Regresión Logística

Shelsy Natalia Rodríguez Barajas

Ciencias de la computación e inteligencia artificial
HPC 2 – Métricas de rendimiento
John Corredor Franco
Universidad Sergio Arboleda
2022-02

UNIVERSIDAD SERGIO ARBOLEDA

1. Introducción

En este trabajo se pueden analizar los resultados de la implementación de la regresión logística con dos métodos de trabajo, se espera que los resultados de los dos modelos sean lo mas similares posibles y de lo contrario se tratara de comprender a que corresponde la diferencia de los resultados

2. Objetivos

2.1 Objetivo General

El objetivo general de este trabajo es modelar las predicciones basadas en la Regresión Logística en dos entornos diferentes, en primer lugar, se quiere modelar con ayuda del modulo scikit-learn de Python y en segundo lugar modelarla en el lenguaje de programación c++ por medio del framework Qt creator. Para finalmente comparar los resultados, tiempo de ejecución y otros factores que se consideren importante en el proceso de la implementación

2.2 Objetivos Específicos

- Seleccionar un dataset
- Hacer una analítica de datos sobre el dataset seleccionado
- Modelar usando la regresión logística usando: Python, Scikit-Learn
- Modelar usando la regresión logística usando: C++, Eigen, Boost
- Comparar los resultados obtenidos del modelo

3. Desarrollo

En este ítem están especificados los pasos que se siguieron para el desarrollo de la implementación de los modelos, en cada numeral se abordan las características mas importantes y las decisiones que se tomaron para la elección.

3.1 Dataset

El dataset seleccionado para esta implementación es sobre el cáncer de mama de Wisconsin, cuenta con 33 atributos que van desde el ID del paciente hasta datos específicos de las masas mamaria como lo son el radio, la textura el perímetro, simetría, área entre otras. En los cuales 32 son numéricos y uno solo es categórico y es correspondiente a la característica objetivo (Diagnostico).

Las características se calculan a partir de una imagen digitalizada de una aspiración con

aguja fina (FNA) de una masa mamaria. Describen características de los núcleos celulares presentes en la imagen.

Adicionalmente a este dataset original se decidió crear otro dataset a partir de las características mas importantes de este, el nuevo dataset cuenta solo con 10 características importantes en formato numérico y en la posición numero 11 se encuentra la variable objetivo también en formato numérico

3.2 EDA

En este paso realizamos un exhaustivo análisis del dataset, iniciamos con un .shape para conocer el tamaño del dataframe

```
# Se visualiza el tamaño del dataset
df.shape
```

```
(569, 33)
```

Se puede observar que hay 569 registros y 33 características. Adicionalmente con la función .info podemos observar el tipo de variable de cada característica y s hay valores nulos y con el .describe una pequeña descripción de cada variable, la cantidad de valores, el promedio, la desviación estándar, los cuartiles, el mínimo y el máximo

```
df.describe().T
```

	count	mean	std	min	25%	50%	75%	max
id	569.0	3.037183e+07	1.250206e+08	8670.000000	869218.000000	906024.000000	8.813129e+06	9.113205e+08
radius_mean	569.0	1.412729e+01	3.524049e+00	6.981000	11.700000	13.370000	1.578000e+01	2.811000e+01
texture_mean	569.0	1.928965e+01	4.301036e+00	9.710000	16.170000	18.840000	2.180000e+01	3.928000e+01
perimeter_mean	569.0	9.196903e+01	2.429898e+01	43.790000	75.170000	86.240000	1.041000e+02	1.885000e+02
area_mean	569.0	6.548891e+02	3.519141e+02	143.500000	420.300000	551.100000	7.827000e+02	2.501000e+03

```
df.info()
```

```
RangeIndex: 569 entries, 0 to 568
Data columns (total 33 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   id                       569 non-null    int64  
 1   diagnosis                569 non-null    object 
 2   radius_mean              569 non-null    float64
 3   texture_mean             569 non-null    float64
 4   perimeter_mean           569 non-null    float64
 5   area_mean                569 non-null    float64
 6   smoothness_mean          569 non-null    float64
 7   compactness_mean         569 non-null    float64
```

3.3 Graficas, estadísticas y relación entre variables

Histograma para visualizar la distribución de los datos

```
df.hist(figsize=(20,20))
plt.show()
```

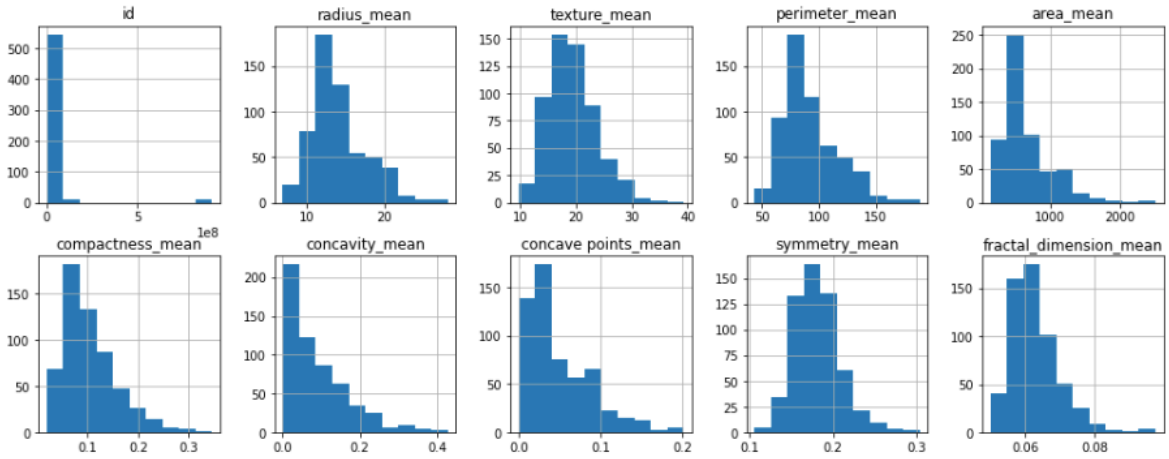
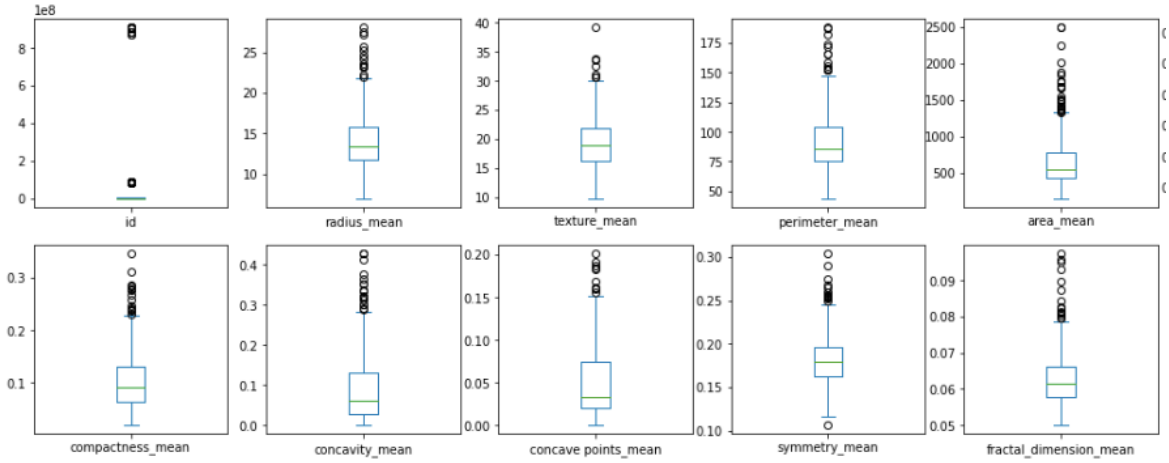


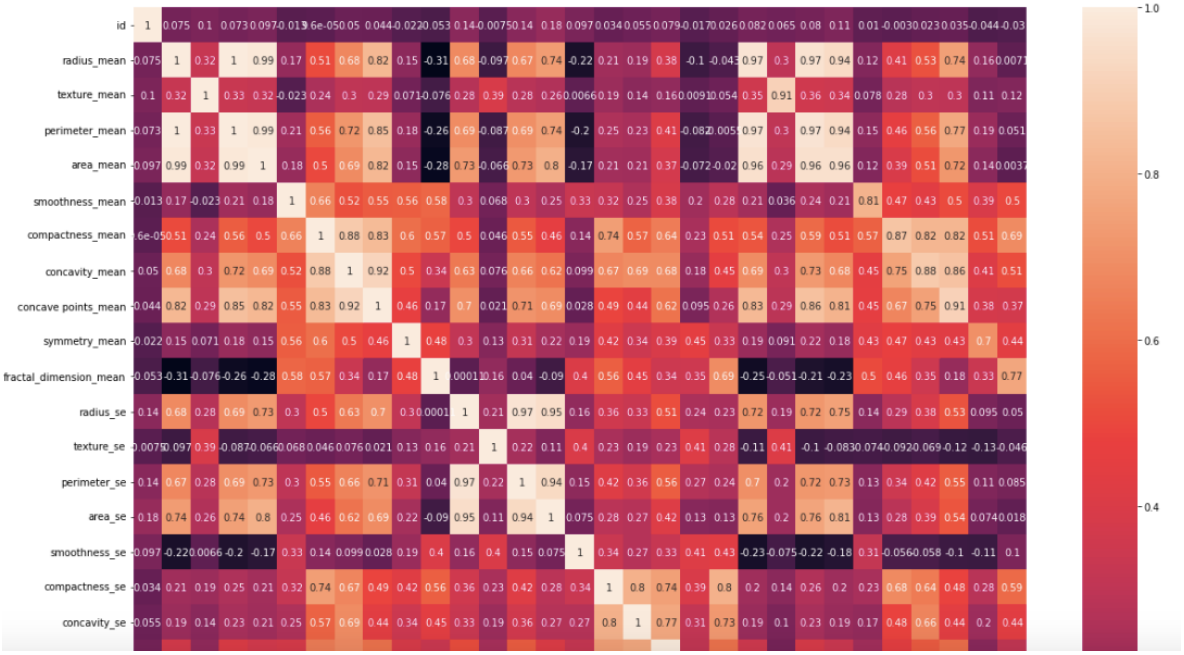
Grafico de cajas y bigotes para identificar los outliers (valores por fuera de la media, por fuera de la distribución)

```
df.plot(kind='box', subplots=True, layout=(6,6), sharex=False, sharey=False, figsize=(20,20))
plt.show()
```



Para ver la relación de las variables se hace la matriz de correlación con mapa de calor, en la que se puede observar la que tanta dependencia hay entre las características del modelo

```
corr = df.corr()
plt.figure(figsize=(20,20))
sns.heatmap(corr, annot=True)
plt.show()
```



Se puede observar que hay características con alta correlación entre si, esto significa que son variables significativas para entrenar el modelo, y son las que mas debemos tener en cuenta, si tomamos en cuenta variables con poca relación va a bajar la precisión del modelo.

3.4 Pipeline de procesamiento

El pipeline de procesamiento es una serie de elementos de procesamiento de datos ordenados de tal modo que la salida de cada uno es la entrada del siguiente, como quien dice una cadena de montaje, pero en vez de orientada a la manufactura, orientada al procesamiento de datos e instrucciones.
Se hace con el fin de preparar los datos para posteriormente con estos entrenar el modelo.

3.5 Métricas de rendimiento

Las métricas de rendimiento usadas para evaluar la calidad del modelo fuero realizadas a partir de la matriz de confusión es una herramienta que permite visualizar el desempeño de un algoritmo Cada columna de la matriz representa el número de predicciones de cada clase, mientras que cada fila representa a las instancias en la clase real., o sea en términos prácticos nos permite ver qué tipos de aciertos y errores está teniendo nuestro modelo a la hora de pasar por el proceso de aprendizaje con los datos.

Las métricas que realizamos fueron:

- Accuracy (Exactitud) que es la cantidad de predicciones positivas que fueron correctas.

- Precisión (Precision) se representa por la proporción de verdaderos positivos dividido entre todos los resultados positivos (tanto verdaderos positivos, como falsos positivos). En otras palabras es el porcentaje de casos positivos detectados.

- Recall (Sensibilidad) Es la proporción de casos positivos que fueron correctamente identificadas por el algoritmo.

- F1 esta es otra métrica muy empleada porque nos resume la precisión y sensibilidad en una sola métrica

3.6 Modelo con Scikit-Learn

Para hacer el modelo de regresión se uso el modulo de LogisticRegression Sklearn de Python, después de dividir los datos en entrenamiento y prueba se entrena el modelo de la siguiente manera

```
# Entrenamiento del modelo de regresion logistica
logreg = LogisticRegression()
logreg.fit(X_train, y_train)
```

Y posteriormente al entrenamiento se predice la variable objetivo

```
# predice la variable objetivo
y_pred = logreg.predict(X_test)
```

Para hacer uso del modulo se debe importar al inicio con el siguiente comando “`from sklearn.linear_model import LogisticRegression`”

3.7 Decisiones

Se decide hacer un nuevo dataset apartir del original para entrenar nuevamente el modelo, pues la calidad del primer modelo era bastante baja, en el nuevo df se ubica la variable objetivo en la ultima columna, a diferencia del original que se encontraba en la segunda columna

3.8 Modelo con C++

Para realizar el modelo de regresión en c++ se hace mediante el framework Qt Creator, acá hacemos uso de la biblioteca de alto rendimiento Eigen, esto nos proporciona una gran ventaja ya que Admite todos los tamaños de matriz, desde pequeñas matrices de tamaño fijo hasta matrices densas arbitrariamente grandes e incluso matrices dispersas. Adicional a esto Admite todos los tipos numéricos estándar, incluidos std::complex, integers.

En esta implementación tenemos las dos clases, la logística y la extracción, de las cuales cada una consta con su header y su source que son los que contienen as funciones.

En la clase header tenemos las funciones para extraer los datos del dataset, crear las filas, columnas, convertirlos de CVS a Eigen Matrix y en el source todos los pasos para hacer la regresión logística

3.9 Comparación de los modelos

- Resultados de la implementación con Scikit-Learn

Cantidad de filas y columnas:

```
# Se visualiza el tamaño del dataset
df.shape
```

```
(569, 33)
```

Métricas de rendimiento:

```
# calcula accuracy
print(metrics.accuracy_score(y_test, y_pred))
```

```
0.9473684210526315
```

```
# calcula la precision
precision_score(y_test, y_pred, average='macro')
```

```
0.9430438842203548
```

```
# calcula el recall
recall_score(y_test, y_pred, average='macro')
```

```
0.9520482692918386
```

```
# calcula el f1-score
f1_score(y_test, y_pred, average='macro')
```

```
0.9463108320251177
```

Tamaño del train y del test:

```
# Se visualiza el tamaño de los datos de entrenamiento y prueba
print('Tamaño de X_train: ',X_train.shape)
print('Tamaño de X_test: ',X_test.shape)
print('Tamaño de y_train: ',y_train.shape)
print('Tamaño de y_test: ',y_test.shape)
```

```
Tamaño de X_train:  (455, 10)
Tamaño de X_test:  (114, 10)
Tamaño de y_train:  (455,)
Tamaño de y_test:  (114,)
```

- Resultados de la implementación del modelo con C++

3.10 Análisis

- En el tiempo de ejecución de los dos modelos podemos encontrar que el modelo de c++ es mas lento, ya que este actualiza el descenso del gradiente para calcular los pesos y así disminuir el error

- Se puede comprobar que el modelo que se realizo con Python es un poco mejor al momento de evaluar métricas de rendimiento

3.11 Recomendaciones

Incluir variables que tengan relación en el modelo, pero no únicamente incluir esas ya que esto podría sesgar nuestro modelo y al momento de realizar el test no mostraría la misma calidad que mostro en el train

4. Conclusiones

4.1 Conclusiones generales

- El modelo inicial que fue entrenado con todas las variables del dataframe no era el mas optimo, por lo que se optó por seleccionar únicamente las variables mas influyentes (con unas mayor correlación) y este segundo modelo, dio mejores resultados, con un acuraccy casi del doble de valor, con esto se puede concluir

que para tener un mejor modelo es importante entrenarlo con variables importantes, con altos pesos que en lugar entrenarlo con una alta cantidad de características que no representan nada

- La distancia a la que se va a desplazar el algoritmo (learning rate) en cada iteración de búsqueda es muy importante debido a que si es muy pequeño, se tardará demasiado en llegar al mínimo y, si es demasiado grande, el algoritmo saltará de una región a otra pasando por encima del mínimo sin alcanzarlo.

- La presicion y el acuraccy del modelo son altos, lo que representa que el modelo de regresión logística es aceptable y se ajusta

4.2 Conclusiones especificas

- Seleccionar un dataset

Del dataset podemos ver que tenemos la gran ventaja de que la mayoría de los datos son numéricos, no se presentan datos nulos, por lo que arreglar el df para el entrenamiento y test no será un proceso muy complicado

No hay una diferencia importante en Python en cuento al lugar de la variable objetivo del dataframe pues al momento de dividir en train y test se reacomoda el orden de las variables

- Hacer una analítica de datos sobre el dataset seleccionado

El dataset presenta una distribución normal de los valores, en c++ es un poco mas difícil realizar el análisis exploratorio de los datos pues no se cuenta con funciones que faciliten en EDA como en python el .describe() .info() .hist(), sino hay que crear manualmente la funcion y luego llamarla desde la clase principal

- Modelar usando la regresión logística usando: Python, Scikit-Learn

Es mas sencillo realizar la implementación con el modulo logisticRegression de Sklearn, ya que es mucho mas fácil pues no hay que crear funciones, únicamente pasar los parámetros que deseemos

- Modelar usando la regresión logística usando: C++, Eigen, Boost

Es un proceso un poco mas largo comparado con el realizado con Python, pero este modelo nos permite aprender el funcionamiento completo de la regresión logística, ya que lo hacemos de manera manual función por función

- Comparar los resultados obtenidos del modelo

El modelo inicial que fue entrenado con todas las variables del dataframe no era el mas optimo, por lo que se optó por seleccionar únicamente las variables mas influyentes (con

unas mayor correlación) y este segundo modelo, dio mejores resultados, con un acuraccy casi del doble de valor, con esto se puede concluir que para tener un mejor modelo es importante entrenarlo con variables importantes, con altos pesos que en lugar entrenarlo con una alta cantidad de características que no representan nada

En el tiempo de ejecución de los dos modelos podemos encontrar que el modelo de c++ es mas lento, ya que este actualiza el descenso del gradiente para calcular los pesos y así disminuir el error

sssssssss

5. Referencias

- https://scikitlearn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
- https://eigen.tuxfamily.org/index.php?title=Main_Page
- https://www.kaggle.com/datasets/uciml/breast-cancer-wisconsin-data