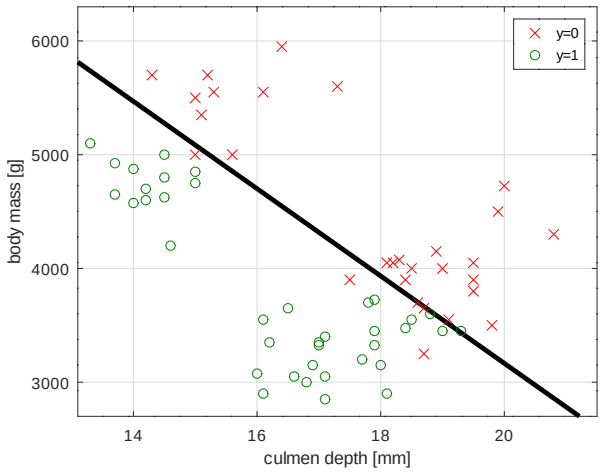

Figura 4: Frontera de decisión y datos de prueba.

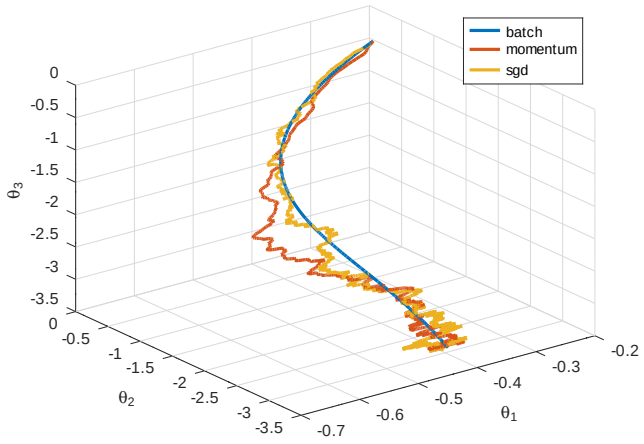

Figura 5: Trayectoria de los parámetros durante el entrenamiento para tres métodos de optimización.

detenga para preguntar algo al usuario. Toda opción debe ser configurada ya sea en el código o como argumentos cuando se llame al script principal.

Esta tarea se realiza en parejas o individual. Informar al profesor si el grupo de trabajo en esta tarea cambió respecto al de la tarea 2.

Solo lo que sea subido al tecDigital será calificado. Entregas por correo serán ignoradas, por lo que asegúrese de haber enviado correcciones a su grupo (aún si es individual) a tiempo.

Esta tarea requiere tiempo. Evite dejarla para los últimos días.

Se revisará el trabajo individual por medio de la actividad en Git.

Toda tarea debe ser resultado del trabajo intelectual propio de la persona o personas que la entregan. Además de la literatura de referencia, solo puede utilizarse el material expresamente así indicado en la tarea, lo que incluye código brindado por el profesor o indicado en los enunciados a ser utilizado como base de la tarea. Expresamente quedan excluidos como material de referencia los trabajos entregados por otros estudiantes en el mismo semestre o semestres anteriores.

Nótese que esto no elimina la posibilidad de discutir estrategias de solución o ideas entre personas y grupos, lo cual es incluso recomendado, pero la generación concreta de cada solución, derivación o programa debe hacerse para cada entrega de forma independiente.

Para toda referencia de código o bibliografía externa deben respetarse los derechos de autor, indicando expresamente de dónde se tomó código, derivaciones, etc. Obsérvese que el código entregado por el profesor usualmente ya incluye encabezados con la autoría correspondiente. Si un estudiante agrega código a un archivo, debe agregar su nombre a los encabezados si la modificación es de más del 50 % del archivo, o indicar expresamente en el código, con comentarios claros, la autoría de las nuevas líneas de código, pues a la autora o al autor del archivo original no se le debe atribuir código que no es suyo.

Si se detecta código o deducciones teóricas iguales o muy cercanas a trabajos de otros estudiantes del mismo semestre o de semestres anteriores, se aplicará lo establecido por la reglamentación vigente, en particular el Artículo 75 del Reglamento de Régimen de Enseñanza y Aprendizaje.

Modificaciones de comentarios, cadenas alfanuméricas, nombres de variables, orden de estructuras independientes, y otras modificaciones menores de código se siguen considerando como clones de código, y las herramientas automatizadas de detección reportarán la similitud correspondiente.

Los estudiantes que provean a otros estudiantes del mismo o futuros semestres soluciones de sus tareas, también son sujetos a las sanciones especificadas en la reglamentación institucional. Por lo tanto, se advierte no poner a disposición soluciones de las tareas a otros estudiantes.