Práctica de Aprendizaje Automático

Predicción de proteínas de un alimento en base a sus nutrientes

2022 - 2023

Marc Nebot Moyano

Alejandro Salvat Navarro

Índice

Introducción

El principal objetivo de nuestro proyecto es conseguir predecir la cantidad de proteína de un alimento en base a sus nutrientes como el hierro, el azúcar, etc. En un principio íbamos a intentar predecir la cantidad de calorías de un alimento, puesto que esto es una simple fórmula todos nuestros resultados daban un 100% de acierto y, por lo tanto, no estábamos haciendo ningún tipo de predicción.

En primer lugar, antes de entrenar los datos, hemos optado por realizar un estudio estadístico de nuestras variables para observar a qué tipo de problema nos estábamos enfrentando y así decidir con qué variables trabajaríamos y cuáles serían desechadas.

En segundo lugar hemos preprocesado los datos una vez finalizado el estudio estadístico de nuestras variables. Para ello hemos eliminado aquellas variables redundantes o que simplemente eran irrelevantes para nuestra predicción de datos. Seguidamente hemos identificado aquellas filas con valores incorrectos, ya que en nuestros datos habían valores que no tenían sentido o que eran desproporcionadamente altos.

Finalmente hemos utilizado modelos lineales y no lineales para comprobar con qué clase de modelos y con qué modelo funciona mejor nuestra base de datos.

Trabajos previos

Antes de realizar el trabajo, hemos hecho un estudio previo para ver qué trabajos se habían realizado en nuestro dataset. Hemos observado que no había ningún tipo de trabajo hecho en este dataset, por lo tanto, seremos los primeros en realizarlo. Sin embargo, buscamos qué tipo de predicciones podríamos hacer en base a los nutrientes que teníamos, es por este motivo que descubrimos que la predicción de calorías no la podíamos realizar, pues era una simple fórmula la que se aplicaba.

Por otro lado, hemos decidido investigar una vez finalizado el trabajo para no vernos influenciados qué tipos de estudios se han realizado en base a los nutrientes de los alimentos y hemos observado que había predicciones de azúcares, grasas e incluso proteínas como nosotros.

Exploración de datos

Al cargar nuestro dataset, observamos que este, inicialmente, presenta 9318 instancias con 29 características por elemento.

Columnas Dataset Original

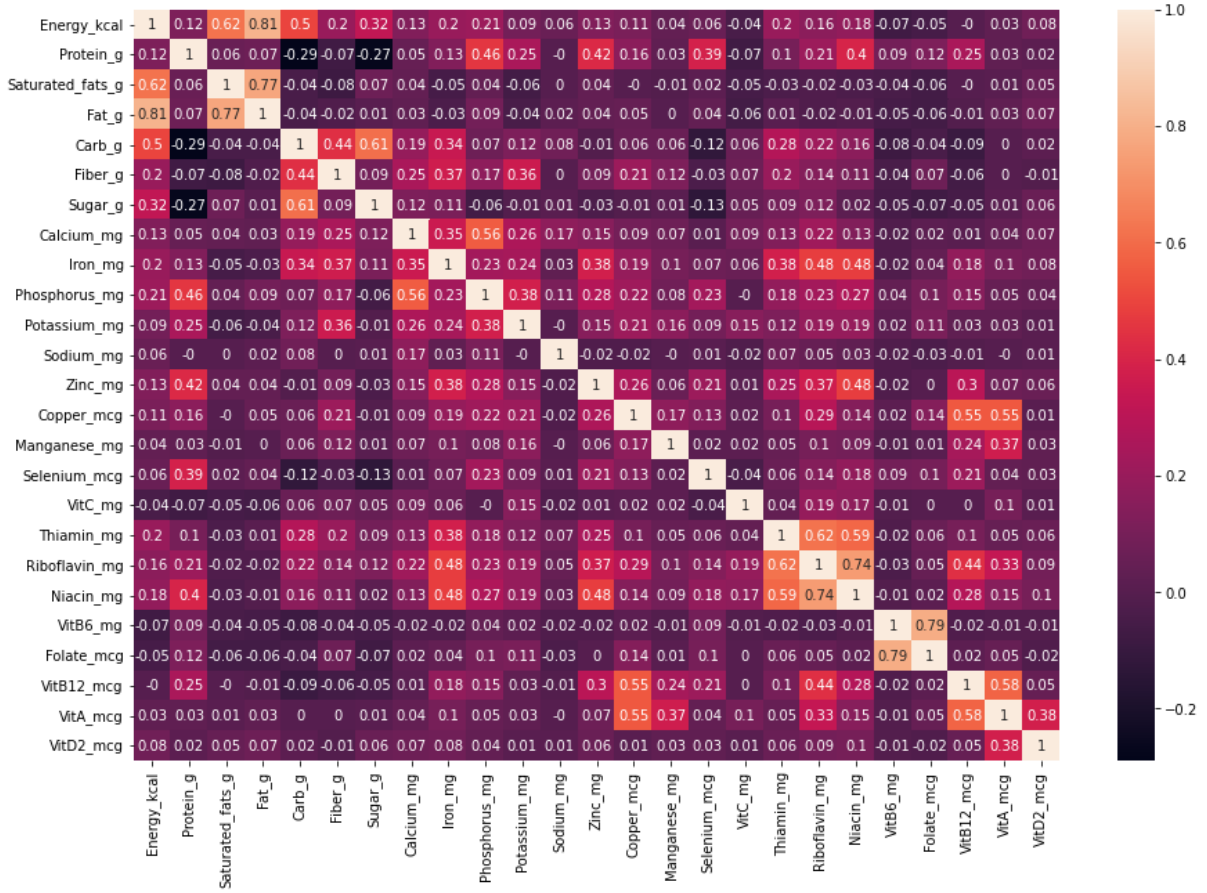
Nuestro dataset cuenta con las siguientes características iniciales (columnas):

- **NDB_No:** Número en la base de datos nacional de nutrientes
- **Descrip:** Breve descripción del alimento
- **Energy_kcal:** Energía del alimento en kilocalorías
- **Protein_g:** Gramos de proteína en el alimento
- **Satured_fats_g:** Gramos de grasas saturadas en el alimento
- **Fat_g:** Gramos de grasas en el alimento
- **Carb_g:** Gramos de carbohidratos en el alimento
- **Fiber_g:** Gramos de fibra en el alimento
- **Sugar_g:** Gramos de azúcar en el alimento
- **Calcium_mg:** Miligramos de calcio en el alimento
- **Iron_mg:** Miligramos de hierro en el alimento
- **Magnesium_mg:** Miligramos de magnesio en el alimento
- **Phosphorus_mg:** Miligramos de fósforo en el alimento
- **Potassium_mg:** Miligramos de potasio en el alimento
- **Sodium_mg:** Miligramos de sodio en el alimento
- **Zinc_mg:** Miligramos de zinc en el alimento
- **Copper_mcg:** Microgramos de potasio en el alimento
- **Manganese_mg:** Microgramos de manganeso en el alimento
- **Selenium_mcg:** Microgramos de selenio en el alimento
- **VitC_mg:** Miligramos de vitamina C en el alimento
- **Thiamin_mg:** Miligramos de tiamina en el alimento
- **Riboflavin_mg:** Miligramos de riboflavina en el alimento
- **Niacin_mg:** Miligramos de niacina en el alimento

- **VitB6_mg:** Miligramos de vitamina B6 en el alimento
- **Folate_mcg:** Microgramos de folato en el alimento
- **VitB12_mcg:** Microgramos de vitamina B12 en el alimento
- **VitA_mcg:** Microgramos de vitamina A en el alimento
- **VitE_mg:** Miligramos de vitamina E en el alimento
- **VitD2_mcg:** Microgramos de vitamina D2 en el alimento

Atributos relevantes

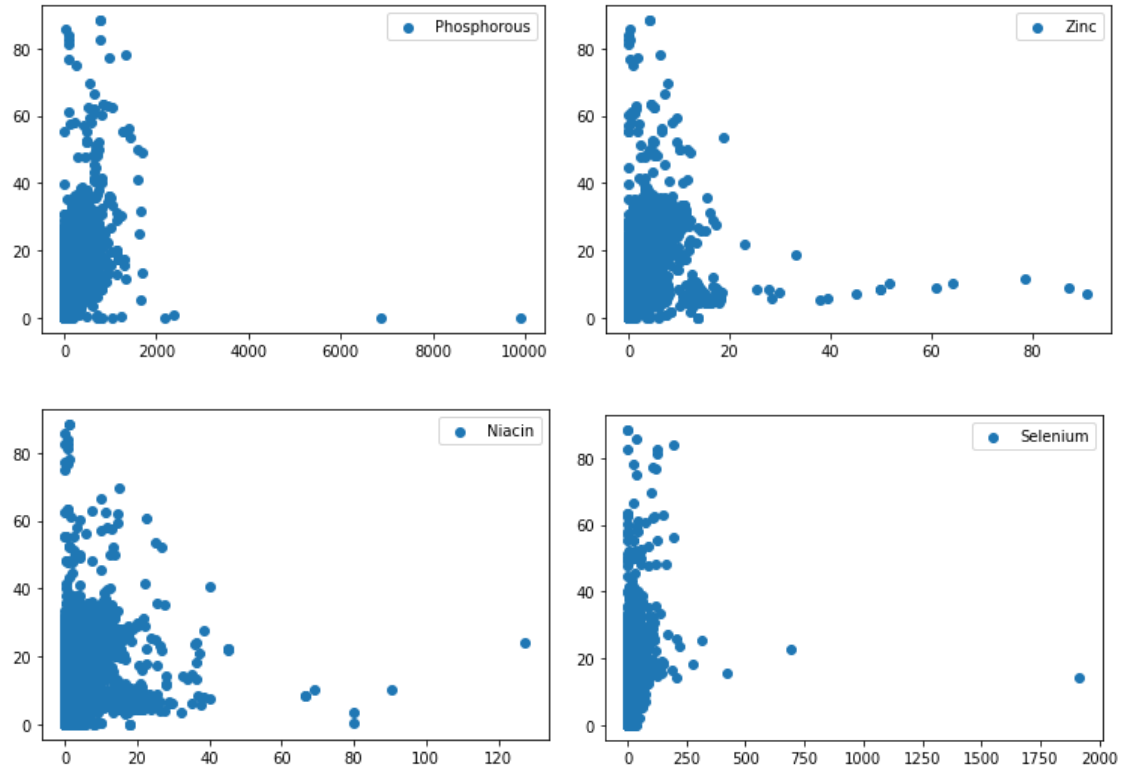
Podemos ver que las variables con mayor correlación con la cantidad de proteína presente en un alimento son el fósforo, el zinc, la niacina y el selenio. Esto lo podemos observar en la matriz de correlación que hemos creado:



Entre las variables podemos observar que no hay ningún índice que destaque, exceptuando lo que hemos comentado anteriormente, las calorías tienen una correlación muy alta con los

distintos nutrientes pues debemos pensar que se trata de una combinación lineal de nutrientes que dan paso a un valor en base a una fórmula de poca complejidad.

Dentro de las variables con mayor correlación, podemos observar una clara tendencia a valores bajos para los alimentos con alta cantidad de proteína, como podemos ver en los siguientes gráficos (Siendo el eje y siempre la cantidad de proteína presente):



Si nos fijamos en nuestro dataset, todos aquellos alimentos con alto contenido de fósforo, zinc, o niacina son cereales y zumos azucarados o agentes leudantes, cuyas cantidades de proteína sabemos que son mínimas.

Eliminación/Transformación

Nuestro dataframe presenta variables que no nos interesa mantener a la hora de realizar los modelos, por lo tanto, las eliminaremos. Estas variables son:

- **NDB_No:** Se trata de una etiqueta para referirnos a nuestro alimento, no nos proporciona ninguna información sobre sus valores nutricionales ni tiene relación alguna con la variable respuesta.
- **Descrip:** Las mismas razones que la variable anterior
- **VitE_mg:** Los datos que presenta este dataset sobre esta variable son complicados de tratar, así que, para simplificar el ejercicio, la eliminaremos

Trata de Missing Values

Para la trata de missing values, solo hemos tenido que eliminar aquellas instancias de nuestro dataset que no tenían valores del magnesio presente en el alimento. El resto de variables (exceptuando la vitamina E, que ya la hemos eliminado por las complicaciones que daba) no presentan ningún tipo de problemas respecto a los datos.

Protocolo de remuestreo

Tras haber limpiado y preprocesado el conjunto de datos, hemos decidido realizar permutaciones aleatorias para eliminar cualquier variante que pueda haber por seguir la ordenación de los datos, pues podrían estar ordenados de mayor a menos en algún campo y no queremos coger los elementos mayores o menores.

Una vez realizada la permutación hemos realizado una partición aleatoria de los datos en 80% el conjunto de entrenamiento y un 20% el conjunto de test la cuál ha sido utilizada para evaluar los modelos adecuadamente.

Seguidamente, a la hora de realizar validación cruzada y probar con los diferentes parámetros, hemos decidido fijar una K=10 que nos servirá a la hora de realizar *cross validation*. Esta *K* ha sido descubierta después de coger el mejor resultado entre los existentes probando aquellas combinaciones de hiper parámetros posibles

Resultados modelos lineales

Seguidamente analizaremos y explicaremos los resultados que hemos obtenido en nuestro problema después de aplicar modelos lineales. Este tipo de modelos se caracterizan por su simplicidad a la hora de hacer predicciones usando técnicas de regresión. Creemos que este tipo de modelos se ajustarán bastante bien a nuestro problema ya que, debido a su bajo coste computacional y simplicidad, deberían ser capaces de resolver este tipo de problemas de baja complejidad al obtener una alta correlación entre las distintas variables.

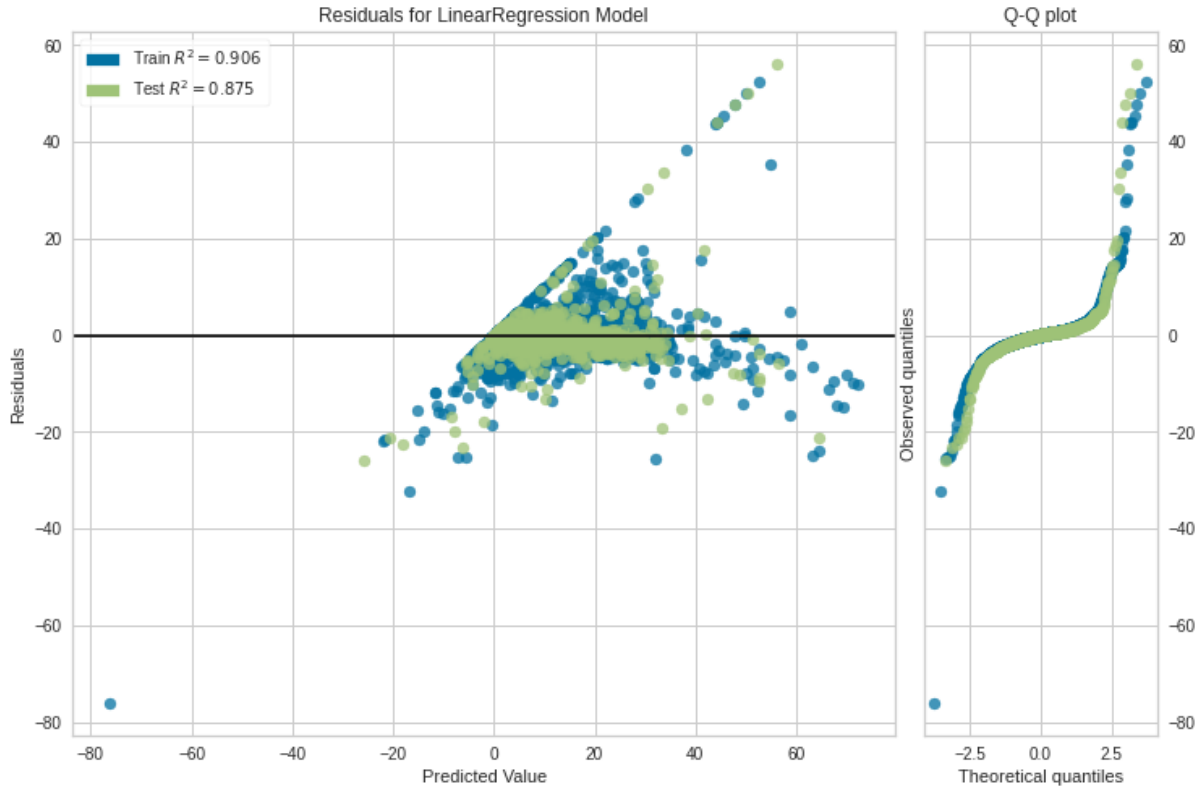
Gracias a esto dicho anteriormente y debido al problema al que nos enfrentemos, los modelos lineales deberían comportarse de manera adecuada y bastante precisa. Estos serán comparados con los modelos no lineales que utilicemos posteriormente.

Linear Regression

Primero de todo, realizaremos validación cruzada para asegurarnos de evitar overfitting en nuestro modelo y ser capaces de observar qué tan preciso podría llegar a ser nuestro modelo una vez lo evaluemos con el conjunto de test.

Realizaremos un primer modelo de regresión lineal, ya que es el más simple de los que hemos aprendido y nos servirá como base para comparar los resultados del resto de modelos. Vemos que este modelo nos da buenos resultados, obteniendo **0.875 de R^2**

Seguidamente hemos decidido hacer un residuals plot para ver el rendimiento del modelo y comparar la desviación de las predicciones que realiza el modelo con el valor real, hemos obtenido el siguiente gráfico:



Podemos observar que como hemos explicado anteriormente, los valores predichos por nuestro modelo son muy cercanos a los valores reales.

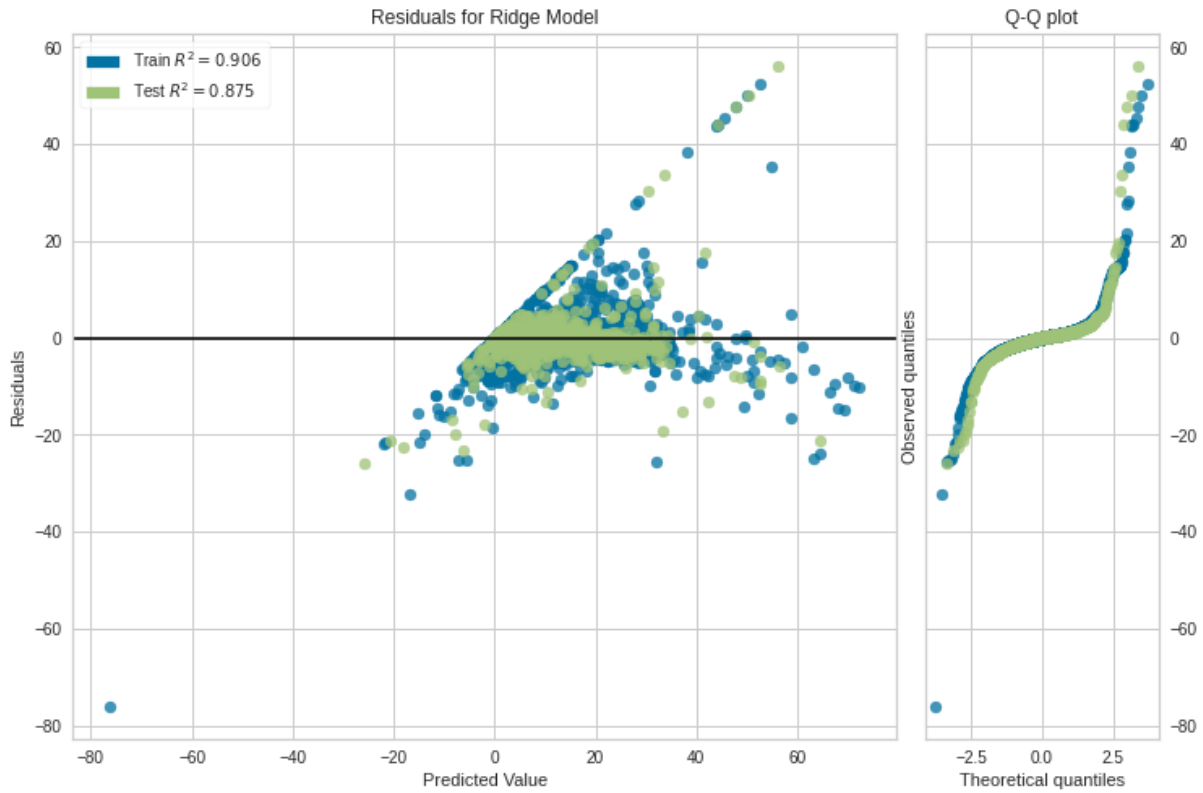
Tiene sentido que nos de un buen resultado, pues nuestro problema carece de complejidad a la hora de relacionar las variables entre sí y con un modelo lineal a veces es más que suficiente, por lo tanto, una regresión lineal en este problema de momento nos serviría para obtener los resultados que queremos.

Ridge Regression

Como segundo modelo lineal, probaremos Ridge Regression, una variante de la regresión lineal previamente utilizada, siendo la diferencia con este la aplicación de una penalización L2.

Al igual que con el modelo de linear regression, empezamos realizando la validación cruzada, testeando así diferentes lambdas para nuestro modelo y quedándonos con la que nos brinda mejores resultados, además de asegurarnos evitar posibles problemas relacionados con el overfitting de nuestro modelo. Hemos obtenido que la mejor lambda posible era **0.0001** con un resultado en la validación cruzada de **0.899 de R^2**.

Realizando este modelo con las mismas condiciones que el anterior, podemos observar que los **resultados son exactamente iguales.**



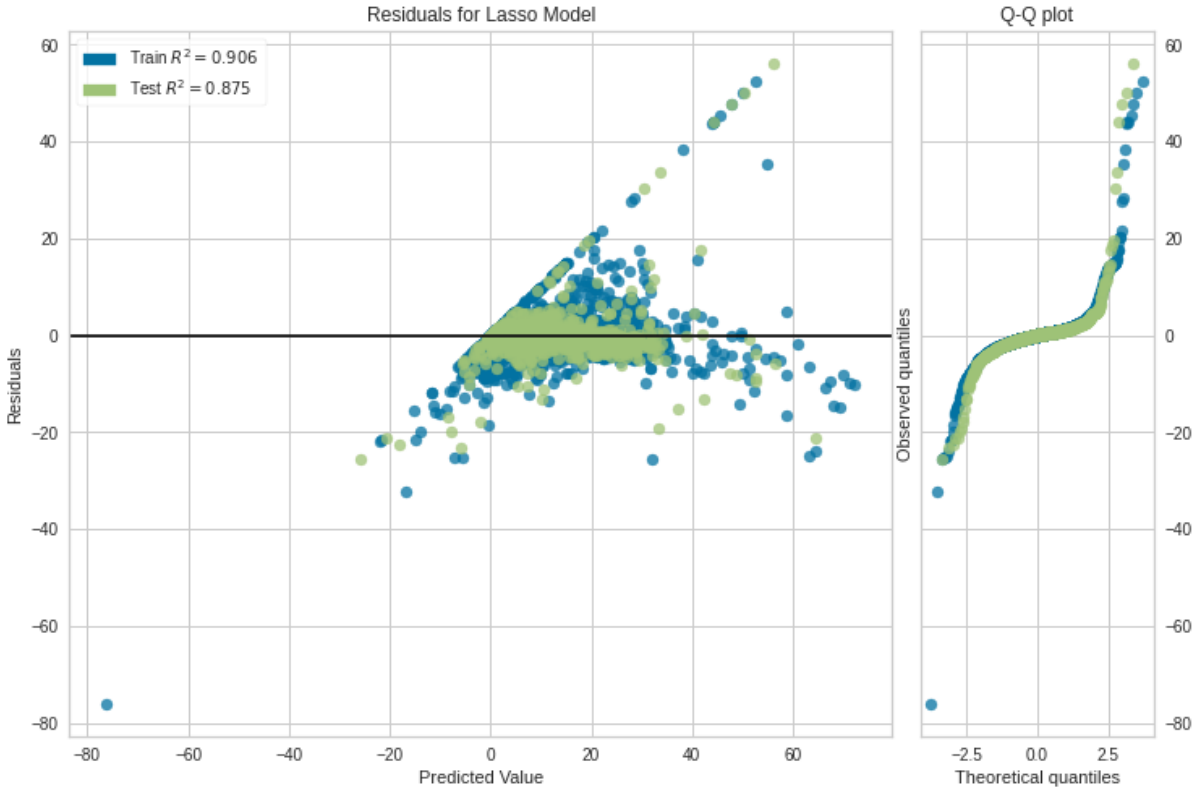
Los resultados dan tal y como lo esperábamos, pues realmente estamos aplicando una regresión lineal ya que como hemos dicho anteriormente, parece ser que una regresión lineal nos devolverá la optimalidad en ridge.

Lasso

El tercer y último modelo lineal que usaremos será Lasso, un modelo que se comporta igual que Ridge Regression, pero aplicando una penalización L1 en vez de L2.

Como hemos hecho con los anteriores modelos testados, en primer lugar realizaremos validación cruzada con K=5 para ver si el modelo podría provocar overfitting. Por otro lado, gracias a realizar la validación cruzada hemos testeado con distintos hiper parámetros quedándonos con el que nos dé mejores resultados, en este caso Lasso solo tiene un único hiper parámetro que es lambda, cuyo valor ha sido el mismo que para el modelo de Ridge Regression. En cuánto a la validación cruzada el mejor resultado que hemos obtenido con K=5 es de **0.862 de R^2.**

Los resultados obtenidos con este modelo son ligeramente mejores que en los dos modelos anteriores, obteniendo un **0.875 de R^2** en el conjunto de test.



Podemos observar como en los anteriores modelos que no se desvían demasiado los valores reales de los valores predichos.

Conclusiones modelos lineales

Finalmente, adjuntamos los resultados obtenidos de cada modelo. Podemos observar que claramente los 3 modelos nos dan casi los mismos resultados, siendo Lasso el mejor de ellos pero por muy poco. También podemos observar que la generalización de nuestros modelos es bastante alta, esto es debido a que no hay overfitting y lo fácil que le ha sido acertar los valores reales en el test.

	lr	ridge	lasso
Test R2	0.875288	0.875288	0.875301
Train R2	0.906018	0.906018	0.906018
lambda	0.000000	0.000100	0.000100

Una vez observado todos los modelos lineales que hemos escogido, podemos ver lo bien que funcionan con nuestro conjunto de datos, pues la correlación entre nuestras variables es bastante alta permitiendo así que un modelo lineal sea bastante eficaz y pueda obtener fácilmente los coeficientes que le permitan realizar la regresión lineal.

Resultados modelos no lineales

Para los modelos no lineales, debido a su alto coste computacional, hemos decidido usar la exploración bayesiana de hiper parámetros, ya que esta brinda buenos resultados con un tiempo de ejecución mucho menor.

SVM

El primero de los modelos no lineales que probaremos será un SVM con kernel RBF. El tiempo de ejecución de este modelo es muy alto, lo que nos lleva a no poder experimentar muchos valores para los hiper parámetros.

Al igual que con los modelos no lineales, lo primero que haremos será calcular la validación, testeando diferentes valores para los hiper parámetros parámetros para nuestro modelo y obteniendo los que nos brindan los mejores resultados, de manera que los usaremos para nuestro modelo, además de asegurarnos evitar problemas relacionados con el overfitting. Una vez realizada la validación cruzada, observamos que, de todos los valores testeados para nuestros diferentes parámetros de nuestro modelo, los que mejores resultados presentan son:

	params	mean_test_score	rank_test_score
11	{'C': 109.64781961431851, 'epsilon': 0.003, 'gamma': 'scale'}	0.858146	1
5	{'C': 109.64781961431851, 'epsilon': 0.01, 'gamma': 'scale'}	0.858140	2
14	{'C': 18.197008586099827, 'epsilon': 0.008, 'gamma': 'scale'}	0.791286	3
13	{'C': 9.120108393559097, 'epsilon': 0.008, 'gamma': 'scale'}	0.768160	4
10	{'C': 7.943282347242813, 'epsilon': 0.008, 'gamma': 'scale'}	0.763400	5

Siendo 0.858 el mejor valor de mean test score obtenido, por lo tanto, esos serán los parámetros que usaremos para nuestro SVM. Una vez creado el modelo con los mejores parámetros encontrados en nuestro testeo, pasaremos a comprobar cuales son las variables más importantes y con más peso a la hora de determinar la respuesta de nuestro modelo, obteniendo la siguiente lista ordenada:

```
Energy_kcal2.293 +/- 0.058
Fat_g   1.500 +/- 0.040
Saturated_fats_g0.981 +/- 0.024
Magnesium_mg0.234 +/- 0.009
Phosphorus_mg0.020 +/- 0.005
Sugar_g 0.016 +/- 0.005
Riboflavin_mg0.016 +/- 0.002
VitB6_mg0.013 +/- 0.003
Potassium_mg0.013 +/- 0.006
VitB12_mcg0.012 +/- 0.003
Manganese_mg0.011 +/- 0.001
Protein_g0.005 +/- 0.000
Iron_mg 0.004 +/- 0.001
Sodium_mg0.002 +/- 0.000
Fiber_g 0.002 +/- 0.000
Carb_g  0.001 +/- 0.000
Calcium_mg0.000 +/- 0.000
```

Como podemos observar, y como era de esperar, las kilocalorías y las grasas son las variables más decisivas en nuestro problema, ya que, como sabemos, los alimentos altos en proteína suelen tener altos valores calóricos y grasos, como, por ejemplo, todo tipo de carne o frutos secos.

Como vemos, el resultado obtenido es un **score de 0.83 R^2**, por lo que nos brinda unos resultados peores que los modelos lineales. Todo esto nos hace concluir que este modelo no es apropiado para nuestro ejercicio, ya que no solo nos da unos resultados peores, sino que su coste es elevadísimo.

Random Forest

El segundo de los modelos no lineales con los que trabajaremos será un RandomForest Regressor. Este tiene un tiempo de ejecución menor lo que nos permitirá probar diferentes valores para nuestros hiper parámetros en la fase de testeo de los mismos, pero, aun así, limitaremos la exploración de hiper parámetros a la exploración bayesiana para reducir el tiempo total de ejecución de nuestro programa.

Realizamos el cálculo de la validación cruzada, obteniendo los hiper parámetros que nos devuelven mejores resultados y evitando problemas relacionados con el overfitting de nuestro modelo. Estos son los mejores hiper parámetros para nuestro random forest:

	params	mean_test_score	rank_test_score
13	{'criterion': 'squared_error', 'max_depth': 15, 'min_samples_leaf': 2, 'n_estimators': 100}	0.919453	1
11	{'criterion': 'friedman_mse', 'max_depth': 10, 'min_samples_leaf': 1, 'n_estimators': 40}	0.909151	2
5	{'criterion': 'squared_error', 'max_depth': 15, 'min_samples_leaf': 5, 'n_estimators': 40}	0.901057	3
6	{'criterion': 'squared_error', 'max_depth': 9, 'min_samples_leaf': 3, 'n_estimators': 10}	0.893245	4
3	{'criterion': 'absolute_error', 'max_depth': 9, 'min_samples_leaf': 3, 'n_estimators': 40}	0.891577	5

Podemos observar que el mejor criterio que ha encontrado es el *squared_error*, junto a una máxima profundidad de 15 con 100 estimadores. Aún así obtenemos una muy buena puntuación de $\mathbf{R^2}$ siendo este de **0.91**, el único temor que podemos encontrar aquí es el sobreajuste del conjunto de train, por lo tanto, seguidamente evaluaremos el modelo para ver qué tanta generalización tenemos.

Seguidamente obtenemos también lo importantes que han sido las features para el modelo:

```
Fat_g   0.549 +/- 0.021
Magnesium_mg0.323 +/- 0.022
Energy_kcal0.072 +/- 0.005
Manganese_mg0.050 +/- 0.011
Sodium_mg0.046 +/- 0.003
Calcium_mg0.043 +/- 0.005
Phosphorus_mg0.024 +/- 0.004
Riboflavin_mg0.018 +/- 0.002
Saturated_fats_g0.016 +/- 0.
Iron_mg 0.013 +/- 0.002
Zinc_mg 0.012 +/- 0.002
VitB6_mg0.007 +/- 0.001
Protein_g0.006 +/- 0.002
Thiamin_mg0.006 +/- 0.001
Fiber_g 0.003 +/- 0.000
Potassium_mg0.003 +/- 0.001
Carb_g  0.002 +/- 0.001
Folate_mcg0.002 +/- 0.001
Niacin_mg0.001 +/- 0.000
Copper_mcg0.001 +/- 0.000
VitC_mg 0.001 +/- 0.000
Selenium_mcg0.001 +/- 0.000
VitB12_mcg0.001 +/- 0.000
```

Observamos así que la grasa junto al magnesio son dos valores determinantes, ya que, como hemos visto antes, los alimentos altos en proteína tienden a tener altos valores calóricos y los que contienen altas cantidades de magnesio suelen ser alimentos de origen vegetal, con bajas cantidades de proteína.

Los resultados obtenidos nos muestran que el **score en el conjunto de test es igual a 0.89 de $\mathbf{R^2}$**, un valor alto y satisfactorio, que pasa por delante a todos los modelos que hemos visto

hasta ahora, podemos observar también que, como ocurre en otros modelos, estamos generalizando bastante y somos capaces de deshacernos del overfitting que temíamos.

Gradient Boosting

El tercer y último modelo de regresión no lineal con el que trabajaremos será un Gradient Boosting Regressor. Este será el modelo con el que más parámetros debemos testear. Al igual que en el resto de modelos no lineales, realizaremos una búsqueda bayesiana sobre diferentes valores posibles para nuestros parámetros.

Realizada la validación cruzada, obtenemos un valor de 0.8999 de cross validación, con lo que nos aseguramos no tener problemas relacionado con el ajuste de nuestro modelo. Por otra banda, obtendremos la lista de los parámetros con los que conseguimos los mejores resultados para nuestro modelo:

	params	mean_test_score	rank_test_score
10	{'criterion': 'squared_error', 'learning_rate': 0.5, 'loss': 'huber', 'max_depth': 15, 'min_samples_leaf': 2, 'n_estimators': 40}	0.899930	1
13	{'criterion': 'friedman_mse', 'learning_rate': 0.5, 'loss': 'huber', 'max_depth': 2, 'min_samples_leaf': 2, 'n_estimators': 40}	0.841680	2
14	{'criterion': 'friedman_mse', 'learning_rate': 0.5, 'loss': 'huber', 'max_depth': 2, 'min_samples_leaf': 2, 'n_estimators': 40}	0.841680	2
11	{'criterion': 'friedman_mse', 'learning_rate': 0.5, 'loss': 'huber', 'max_depth': 2, 'min_samples_leaf': 2, 'n_estimators': 25}	0.821871	4
12	{'criterion': 'friedman_mse', 'learning_rate': 0.5, 'loss': 'huber', 'max_depth': 2, 'min_samples_leaf': 2, 'n_estimators': 25}	0.821871	4

Como vemos, los parámetros que nos brindan los mejores resultados se basan en un criterio de squared error, una tasa de aprendizaje de 0.5, una función de pérdida basada en la fusión de squared y absolute error, una profundidad máxima de 15 de los estimadores, un número de ejemplos mínimo por hoja de 2, y 40 etapas de boosting.

Una vez creado nuestro modelo con los mejores valores para sus parámetros, exploramos cuáles són las variables con mayor peso para la respuesta que nos proporcionará.

```
Fat_g   0.613 +/- 0.028
Magnesium_mg0.291 +/- 0.030
Energy_kcal0.252 +/- 0.011
Saturated_fats_g0.145 +/- 0.012
Manganese_mg0.105 +/- 0.029
Sodium_mg0.084 +/- 0.005
Calcium_mg0.064 +/- 0.009
Iron_mg 0.028 +/- 0.003
Folate_mcg0.024 +/- 0.004
Riboflavin_mg0.019 +/- 0.002
Zinc_mg 0.017 +/- 0.003
Phosphorus_mg0.015 +/- 0.001
Thiamin_mg0.012 +/- 0.002
VitB6_mg0.012 +/- 0.004
Protein_g0.009 +/- 0.002
Carb_g  0.007 +/- 0.001
Potassium_mg0.006 +/- 0.002
Fiber_g 0.005 +/- 0.001
Copper_mcg0.004 +/- 0.001
VitC_mg 0.003 +/- 0.001
Selenium_mcg0.001 +/- 0.000
```

Como vemos, al igual que en el modelo anterior, las tres variables más significativas son las kilocalorías, el magnesio y las grasas,

Como vemos, **el score devuelto por este modelo es de 0.86 de R^2**, por lo que proporciona una respuesta igual de buena que los modelos lineales, estando, por lo tanto, por debajo de la calidad de nuestro Random Forest.

Conclusiones modelos no lineales

Una vez observado todos los modelos no lineales que hemos escogido, hemos observado que el modelo que nos ha dado mejores resultados ha sido random forest con un satisfactorio R2 de **0.89**. Sin embargo, hemos observado que los resultados son muy similares a los modelos lineales y la exploración de hiper parámetros tiene un coste demasiado elevado para la poca diferencia entre ambos conjuntos

Por otro lado, podemos ver lo bien que funcionan con nuestro conjunto de datos aunque los resultados son muy similares al conjunto de modelos lineales, pues estos funcionan muy bien con nuestro conjunto de datos e incluso superan a los resultados del modelo de SVM.

Selección de modelo final

Habiendo obtenido los resultados de los 6 modelos anteriores, podemos concluir que el mejor de todos ellos ha sido el Random Forest, ya que todos nos han presentado buenos resultados de score con el conjunto de entrenamiento, pero Random Forest ha sido el que ha destacado en el conjunto de test. Hemos comprobado también que los modelos lineales funcionan bastante bien en nuestro conjunto de datos, esto es debido a la sencillez y la correlación de las variables con la variable *target*.

Debido a que Random Forest es el modelo que más R2 nos da, hemos decidido quedarnos con él ya que no perderemos tanta puntuación a la hora de generalizar y será el que mejor funcione cuando obtengamos datos nuevos.

Por otro lado, pensamos que quizá un modelo lineal podría encajar bastante bien en nuestros datos también y podría ser bueno valorar el tema eficiencia, pues Random Forest es un modelo el cuál tiene un coste elevado si queremos realizar una búsqueda en profundidad en sus hiper parámetro. Sin embargo, ese 0.02 de puntuación más que tiene Random Forest sobre los modelos lineales podría no merecer tanto la pena.

En conclusión, nos gustaría probar con nuevos inputs modelos lineales que creen regresiones sencillas para comprobar si merece la pena quedarnos con Random Forest como hemos decidido anteriormente, ya que no sabemos si la poca diferencia que hay entre ambos modelos seguirá escalando o simplemente se volverán igual de precisos debido a la baja complejidad del problema y facilidad de establecer una regresión gracias a la alta correlación entre las variables.

Conclusiones

Creemos que el trabajo realizado en esta práctica es satisfactorio, tanto por la parte de encontrar un buen dataset (ya que el dataset utilizado contiene mucha información y apenas variables irrelevantes o valores perdidos), como a la hora de tratarlo en la creación de modelos.

Nos gustaría haber tenido un mayor conocimiento sobre los macronutrientes de los alimentos y su posible relación con la cantidad de proteína presente en el, ya que así podríamos haber hecho un preproceso mucho más completo y haber obtenido resultados mejores.

Por otra banda, nos sorprende que los modelos lineales nos presenten resultados tan buenos siendo tan simples y rápidos de ejecutar, ya que, los no lineales representan un coste computacional mucho mayor y solo uno de ellos nos ha brindado resultados mejores.

Referencias

Librerías utilizadas

- Pandas
- Seaborn
- Sklearn
- Matplotlib
- Numpy
- Skopt

```
import pandas as pd
import seaborn as sns
from sklearn.model_selection import GridSearchCV
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import numpy as np
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.linear_model import LinearRegression, Ridge, RidgeCV, Lasso, LassoCV
from sklearn.model_selection import train_test_split,  KFold, cross_val_score
from sklearn.svm import LinearSVR, SVR
from sklearn.metrics import mean_squared_error, make_scorer, mean_absolute_error
!pip install scikit_optimize
!apt install skopt
from skopt import BayesSearchCV
from IPython.display import display, HTML
show_html = lambda html: display(HTML(html))
from sklearn.inspection import permutation_importance
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
```

Documentación bibliográfica

- *scikit-learn: machine learning in Python — scikit-learn 0.16.1 documentation.* (s. f.). https://scikit-learn.org
- Koehrsen, W. (2019, 10 diciembre). *Hyperparameter Tuning the Random Forest in Python - Towards Data Science.* Medium. https://towardsdatascience.com/hyperparameter-tuning-the-random-forest-in-python-using-scikit-learn-28d2aa77dd74
- *How does cross-validation overcome the overfitting problem?* (2011, 1 abril). Cross Validated. https://stats.stackexchange.com/questions/9053/how-does-cross-validation-overcome-the-overfitting-problem