Práctica de Laboratorio Regresión Logística CComp9-1

Ricardo Manuel Lazo Vásquez
ricardo.lazo@ucsp.edu.pe

Universidad Católica San Pablo

1 Introducción

En el presente informe se procederá a explicar la implementación de Regresión Logística implementada con k-folds de 3 y con validación cruzada. El informe está dividido de la siguiente manera: En la Sección 2 se describira la organización del github de la implementación así como las funciones a implementar, en la Sección 3 se explicarán los experimentos implementados, así como la organización de los resultados obtenidos para su replicación; finalmente en la Sección 4 se darán las conclusiones.

2 Implementación

El código se encuentra disponible en https://github.com/TheReverseWasp/TIA-Lab-Regresion_Logistica y se organiza en las siguientes carpetas que se muestran en la Tabla 1

Carpeta	Descripción
./Datos	Carpeta donde se ubican los dos datasets a utilizar: diabetes.csv y Enfermedad_Cardiaca.csv.
./Implementacion	Carpeta donde se ubican los scripts de la implementación.
./Practica	Carpeta donde se ubican el pdf con las especificaciones para este trabajo.
./Resultados	Carpeta donde se almacenan los resultados mostrados en este informe.

Table 1: Carpetas del Github.

La organización de la carpeta de Resultados es importante en caso de replicar el repositorio. El principal problema que se puede encontrar es si esta carpeta o las carpetas que contiene no existen, es por eso que se pide que estén creadas de antemano. En caso de tener experimentos grabados previamente en estas carpetas simplemente se sustituyen cuando se corra main.py

Las carpetas que deben estar presentes dentro de la carpeta de Resultados son "Exp1/" y "Exp2/". Y dentro de "Exp1/" deben existir las siguientes carpetas "train/" y "test/".

Los scripts dentro de la carpeta implementación están en la Tabla 2, donde se explica los detalles de cada archivo de manera general.

Script .py	**Detalles**
complementos.py	Funciones complementarias para la librería my_lib.py
my_lib.py	Script con las funciones a evaluar en esta implementación.
experimentos.py	Implementación de los dos experimentos a evaluar.
main.py	Archivo donde se lanza el menú de selección de experimentos
cross_calc.py	Script semimanual para hallar la validación cruzada.

Table 2: Scripts de implementación.

la ejecución para replicar los experimentos se da con la siguiente linea de código:

```
python3 main.py
```

Donde mostrará un menú de selección para ejecutar los dos experimentos de manera independiente y guardar los resultados de manera automática una vez finalizados.

2.1 Detalles de la implementación

La implementación requerida de la práctica ubicada en "./Practica/" especifica que la implementación debe tener las funciones de la Tabla 3, todas ellas están implementadas en el script "my_lib.py".

2.2 Descripción de k-folds

Para esta implementación se pidió que el número de folds sea igual a tres, pero la implementación puede trabajar con un k mayor sin problemas.

La manera de utilizar los k-folds será con validación cruzada, en ella se permutan los folds seleccionando los dos primeros para entrenamiento y uno para prueba. Finalmente se promediaron los índices de accuracy para obtener un índice intermedio entre cada conjunto de datos.

Para esta tarea se separó los conjuntos en la carpeta "./Resultados/Exp1/train" y "./Resultados/Exp1/test" por cada dataset y por cada permutación y se promedió en un script de manera manual en un script externo a este proyecto. Esto se realizó de esta manera para no perder los datos de las permutaciones en caso de una verificación externa.

Las permutaciones serian 6 para un 3 folds, estas también se muestran en la ejecución y serían las mostradas en la Tabla 4:

Función	Descripción
Leer_Datos	Recibe el nombre del archivo y devuelve un *numpy_array*
Normalizar_Datos	Normaliza los datos de un *numpy_array* hacia un intervalo más pequeño (Usualmente de -1 a 1).
Sigmoidal	Que recibe un X y un θ y calcula la función sigmoidal (tambien conocida como función logistica).
Calcular_Funcion_Costo	Que recibe un X, un θ y un Y. Calcula el costo por entropia cruzada.
Calcular_Gradiente	Que recibe un X, un θ y un Y. Devuelve la gradiente asociado a X e Y.
Gradiente_Descendiente	Que recibe un X, un θ y un Y; ademas del número de iteraciones, y el índice de aprendizaje. Devuelve los θ actualizados y el costo por iteración.
Calcular_Accuracy	Que recibe un X, un θ y un Y. Devuelve el accuracy asociado a X e Y.
Crear_k_folds	Recibe un conjunto normalizado y lo divide en k subconjuntos disjuntos del conjunto de entrada. Donde cada subconjunto debe tener la misma proporción del conjunto base pero en menor número.

Table 3: Descripción de las funciones implementadas en my_lib.py descritas en los objetivos de la practica.

Nro. de Permutación	Orden de la Permutación
1	[0, 1, 2]
2	[0, 2, 1]
3	[1, 0, 2]
4	[1, 2, 0]
5	[2, 0, 1]
6	[2, 1, 0]

Table 4: Orden de los folds por permutación

3 Experimentos y Resultados

3.1 Descripción de los experimentos

En la práctica para esta implementación se solicitan dos experimentos descritos en la Tabla 5, ambos están implementados en el script "experimentos.py" y se ejecután automaticamente al ejecutar "main.py".

3.2 Descripción de las funciones de Experimentos

En el caso del primer experimento, este almacena sus resultados en la carpeta "./Resultados/Exp1/" separados en las carpetas "train" y "test". Para obtener

Experimentó	Descripción
Experimento_1	Que debe retornar una matriz de accuracy de cada dataset, tanto para sus datos de entrenamiento como de prueba.
Experimento_2	Plotear el histórico de costos de cada dataset.

Table 5: Descripción de los experimentos implementados en "experimentos.py" y solicitados en la practica de "./Practica/".

la validación cruzada se utiliza el script "cross_calc.py" donde se insertan las matrices de resultados de las 6 permutaciones manualmente obteniendo el resultado observado en los Resultados.

El experimento 2 tras su ejecución almacena los resultados en la carpeta "./Resultados/Exp2/" con el siguiente formato: "(número de dataset)-(indice de aprendizaje)-p(número de permutación).jpg". Para los resultados mostrados en la Sección de Experimentos se usaron los dos datasets en el índice de aprendizaje 0.3 por razones que se explicara en la Sección Resultados.

3.3 Resultados

Experimento 1 Para el experimento 1 se realizó el cálculo del accuracy por cada permutación de un 3-folds, siendo un total de 6 permutaciones.

Se separaron los datos por entrenamiento y prueba así como por el dataset al que pertenecen, los resultados pueden observarse en las tablas 6 y 7 para el dataset de diabetes y las tablas 8 y 9 para el dataset de enfermedades cardiacas.

	0.01	**0.05**	**0.1**	**0.2**	**0.3**	**0.4**
500	0.6510	0.6556	0.6960	0.7409	0.7526	0.7630
1000	0.6510	0.6960	0.7409	0.7630	0.7689	0.7682
1500	0.6510	0.7278	0.7526	0.7689	0.7676	0.7715
2000	0.6523	0.7409	0.7630	0.7682	0.7715	0.7728
2500	0.6556	0.7480	0.7676	0.7695	0.7721	0.7754
3000	0.6634	0.7526	0.7689	0.7708	0.7741	0.7754
3500	0.6777	0.7584	0.7682	0.7715	0.7760	0.7754

Table 6: Accuracy en la data de diabetes con la data de entrenamiento

En el caso del primer dataset (diabetes), se puede observar en las tablas 6 y 7 que los mejores hiperparametros serian un índice de aprendizaje entre 0.3 y 0.4 y una cantidad de iteraciones entre 3000 y 3500 donde obtiene mejor rendimiento tanto en los datos de entrenamiento como en los de prueba.

Por otra parte en el segundo dataset (enfermedad cardiaca), se puede notar en las tablas 8 y 9 que los mejores hiperparametros serían un índice de aprendizaje

	0.01	**0.05**	**0.1**	**0.2**	**0.3**	**0.4**
500	0.651	0.6537	0.6979	0.7291	0.7473	0.7551
1000	0.651	0.6979	0.7291	0.7551	0.7642	0.7629
1500	0.651	0.7278	0.7473	0.7642	0.759	0.7642
2000	0.651	0.7291	0.7551	0.7629	0.7642	0.7642
2500	0.6537	0.7356	0.7603	0.7629	0.7642	0.7694
3000	0.6589	0.7473	0.7642	0.7642	0.7668	0.7681
3500	0.668	0.7564	0.7629	0.7681	0.7694	0.7681

Table 7: Accuracy en la data de diabetes con la data de prueba

	0.01	**0.05**	**0.1**	**0.2**	**0.3**	**0.4**
500	0.802	0.8119	0.8317	0.8432	0.8548	0.8564
1000	0.8069	0.8317	0.8432	0.8564	0.8564	0.8564
1500	0.8119	0.8432	0.8548	0.8564	0.8564	0.8597
2000	0.8119	0.8432	0.8564	0.8564	0.8597	0.8614
2500	0.8119	0.8515	0.8597	0.8564	0.8614	0.8614
3000	0.8168	0.8548	0.8564	0.8597	0.863	0.8614
3500	0.8234	0.8548	0.8564	0.8614	0.863	0.8614

Table 8: Accuracy en la data de enfermedad cardiaca con la data de entrenamiento

	0.01	**0.05**	**0.1**	**0.2**	**0.3**	**0.4**
500	0.7921	0.8152	0.802	0.8185	0.8218	0.8251
1000	0.802	0.802	0.8185	0.8251	0.835	0.8317
1500	0.8053	0.8086	0.8218	0.835	0.835	0.8416
2000	0.8119	0.8185	0.8251	0.8317	0.8416	0.8383
2500	0.8152	0.8218	0.8317	0.8383	0.8383	0.835
3000	0.8152	0.8218	0.835	0.8416	0.835	0.8383
3500	0.8185	0.8218	0.835	0.8416	0.835	0.8383

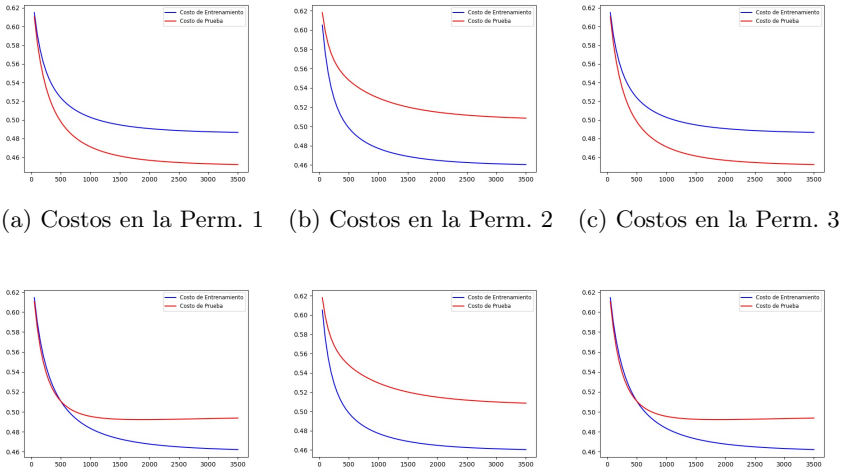
Table 9: Accuracy en la data de enfermedad cardiaca con la data de prueba

entre 0.1 y 0.3 y un número de iteraciones entre 3000 y 3500, pero se debe destacar que se tiene un rendimiento aceptable en estos índices de aprendizaje desde la iteración 500.

Experimento 2 En el caso del segundo experimento se obtuvo resultados para cada lista de hiperparametros, pero se resaltaron los más simbólicos en accuracy del Experimento 1.

En el caso del primer dataset (diabetes) los resultados estarán visibles en la Figura 1, donde se puede observar que en algunas permutaciones como la 2, 4,

5 y 6 la función de costo de la parte de prueba es menor a la de entrenamiento, en el resto la función tiene un comportamiento normal.



(a) Costos en la Perm. 1 (b) Costos en la Perm. 2 (c) Costos en la Perm. 3

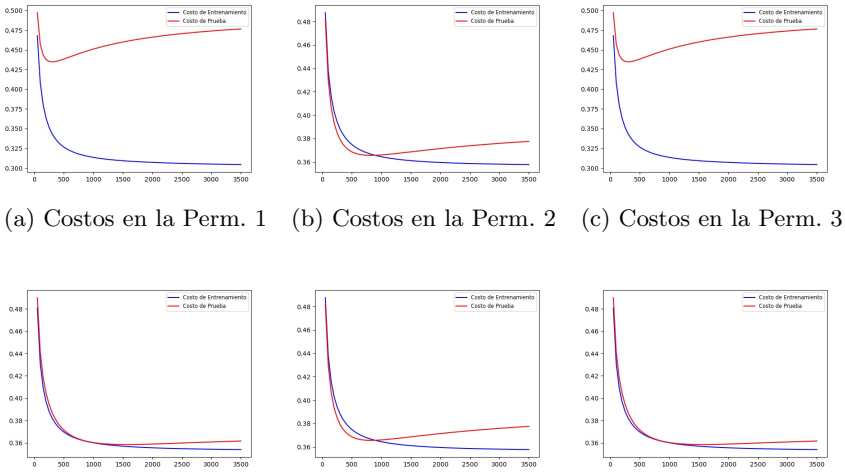
(d) Costos en la Perm. 4 (e) Costos en la Perm. 5 (f) Costos en la Perm. 6

Fig. 1: Resultados de la función costo entrenamiento (azul) con la de prueba (rojo) con indice de aprendizaje 0.3 para el dataset de diabetes.

Por otra parte en el caso del segundo dataset (enfermedades cardiacas) los resultados de la función costo tanto para entrenamiento como para prueba pueden observarse en la Figura 2 donde se puede observar que con algunas permutaciones se tiene un overfitting temprano, en el caso de la Permutación 2 y 5 se puede observar que seria una buena estrategia aplicar Early stopping alrededor de la iteración 1500 y finalmente en el resto de permutaciones se ve un comportamiento normal.

4 Conclusiones

En el presente informe se describió la implementación de regresión logística, en ella se pudo notar que la elección de hiperparametros es importante pero que también el uso de estrategias como Early Stopping puede ser de mucha ayuda al momento de obtener mejores resultados; se vio también cómo usar diferentes distribuciones de datos de entrenamiento (k-folds) puede afectar de tanto el accuracy como el costo de nuestra implementación.



(a) Costos en la Perm. 1 (b) Costos en la Perm. 2 (c) Costos en la Perm. 3

(d) Costos en la Perm. 4 (e) Costos en la Perm. 5 (f) Costos en la Perm. 6

Fig. 2: Resultados de la función costo entrenamiento (azul) con la de prueba (rojo) con indice de aprendizaje 0.3 para el dataset de enfermedades cardiacas.

References

1. Andrew Ng. Specialization in Deep Learning.
`https://www.coursera.org/specializations/deep-learning`