Introducción a la regresión logística

Objetivos de aprendizaje

Al final de este módulo, serán capaces de:

- Describir la regresión logística
- Indicar las ventajas y desventajas de la regresión logística

Regresión logística

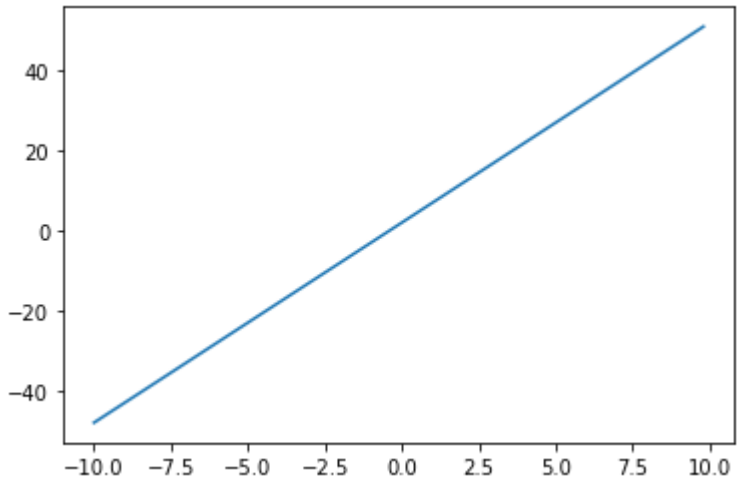
Puede ser muy confuso al principio, pero los modelos de regresión logística se usan para los problemas de clasificación a pesar de tener la palabra “regresión” en el nombre.

Antecedentes - Delimitación de salida

Antes de explicar la regresión logística, hay que entender cómo delimitar una salida entre 0 y 1 utilizando una función sigmoide.
Primero mostremos una relación linear donde una salida continua (y) se modela como una combinación linear de características.

$$y = \beta_0 + \beta_1 x$$

```
import numpy as np
import matplotlib.pyplot as plt
# B0 is 2 in this case
# B1 is 5 in this case
# I put in numbers to show where they go
x = np.arange(-10., 10., 0.2)
y = 2 + 5* x
plt.plot(x, y)
```

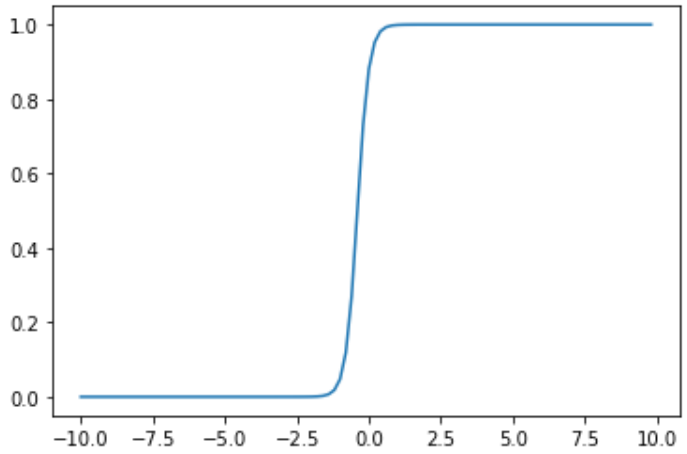


La regresión logística utiliza algo llamado una función sigmoide, el cual delimita la salida entre 0 y 1. Cuando el objetivo de lo que quieren predecir es binario, esto hará que la regresión logística produzca las probabilidades de una clase específica. Las probabilidades se pueden convertir en predicciones de clase.

$$y = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x)}}$$

Así es cómo se convierte un algoritmo de regresión como una regresión linear para tener un algoritmo de clasificación como una regresión logística.
Pueden pensar del nombre de regresión logística como un nombre engañoso. A pesar de su nombre, la regresión logística es un algoritmo que usa para los problemas de clasificación. Es importante entender esto para que se mencione otra vez. La regresión logística es un algoritmo que usa para los problemas de clasificación.

```
# B0 is 2 in this case
# B1 is 5 in this case
# I put in numbers to show where they go
def sigmoid(x):
  s = 1 / (1 + np.exp(-(2 + 5*x)))
  return s
plt.plot(x, sigmoid(x))
```



Esto tendrá más sentido en las secciones siguiente con un ejemplo. Miren el próximo video para entender cómo funciona la regresión logística intuitivamente.

Ahora intentémoslo en Python

```
#Monten los datos

from google.colab import drive
drive.mount('/content/drive')
```

```
#Importen las bibliotecas
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
```

Carguen el conjunto de datos
Los datos que usaremos es el conjunto de datos de Cáncer de Mama de Wisconsin (Diagnóstico), que se puede descargar aquí. El objetivo de esta predicción es clasificar el cáncer de manera eficaz como maligno (1) o benigno (0).

```
df = pd.read_csv('/content/drive/My Drive/CHANGEYOURPATH/wisconsinBreastCancer.csv')
df.head()
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	...	radius_worst	texture_worst	perimeter_worst	ar
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	...	25.38	17.33	184.60	
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	...	24.99	23.41	158.80	
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	...	23.57	25.53	152.50	
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	...	14.91	26.50	98.87	
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	...	22.54	16.67	152.20	

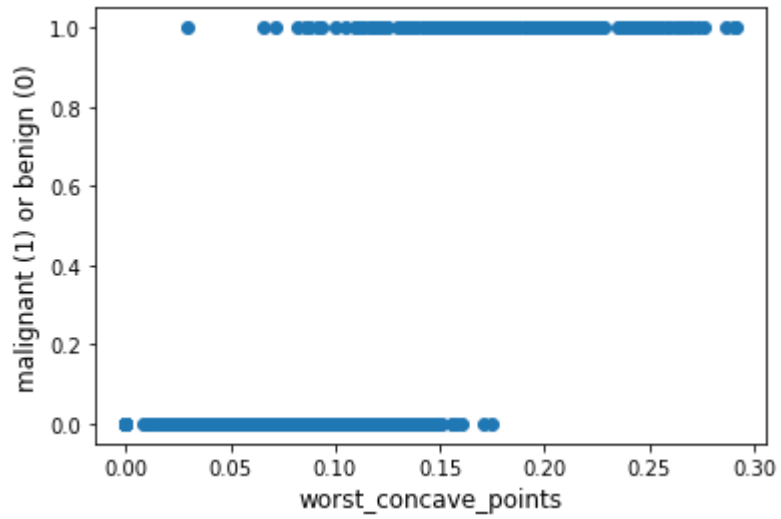
5 rows x 32 columns

```
#Conviertan el objetivo a valores numéricos

df['diagnosis'] = df['diagnosis'].replace({'B':0, 'M':1})
```

Visualicen la relación entre concave points_worst y diagnosis (objetivo)

Es una buena idea tratar de visualizar la salida.

```
plt.scatter(df['concave points_worst'], df['diagnosis'])
plt.ylabel('malignant (1) or benign (0)', fontsize = 12)
plt.xlabel('concave points_worst', fontsize = 12)
```

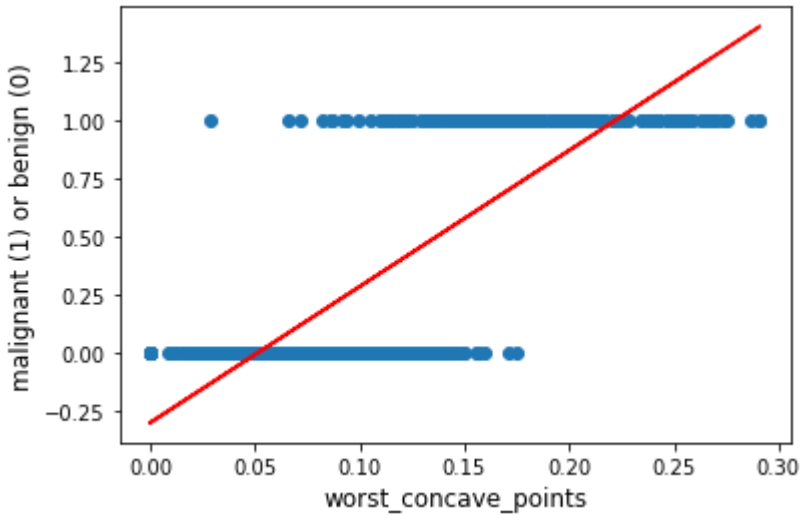


Intento de regresión lineal para clasificar los datos

La regresión lineal era buena cuando queríamos predecir un valor continuo. Esta sección solo muestra cómo intentar usar la regresión lineal para clasificar si el problema es maligno (1 en el gráfico anterior) o benigno (0 en el gráfico de abajo).

```
# Assign feature matrix and target vector
X = df[['concave points_worst']]
y = df['diagnosis']
# Train Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)
```

```
# Make a linear regression instance
lr = LinearRegression()
# Training the model on the data, storing the information learned from the data
# Model is learning the relationship between X and y
lr.fit(X_train,y_train)
# Get Predictions
predictions = lr.predict(X_test)
# This code shows how to graph the results comparing the prediction with the actual value
plt.scatter(X_test['concave points_worst'], X_test['diagnosis'])
plt.plot(X_test['concave points_worst'], predictions, color='red')
plt.ylabel('malignant (1) or benign (0)', fontsize = 12)
plt.xlabel('concave points_worst', fontsize = 12)
```



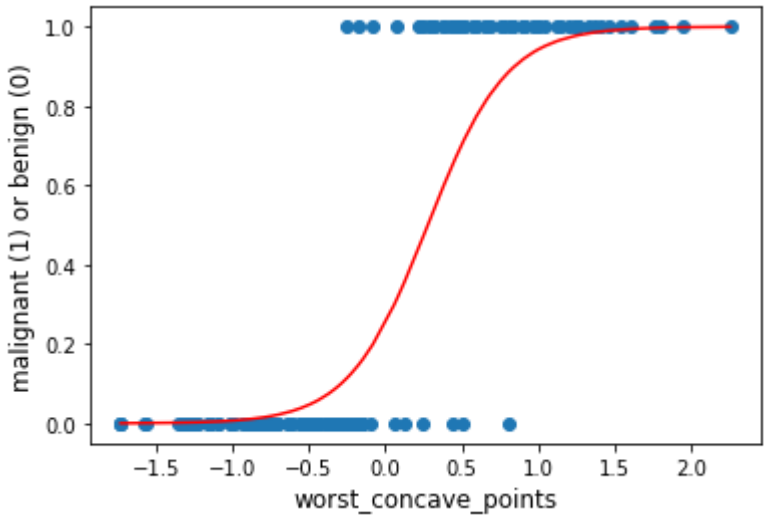
De acuerdo con esto, cualquier valor de predicción (rojo) >= 0,5 (que corresponde al valor cerca de 0,15 para worst_concave_point), predecimos una clase 1 (maligno), sino predecimos una clase de 0 (benigno).

Problema: Si el valor para worse_concave_points is 0,0, ¿qué significa cuando tenemos -0,25 para nuestra clase en lugar de 1 o 0? Parece extraño. Quizás deberíamos restringir nuestras predicciones entre 0 y 1. Eso se puede hacer usando la regresión logística.

```
# Make an instance of the model
logreg = LogisticRegression(C = 1000)
# Instantiate Scaler
scaler = StandardScaler()
# Create pipeline
logreg_pipe = make_pipeline(scaler, logreg)
# Training the model on the data, storing the information learned from the data
# Model is learning the relationship between X and y
logreg_pipe.fit(X_train,y_train)
```

```
Pipeline(steps=[('standardscaler', StandardScaler()),
                ('logisticregression', LogisticRegression(C=1000))])
```

```
# This code is just to plot/visualize the predictions
example_df = pd.DataFrame(data = {'worst_concave_points': X_test['concave
points_worst'],
                  'diagnosis': y_test})
example_df['logistic_preds']                                                =
pd.DataFrame(logreg_pipe.predict_proba(X_test)).loc[:, 1].values
example_df = example_df.sort_values(['logistic_preds'])


plt.scatter(X_test['concave points_worst'], y_test)
plt.plot(example_df['worst_concave_points'],
example_df['logistic_preds'].values, color='red')
plt.ylabel('malignant (1) or benign (0)', fontsize = 12)
plt.xlabel('concave points_worst', fontsize = 12)
```



Regresión logística

Observen que podemos escalar los datos para usarlos en el modelo de modelo de regresión.
Las probabilidades previstas están ahora más calibradas por lo que ahora se puede predecir un 0 o un 1 dependiendo de los puntos de worst_concave_points.

Ventajas y desventajas del algoritmo

Ventajas

- Capaz de interpretar cómo el modelo hace predicciones
- El modelo de entrenamiento y predicción son relativamente rápidos

- Puede funcionar bien con un número reducido de observaciones
- Generalmente no se necesita de ajustes para la regresión logística a menos se quiera regularizar el modelo.

Desventajas

- Requiere un escalado de características
- El algoritmo de clasificación binaria no funciona para problemas multiclase.

Lecturas opcionales

Métrica de predicciones binarias: http://mfviz.com/binary-predictions/
Regresión logística en Python: https://realpython.com/logistic-regression-python/

Afinación de hiperparámetro en regresión logística

Objetivos de aprendizaje

Al final de este módulo, serán capaces de:

- Comprender la regularización L1 y L2
- Afinar los hiperparámetros en la regresión logística para balancear el sesgo y la varianza
- Explicar el parámetro C

Tarea

El propósito de esta sección es aprender cómo afinar un modelo de regresión logística y específicamente comprender el parámetro C de scikit-learn. No se preocupen si no comprenden todos los cálculos.

Propósito de la regularización L1 y L2

La regularización tenderá a reducir el sobreajuste en un modelo. Tenderá a reducir el rendimiento del modelo en el conjunto de entrenamiento, pero finalmente a mejorar el rendimiento en conjuntos de pruebas.
Una forma de pensar sobre la regularización es como una herramienta para introducir información adicional (sesgo) para penalizar valores extremos de parámetros (peso). Existen dos tipos comunes de regularización que trataremos en este curso: L1 y L2. Ambos introducen un término de penalización a las ponderaciones (coeficientes) que el modelo usa para hacer predicciones. Más grandes las ponderaciones, más grande es la penalización. Si el rendimiento del modelo es su principal preocupación, es mejor intentar ambos.

¿Cuándo usar L1?
La regularización L1 introduce una penalización que empujará algunas o varias ponderaciones a cero. Reducir la ponderación de característica a cero es útil en casos donde se tiene un número grande de características.

¿Cuándo usar L2?
L2 puede funcionar particularmente bien si hay un alto grado de multicolinealidad en el modelo (alta correlación entre las características), filtrando el ruido de los datos y previniendo el sobreajuste (L1 también puede ayudar en el sobreajuste).

¿Cómo se relaciona esto con scikit-learn?

El parámetro C en la regresión logística es el inverso de alfa. (Verán donde se usa el alfa en la sección de matemáticas de abajo). Por lo tanto, al disminuir el valor de C, aumenta la fuerza de la regularización. En resumen, C = 1 / alfa. Tengan en cuenta que se trata de una convención que puede resultar bastante confusa al principio.

Las matemáticas

Contexto - Función de costo de la regresión logística sin regularización
Un objetivo común de un modelo de regresión logística es minimizar la siguiente función de costo (hay variantes de funciones de costo que a veces se usan). Esta función de costo comúnmente se llama “log loss” (pérdida logarítmica). Tengan en cuenta que esto se menciona porque es una pregunta común de entrevista.

$$LogLoss = -\frac{1}{m}\sum_{i=1}^{m} y^{(i)} log(h_\theta(x^{(i)})) + (1 - y^{(i)}) log(1 - h_\theta(x^{(i)}))$$

Las matemáticas: regularización L1

La ecuación LogLoss es la misma hasta que la última pieza agregada al final. El último término es la penalización de regularización L1.

$$LogLoss = -\frac{1}{m}\sum_{i=1}^{m} y^{(i)} log(h_\theta(x^{(i)})) + (1 - y^{(i)}) log(1 - h_\theta(x^{(i)})) + \alpha \sum_{j=1}^{n} |\theta_j|$$

Las matemáticas: Regularización L2

La ecuación LogLoss es la misma hasta que la última pieza agregada al final. El último término es la penalización de regularización L2. La diferencia entre la penalización L1 y L2 es el cuadrado de zeta en el término de penalización. L1 penaliza utilizando el valor absoluto de zeta mientras que L2 penaliza basándose en el cuadrado de zeta.

$$LogLoss = -\frac{1}{m}\sum_{i=1}^{m} y^{(i)} log(h_\theta(x^{(i)})) + (1 - y^{(i)}) log(1 - h_\theta(x^{(i)})) + \alpha \sum_{j=1}^{n} \theta_j^2$$

- m is the number of samples
- $h_\theta(x_i)$ is the model prediction
- $y^{(i)}$ is the observed result
- n is the number of features.
- θ_j is a model coefficient.
- α is a tuning parameter:
  - A tiny α imposes no penalty on the coefficient size, and is equivalent to a normal logistic regression model.
  - Increasing the α penalizes the coefficients and thus shrinks them.

Ejemplo: Optimización de C

Los datos que utilizaremos es el conjunto de datos del cáncer de mama en Wisconsin (diagnóstico) que usamos en un módulo anterior. El objetivo de esta predicción es clasificar el cáncer como maligno (1) o benigno (2).

Importen las bibliotecas

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
```

Carguen el conjunto de datos

```
df = pd.read_csv('YOURPATH')
df.head()
```

Regresión logística

El código de abajo muestra un modelo de regresión logística por defecto. El modelo por defecto utiliza solver = 'lbfgs' con regularización L2 y un valor C de 1,0.

```
X = df.drop(columns=['diagnosis', 'id'])
```

```
y = df['diagnosis']

# Train test split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)
# Make an instance of the model with default parameters
logreg = LogisticRegression()
# Instantiate Standard Scaler because scaling is required for logistic regression
scaler = StandardScaler()
# Put scaler and model in a pipeline
logreg_pipe = make_pipeline(scaler, logreg)
# Training the model on the data, storing the information learned from the data
# Model is learning the relationship between X and y
logreg_pipe.fit(X_train, y_train)
print(logreg_pipe.score(X_train, y_train))
print(logreg_pipe.score(X_test, y_test))
copy
```

```
Training Score 0.9859154929577465
Test Score 0.9790209790209791
```

Aquellos resultados son bastante buenos, pero veamos si podemos mejorar la puntuación de prueba al cambiar el valor de C.

Generalmente cuando cambiamos los valores de C, los cambiamos por orden de magnitud. La diferencia entre 0,1 y 0,2 no será de mucho, pero la diferencia entre 0,1 y 1 o 1 y 10 será mayor.

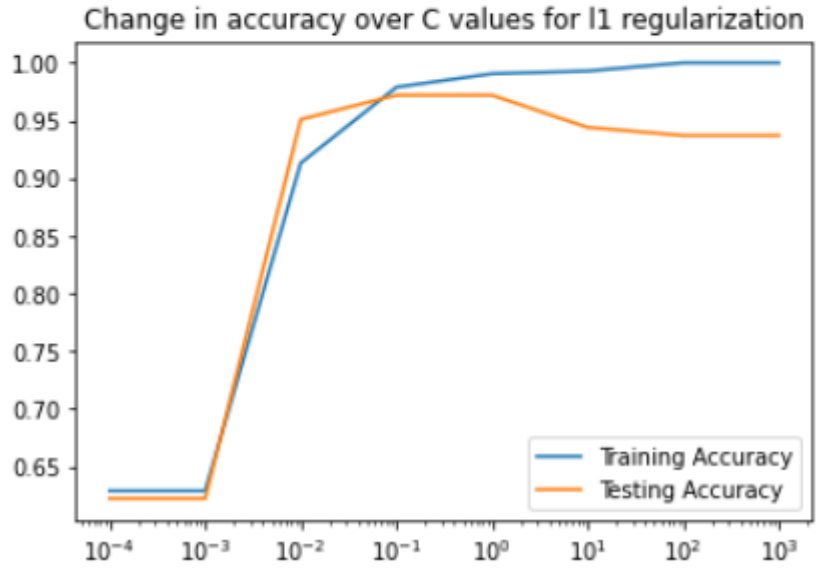
Afinación de L1:

Primero, intentemos la regularización L1. El solver por defecto no hará la regularización L1, por lo que necesitamos cambiar el solver en el modelo de regresión logística. Utilizaremos solver = 'liblinear'. Para más información sobre los hiperparámetros de regresión logística revisen la documentación sklearn.

```
# create a list of c values and empty lists for scores
c_values = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]
train_scores = []
test_scores = []
#iterative over the c values
for c in c_values:
```

```
  # instantiate a model with each value of c and fit it on the data
   log_reg = LogisticRegression(C=c, max_iter=1000, solver='liblinear',
penalty='l1')
  log_reg_pipe = make_pipeline(scaler, log_reg)
  log_reg_pipe.fit(X_train, y_train)

  # add the training and testing scores to our scores lists
  train_scores.append(log_reg_pipe.score(X_train, y_train))
  test_scores.append(log_reg_pipe.score(X_test, y_test))

#plot the change in accuracy as we change the value of C
fig, ax = plt.subplots(1,1)
ax.plot(c_values, train_scores, label='Training Accuracy')
ax.plot(c_values, test_scores, label='Testing Accuracy')
ax.set_xticks(c_values)
ax.set_title('Change in accuracy over C values for l1 regularization')
ax.legend()

#set the x axis to a logarithmic scale to show the values of C in even
intervals
ax.set_xscale('log')
# print a dictionary of values of C and accuracy scores
{c:score for c, score in zip(c_values, test_scores)}copy
```

```
{0.0001: 0.6223776223776224,
 0.001: 0.6223776223776224,
 0.01: 0.951048951048951,
 0.1: 0.972027972027972,
 1: 0.972027972027972,
 10: 0.9440559440559441,
 100: 0.9370629370629371,
 1000: 0.9370629370629371}
```



Recuerden que nuestro objetivo final es mejorar la exactitud en el conjunto de prueba.

Según este gráfico, obtenemos la mejor exactitud en el conjunto de prueba cuando C=0,1 (10^-1). Observen que a medida que se aumenta el valor de C, la puntuación en los datos de entrenamiento continúa aumentando. Esto se debe porque menos regularización tiende a reducir el sesgo. Sin embargo, después de cierto punto, menos regularización (mayor valor C) reduce la exactitud en los datos de prueba a medida que el modelo se sobreajusta demasiado. Más regularización (un valor C más pequeño) tiende a reducir la varianza mientras disminuye el sesgo.

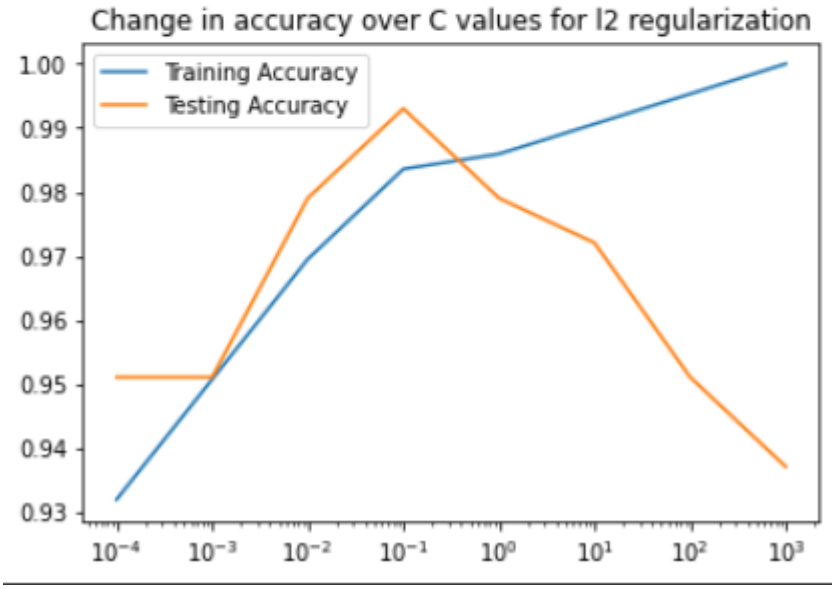
Afinación de L2

Cambiaremos el código levemente para usar la regularización L2.

```
# create a list of c values and empty lists for scores
c_values = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]
train_scores = []
test_scores = []
```

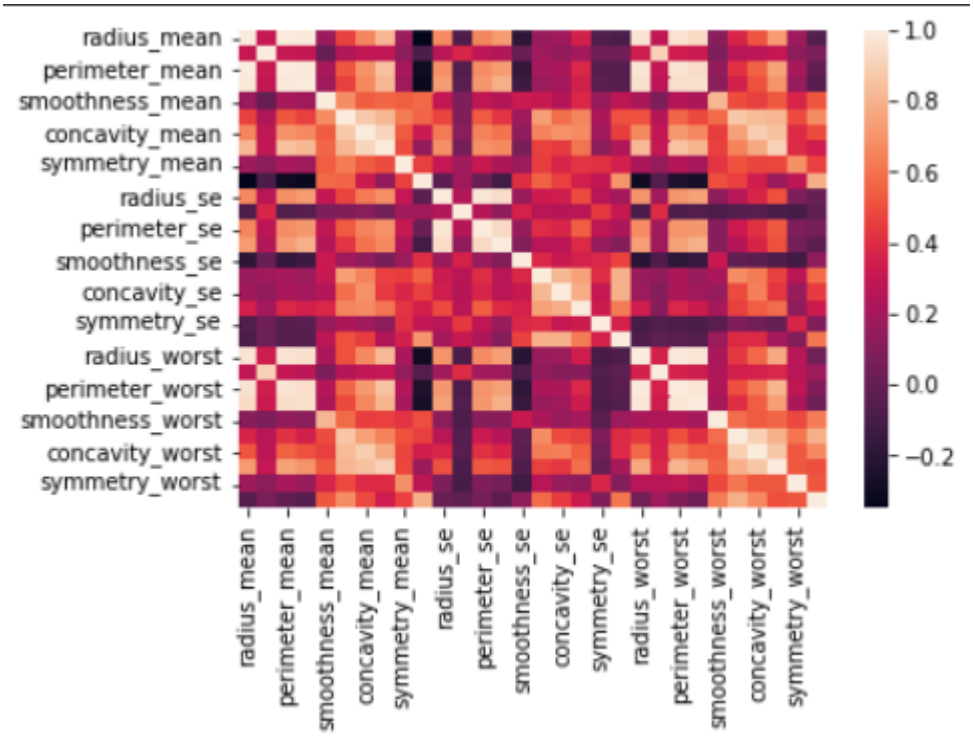
```
#iterative over the c values
for c in c_values:
  # instantiate a model with each value of c and fit it on the data
   log_reg = LogisticRegression(C=c, max_iter=1000, solver='liblinear',
penalty='l2')
  log_reg_pipe = make_pipeline(scaler, log_reg)
  log_reg_pipe.fit(X_train, y_train)

  # add the training and testing scores to our scores lists
  train_scores.append(log_reg_pipe.score(X_train, y_train))
  test_scores.append(log_reg_pipe.score(X_test, y_test))

#plot the change in accuracy as we change the value of C
fig, ax = plt.subplots(1,1)
ax.plot(c_values, train_scores, label='Training Accuracy')
ax.plot(c_values, test_scores, label='Testing Accuracy')
ax.set_xticks(c_values)
ax.set_title('Change in accuracy over C values for l2 regularization')
ax.legend()
#set the x axis to a logarithmic scale to show the values of C in even
intervals
ax.set_xscale('log')
# print a dictionary of values of C and accuracy scores
{c:score for c, score in zip(c_values, test_scores)}
```

```
{0.0001: 0.951048951048951,
 0.001: 0.951048951048951,
 0.01: 0.9790209790209791,
 0.1: 0.993006993006993,
 1: 0.9790209790209791,
 10: 0.972027972027972,
 100: 0.951048951048951,
 1000: 0.9370629370629371}
```



Con la regularización L2 conseguimos nuestra mejor puntuación en C=0,1, y las puntuaciones de prueba caen mucho más drásticamente a medida que C aumenta.

Con una exactitud de 99,3 %, nuestro mejor conjunto de hiper parámetro para nuestro modelo de regresión logística es solver='liblinear', penalty='l2' y C=0,1. Si observamos nuestras correlaciones de característica, tiene sentido que L2 será la mejor opción debido a que numerosas características se correlacionan. El mapa de calor de abajo muestra las características altamente correlacionadas como cuadros muy claros u oscuros en las intersecciones de las filas y columnas de esas características. Observamos numerosos cuadros claros y oscuros. Como sabemos, la regularización L2 es útil para modelar conjuntos de datos con multicolinealidad o características altamente correlacionadas.



Resumen:

La regresión logística puede usar una regularización L1 o L2, aunque algunos solver solo pueden usar ciertos tipos. Podemos ajustar la fuerza del término de la regularización al afinar C. Los valores más altos de C son una regularización más débil, y los números más bajos son más fuertes.
Encontrar la cantidad óptima de regularización para los modelos para balancear el sesgo y la varianza es un paso importante en el desarrollo del modelo.