Guía para la obtención de modelos de regresión lineal con Python.

1 Obtención de modelos

Los pasos a seguir para construir un modelo de regresión lineal con Python son los siguientes:

1. Cargar los datos en un DataFrame y preparar las variables predictoras (explicativas) y la variable objetivo. Los datos deben estar limpios y sin valores faltantes. Cargar los datos desde un archivo CSV o EXCEL.

```
data = pd.read_csv('tu_archivo.csv')
data = pd.read_excel('tu_archivo.xlsx')
```

   Una vez cargados los datos se definen las variables predictoras (X) y la variable objetivo (Y)

```
X = data[['Variable1', 'Variable2', ...]]
Y = data['VariableObjetivo']
```

2. Dividir los datos en un conjunto de entrenamiento y un conjunto de prueba para evaluar el modelo.

```
X_train, X_test, Y_train, Y_test =   train_test_split(X, Y, test_size
    =0.2, random_state=1234567)
```

   $train\_test\_split(X, Y, test\_size = 0.2, random\_state = 1234567)$, divide los datos en conjuntos de entrenamiento y prueba de acuerdo con los siguientes parámetros:

   - X y Y son dos conjuntos de datos. X contiene las variables predictoras, mientras que Y contiene la variable objetivo que se quiere predecir.
   - $test\_size = 0.2$ - Establece la proporción del conjunto de prueba en 0.2, lo que significa que el 20% de los datos se utilizarán como conjunto de prueba y el 80% restante se utilizarán como conjunto de entrenamiento.
   - $random\_state = 1234567$ - Establece una semilla para la generación de números aleatorios. Esto asegura que la división de datos sea reproducible. Si se usa la misma semilla en diferentes ejecuciones, se obtendrá la misma división de datos. Esto es útil para la reproducibilidad de los resultados.

   Después de ejecutar la función $train\_test\_split$, se obtienen los siguientes conjuntos de datos:

- X_train.- Conjunto de entrenamiento que contiene las variables predictoras para entrenar el modelo.
- X_test.- Conjunto de prueba que contiene las variables predictoras, pero se utiliza para evaluar el rendimiento del modelo.
- Y_train - Variable objetivo correspondiente al conjunto de entrenamiento.
- Y_test - Variable objetivo correspondiente al conjunto de prueba.

3. Ajustar un modelo de regresión lineal. La función *lm*, definida en *FuncionesMineria*, permite ajustar modelos de regresión lineal en Python, con la capacidad de manejar variables continuas, categóricas e interacciones entre variables.

```
    def lm(varObjCont, datos, var_cont, var_categ, var_interac=[]):
    """
    Ajusta un modelo de regresión lineal a los datos y devuelve
        información relacionada con el modelo.

    Parámetros:
    varObjCont (Series o array): La variable objetivo continua que se
        está tratando de predecir.
    datos (DataFrame): DataFrame de datos que contiene las variables de
         entrada o predictoras.
    var_cont (lista): Lista de nombres de variables continuas.
    var_categ (lista): Lista de nombres de variables categóricas.
    var_interac (lista, opcional): Lista de pares de variables para la
        interacción (por defecto es una lista vacía).

    Returns:
    dict: Un diccionario que contiene información relacionada con el
        modelo ajustado, incluyendo el modelo en sí,
          las listas de variables continuas y categóricas, las
              variables de interacción (si se especifican)
          y el DataFrame X utilizado para realizar el modelo.

    """
    # Prepara los datos para el modelo, incluyendo la dummificación de
        variables categóricas y la creación de interacciones.
    datos = crear_data_modelo(datos, var_cont, var_categ, var_interac)

    # Ajusta un modelo de regresión lineal a los datos y almacena la
        información del modelo en 'Modelo'.
    output = {
        'Modelo': sm.OLS(varObjCont, sm.add_constant(datos)).fit(),
        'Variables': {
            'cont': var_cont,
            'categ': var_categ,
            'inter': var_interac
        },
        'X': datos
    }
```

```
    return output
```

Para crear los datos de entrada al modelo se utiliza la función propia *crear_data_modelo*. Esta función convierte las variables categóricas en *dummies*. Para evitar la colinealidad perfecta entre las variables *dummies*, se elimina una de las categorías para evitar multicolinealidad, concretamente la primera (ordenadas alfabéticamente o numéricamente) que será considerada como categoría de referencia. Además, genera *interacciones* entre las variables seleccionadas. Finalmente, el conjunto de datos contiene las variables continuas, las variables categóricas que se han convertido en *dummies* y las *interacciones* entre las variables seleccionadas.

La función *sm.OLS* de la biblioteca *Statsmodels* ajusta un modelo de regresión lineal. Los argumentos de esta función son *varObjCont* como la variable objetivo y *sm.add_constant*(*datos*) que incluye tanto las variables predictoras como la constante al modelo de regresión lineal. Un ejemplo de un modelo a ajustar sería el siguiente:

```
var_cont = ['X1', 'X2']
var_categ = ['X3', 'X4', 'X5']
var_interac = [('X4', 'X5')]
modelo = lm(y_train, x_train, var_cont, var_categ, var_interac)
```

Suponiendo que la variable X_3 tiene tres categorías, la variable X_4 tiene dos categorías y la variable X_5 tiene dos categorías. El modelo de regresión lineal que ajustaría sería el siguiente:

$$Y = \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3^2 + \beta_4 X_3^3 + \beta_5 X_4^2 + \beta_6 X_5^2 + \beta_7 X_4^2 X_5^2.$$

Donde el subíndice hace referencia a la variable y el súper-índice hace referencia a la categoría correspondiente de esa variable. Por tanto, para las variables categóricas hay tantos parámetros como categorías tenga la variable menos una, que será la correspondiente a la categoría de referencia. Si se ha considerado que la categoría de referencia es la primera, la variable X_3 tiene dos parámetros, el correspondiente a la categoría 2 y el correspondiente a la categoría 3. De igual forma se ha procedido para las variables X_4, X_5 y la interacción.

Si se quiere obtener información acerca del contraste general de regresión, de los contrastes de hipótesis sobre los parámetros, y los estadísticos R^2 y $R^2\_adjustado$, debemos utilizar la función *summary*(): $modelo['Modelo'].summary()$. Por ejemplo, para el conjunto de datos de características del vino se tiene:

```
var_cont = []
var_categ = ['Etiqueta', 'CalifProductor', 'Clasificacion']
modelo = lm(y_train, x_train, var_cont, var_categ)
modelo['Modelo'].summary()
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.472
Model:                            OLS   Adj. R-squared:                  0.471
Method:                 Least Squares   F-statistic:                     378.5
Date:                Sun, 22 Oct 2023   Prob (F-statistic):               0.00
Time:                        11:56:27   Log-Likelihood:                -34787.
No. Observations:                5092   AIC:                         6.960e+04
Df Residuals:                    5079   BIC:                         6.968e+04
Df Model:                          12
Covariance Type:            nonrobust
================================================================================================
                                   coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------------------------
const                          527.0140     13.291     39.652      0.000     500.958     553.070
Etiqueta_M                    -148.0691      9.388    -15.772      0.000    -166.473    -129.665
Etiqueta_MB                     78.8454     17.882      4.409      0.000      43.789     113.902
Etiqueta_MM                   -206.4782     17.366    -11.890      0.000    -240.524    -172.432
Etiqueta_R                     -86.2992      8.216    -10.503      0.000    -102.407     -70.192
CalifProductor_2               -20.6327     11.059     -1.866      0.062     -42.314       1.048
CalifProductor_3               -34.4671     11.290     -3.053      0.002     -56.601     -12.333
CalifProductor_4               -89.9824     13.749     -6.545      0.000    -116.936     -63.028
CalifProductor_5-12           -177.7691     14.924    -11.912      0.000    -207.026    -148.512
Clasificacion_**               154.4193      8.872     17.404      0.000     137.025     171.813
Clasificacion_***              244.9226     10.238     23.923      0.000     224.852     264.993
Clasificacion_****             348.6433     16.001     21.789      0.000     317.275     380.012
Clasificacion_Desconocido     -195.1680      8.933    -21.848      0.000    -212.681    -177.655
==============================================================================
Omnibus:                       69.312   Durbin-Watson:                   1.953
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               71.837
Skew:                           0.289   Prob(JB):                     2.52e-16
Kurtosis:                       3.065   Cond. No.                         10.3
==============================================================================
```

Figure 1: Resumen función lm()

Se tiene la siguiente salida, como se puede observar en la Figura 1 los objetos de tipo *lm* contienen toda la información del modelo. En particular, podemos destacar: $modelo['Modelo'].params$ contiene los coeficientes estimados; $modelo['Modelo'].resid$, contiene los residuos y $modelo['Modelo'].predict(datos\_nuevos)$ contiene los valores predichos para la variable objetivo a partir del modelo ajustado con las variables explicativas contenidas en el conjunto de datos $'datos\_nuevos'$. Por tanto, permite obtener predicciones para un nuevo conjunto de datos. Este nuevo conjunto de datos debe contener exactamente las mismas variables explicativas que han sido usadas en la estimación del modelo, es decir, si hay variables categóricas entre las variables explicativas se deben crear variables *dummy* y si hay *interacciones* entre las variables también deben ser creadas. Esto se realiza con la función propia $'crear\_data\_modelo'$ creada en el fichero $'FuncionesMineria'$. Concretamente, $'datos\_nuevos'$ se genera de la siguiente forma:

```
1    datos_nuevos = crear_data_modelo(x_test, var_cont, var_categ,
         var_interac)
```

Python también permite el cálculo de los índices *AIC* y *BIC* con las siguientes funciones $modelo['Modelo'].aic$ y $modelo['Modelo'].bic$, respectivamente.

A la hora de evaluar un modelo de regresión, es importante hacerse una idea de la importancia de las variables que lo componen. Para ello, una alternativa es calcular como se ve afectado el mismo al eliminar cada una de las variables individualmente. De esta manera, las variables más importantes harían que el modelo empeore mucho al eliminarlas, mientras que las variables menos útiles apenas tendrían efecto sobre la calidad del mismo. Para obtener dichos resultados, podemos utilizar la función $modelEffectSizes$. Esta función calcula la disminución en el R^2 al eliminar las distintas variables y muestra un gráfico que contiene dicha información.

```
modelEffectSizes(modelo, y_train, x_train, var_cont, var_categ, var_interac
    )
```

Por último, es importante conocer siempre el número de parámetros que componen el modelo para tener una idea de la complejidad del mismo. Una manera rápida de obtenerlo es contando el número de parámetros que contiene: $len(modelo['Modelo'].params)$

2 Selección de variables

En Python podemos llevar a cabo los métodos de selección de variables ya estudiados: hacia delante (forward), hacia atrás (backward) y paso a paso (stepwise).

En cada iteración de los métodos se evalúa el AIC/BIC del modelo resultante. De esa forma, se incluyen o eliminan variables siempre y cuando mejore dicho estadístico. Para poder llevar a cabo dicha selección, es necesario que indiquemos el rango de los modelos a evaluar, es decir, el modelo con el máximo número de variables y el modelo con el mínimo número de variables. Este proceso se realiza a partir de las siguientes funciones:

```
lm_backward(varObjCont, datos, var_cont, var_categ, var_interac = [],
    metodo = 'AIC')
```

Esta función parte del modelo con todas las variables indicadas ($var\_cont, var\_categ, var\_interac$) y elimina variables en cada iteración según el criterio AIC. Pudiendo llegar, como mínimo, al modelo con ninguna variable. Si en lugar de usar el criterio AIC se utiliza el criterio BIC en la función únicamente hay que cambiar el método ($metodo =' BIC'$).

```
lm_forward(varObjCont, datos, var_cont, var_categ, var_interac = [], metodo
    = 'AIC')
```

Esta función parte del modelo sin ninguna variable y va añadiendo variables en cada iteración según el criterio AIC. Pudiendo llegar como máximo, al modelo con todas las variables indicadas ($var\_cont, var\_categ, var\_interac$). Si en lugar de usar el criterio AIC se utiliza el criterio BIC en la función únicamente hay que cambiar el método ($metodo =' BIC'$).

```
lm_stepwise(varObjCont, datos, var_cont, var_categ, var_interac, metodo = '
    AIC')
```

Esta función parte del modelo sin ninguna variable y va añadiendo variables en cada iteración según el criterio AIC. La diferencia con el método forward es que si una variable ha sido añadida en sucesivas iteraciones puede volver a ser eliminada. Pudiendo llegar como máximo, al modelo con todas las variables indicadas ($var\_cont, var\_categ, var\_interac$) y como mínimo, al modelo sin

ninguna de ellas. Si en lugar de usar el criterio AIC se utiliza el criterio BIC en la función únicamente hay que cambiar el método ($metodo =' BIC'$).

2.1 Lista de todas las posibles interacciones

De cara a considerar todos los posibles efectos que puedan aportar información sobre la variable objetivo, sería interesante poder generar automáticamente una fórmula que los contenga todos para poderla usar en el proceso de selección de variables o de búsqueda exhaustiva. Para ello, podemos crear en Python diferentes formas de crear interacciones, en función de nuestros datos y objetivos.

No obstante, siempre que añadamos interacciones tenemos que tener en cuenta nuestras limitaciones de computación y tiempo, pues es común en esta parte del proceso de predicción encontrar con dichas limitaciones.

A continuación se muestran diferentes alternativas para crear interacciones.

- Queremos todas las posibles combinaciones n a n de una única lista con variables

```
#======Iteraciones n a n de unalista==================
 interacciones_unicas = list(itertools.combinations(list1, n)) #
#=====================================================
```

- Si tenemos dos listas diferentes de variables, por ejemplo por un lado las variables continuas y por otro las variables categóricas, nos podría interesar tener todas las posibles interacciones resultantes de combinar los elementos de la primera lista con los de la segunda.

```
# == Interacciones entre 2 listas de variables distintas ======
# Genero interacciones:
 interacciones = list(itertools.product(list1, list2))
# itertools.product genera duplicados
# eliminamos duplicados
 interacciones_unicas = []
for x in interacciones:
    if (sorted(x) not in [sorted(t) for t in interacciones_unicas]) and
         (x[0] != x[1]):
        interacciones_unicas.append(x)
#=============================================================
```

Recordamos que para unir dos listas diferentes en python, hay que hacer simplemente lo siguiente:

```
    list_total = list1 + list2
```

3 Comparación de modelos a partir de Training-Test

Como ya hemos visto, una de las mejores formas de comparar modelos ya construidos es a través de un conjunto de datos test, que no forme parte de la construcción del modelo y que nos permita obtener una estimación insesgada del funcionamiento del mismo. En primer lugar, se dividen los

datos en un conjunto de datos de entrenamiento y un conjunto de datos test tal y como se ha indicado en el apartado 1.**Obtención del Modelo**, paso 2.

Una vez obtenidos estos subconjuntos, deberemos obtener los modelos únicamente con el conjunto de datos de entrenamiento y después evaluarlos y decidir el mejor de entre ellos a partir del conjunto de datos de prueba (Test). Para ello, debemos predecir, utilizando el modelo obtenido con los datos de entrenamiento, la variable objetivo para las observaciones del conjunto de datos Test y comparar con los valores reales (y_test).

Para medir la bondad del ajuste de los modelos obtenidos se calcula el valor de R^2 para train y para test con la siguiente función:

```
    def Rsq(modelo, varObj, datos):
    """
    Calcula el coeficiente de determinación (R-squared) de un modelo de
        regresión lineal.

    Parámetros:
    modelo (RegressionResultsWrapper): El modelo de regresión lineal
        ajustado.
    varObj (Series o array): La variable objetivo numérica (variable a
        predecir).
    datos (DataFrame): El DataFrame de datos utilizado para ajustar el
        modelo.

    Returns:
    float: El coeficiente de determinación (R-squared) del modelo.
    """

    # Selecciona las variables independientes del modelo
    datos = datos[modelo.model.exog_names[1:]]

    # Realiza predicciones utilizando el modelo
    testpredicted = modelo.predict(sm.add_constant(datos))

    # Calcula la suma de los cuadrados de los errores (SSE)
    sse = sum((testpredicted - varObj) ** 2)

    # Calcula la suma total de los cuadrados (SST)
    sst = sum((varObj - varObj.mean()) ** 2)

    # Calcula y devuelve el coeficiente de determinación (R-squared)
    return 1 - sse / sst

# Calculamos la medida de ajuste R^2 para los datos de entrenamiento
Rsq(modelo['Modelo'], y_train, modelo['X'])
# Preparamos los datos test para usar en el modelo
x_test_modelo = crear_data_modelo(x_test, var_cont, var_categ, var_interac)
# Calculamos la medida de ajuste R^2 para los datos test
Rsq(modelo['Modelo'], y_test, x_test_modelo)
```

Es importante no olvidar dos cosas:

- El valor del R^2 obtenido en test se trata de una estimación más realista del comportamiento futuro del modelo y, por tanto, nos permitirá hacernos una idea sobre como de bueno o malo será el modelo aplicado a datos futuros.
- La diferencia en los valores de R^2 en train y test nos permite saber cómo de estable es el modelo (cuando los valores se parecen, es esperable que para cualquier conjunto de datos futuro esto sea también así), así como de la posible presencia de sobreajuste (variables que mejoran el modelo en train, no así en test).

4 Comparación de modelos a partir de validación cruzada

Este método consiste en dividir el conjunto de datos en submuestras e iterativamente construir el modelo con todas las observaciones menos las de una submuestra y evaluarlo a continuación con las observaciones de dicha submuestra excluida.

Nótese que cada ejecución de validación cruzada da lugar a una única predicción de todas las observaciones por lo que se puede obtener la SSE correspondiente. Para evitar trabajar con cantidades tan grandes, la comparación de modelos se suele llevar a cabo a partir del R^2. La función que nos permitirá realizar este proceso de validación cruzada es la siguiente:

```
    def validacion_cruzada_lm(n_cv, datos, varObjCont, var_cont, var_categ,
        var_interac=[]):
    """
    Realiza la validación cruzada de un modelo de regresión lineal y
        devuelve una lista de puntajes R-squared.

    Parámetros:
    n_cv (int): El número de divisiones para la validación cruzada (k-fold)
        .
    datos (DataFrame): El DataFrame de datos que contiene las variables de
        entrada.
    varObjCont (Series o array): La variable objetivo continua que se está
        tratando de predecir.
    var_cont (lista): Lista de nombres de variables continuas.
    var_categ (lista): Lista de nombres de variables categóricas.
    var_interac (lista, opcional): Lista de pares de variables para la
        interacción (por defecto es una lista vacía).

    Returns:
    list: Una lista de puntajes R-squared obtenidos en cada fold de la
        validación cruzada.
    """

    # Prepara los datos para el modelo, incluyendo la codificación de
        variables categóricas y la creación de interacciones.
    datos = crear_data_modelo(datos, var_cont, var_categ, var_interac)

```

```
    # Realiza la validación cruzada utilizando un modelo de regresión lineal y puntajes R-squared.
    return list(cross_val_score(LinearRegression(), datos, varObjCont, cv=n_cv, scoring='r2'))
```

Hay que indicar el número de grupos de la validación cruzada (n_cv), el conjunto de datos, la variable objetivo, las variables continuas, categóricas e interacciones del modelo que se ha estimado.

Con el objetivo de obtener una evaluación más robusta y confiable del rendimiento de un modelo, en lugar de realizar este proceso de validación cruzada solo una vez, se repite el proceso múltiples veces. Cada repetición implica una nueva división del conjunto de datos en submuestras e iterativamente construir el modelo con todas las observaciones menos las de una submuestra y evaluarlo a continuación con las observaciones de dicha submuestra excluida. A este método se le denomina validación cruzada repetida.

Una forma de interpretar fácilmente los últimos resultados es construir un diagrama de cajas con todos los R^2 obtenidos al repetir el proceso de validación cruzada. Si se representan estos diagramas de cajas para distintos modelos sobre la misma escala, podremos concluir que modelo es preferible sobre el resto.

```
# Validacion cruzada repetida para ver que modelo es mejor
# Crea un DataFrame vacío para almacenar resultados
results = pd.DataFrame({
    'Rsquared': [],
    'Resample': [],
    'Modelo': []
})

# Realiza el siguiente proceso 20 veces (representado por el bucle `for rep in range(20)`)
for rep in range(20):
    # Realiza validación cruzada en cuatro modelos diferentes y almacena sus R-squared en listas separadas
    modelo1VC = validacion_cruzada_lm(5, x_train, y_train, var_cont1, var_categ1)
    modelo2VC = validacion_cruzada_lm(5, x_train, y_train, var_cont2, var_categ2)
    modelo3VC = validacion_cruzada_lm(5, x_train, y_train, var_cont3, var_categ3)
    modelo4VC = validacion_cruzada_lm(5, x_train, y_train, var_cont4, var_categ4, var_interac4)

    # Crea un DataFrame con los resultados de validación cruzada para esta repetición
    results_rep = pd.DataFrame({
        'Rsquared': modelo1VC + modelo2VC + modelo3VC + modelo4VC,
        'Resample': ['Rep' + str((rep + 1))] * 5 * 4,  # Etiqueta de repetición
        'Modelo': [1] * 5 + [2] * 5 + [3] * 5 + [4] * 5  # Etiqueta de modelo (1, 2, 3 o 4)
```

```
    })

    # Concatena los resultados de esta repetición al DataFrame principal 'results'
    results = pd.concat([results, results_rep], axis=0)


# Boxplot de la validación cruzada
plt.figure(figsize=(10, 6))  # Crea una figura de dimensiones 10x6
plt.grid(True)  # Activa la cuadrícula en el gráfico
# Agrupa los valores de R-squared por modelo
grupo_metrica = results.groupby('Modelo')['Rsquared']
# Organiza los valores de R-squared por grupo en una lista
boxplot_data = [grupo_metrica.get_group(grupo).tolist() for grupo in grupo_metrica.groups]
# Crea un boxplot con los datos organizados
plt.boxplot(boxplot_data, labels=grupo_metrica.groups.keys())  # Etiqueta los grupos en el boxplot
# Etiqueta los ejes del gráfico
plt.xlabel('Modelo')  # Etiqueta del eje x
plt.ylabel('Rsquared')  # Etiqueta del eje y
plt.show()  # Muestra el gráfico
```

Para el ejemplo de las características del vino se ha utilizado la siguiente sentencia y obtenido los resultados que aparecen en la Figura 2.

```
var_cont1 = ['Acidez', 'AcidoCitrico', 'Azucar', 'CloruroSodico', 'Densidad', 'pH', 'Sulfatos',
             'Alcohol', 'PrecioBotella']
var_categ1 = ['Etiqueta', 'CalifProductor', 'Clasificacion', 'Region', 'prop_missings']

var_cont2 = []
var_categ2 = ['Etiqueta', 'CalifProductor', 'Clasificacion','prop_missings']

var_cont3 = []
var_categ3 = ['Etiqueta', 'CalifProductor', 'Clasificacion']

var_cont4 = []
var_categ4 = ['Etiqueta', 'CalifProductor', 'Clasificacion']
var_interac4 = [('Clasificacion', 'Etiqueta')]
```

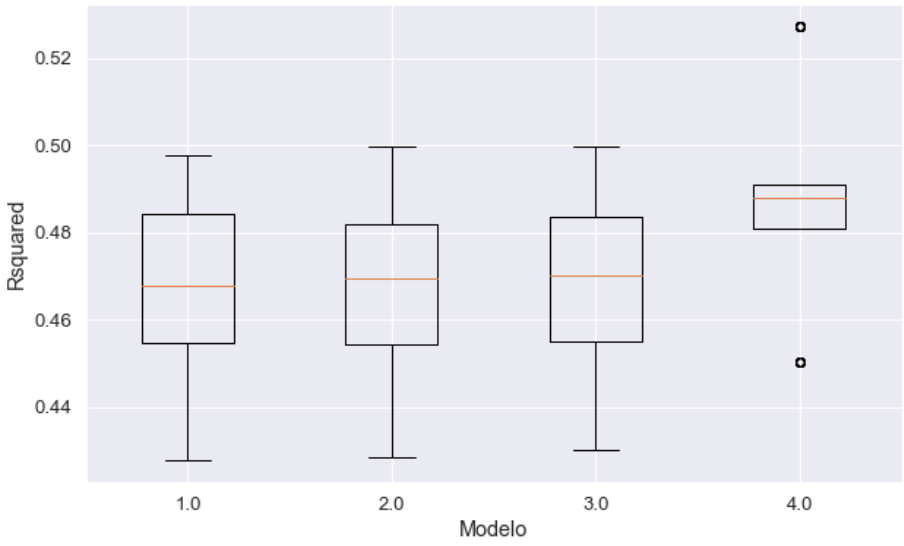

Figure 2: Gráfico de barras y bigotes para validación cruzada repetida

Como se puede comprobar en la Figura 2, el mejor modelo es el número 4 en términos de "bondad media". Se puede observar que la variabilidad de los modelos 1, 2 y 3 es muy parecida. Otra ventaja de estos gráficos es que permiten evaluar fácilmente la mejora conseguida al incluir/eliminar factores. Por ejemplo, la diferencia entre los dos primeros modelos y el tercero pone de manifiesto que el efecto de las variables continuas de los dos primeros modelos no es significativo. Por el contrario, si comparamos el tercero y el cuarto, que se diferencian en la inclusión de la interacción entre las variables $'Clasificacion'$ y $'Etiqueta'$, comprobamos que dicha interacción si aporta información al modelo.

En ocasiones, no obstante, no queda claro visualmente que modelo es preferible. Para esos casos, podemos calcular la media y desviación típica de los R^2 de las repeticiones de la validación cruzada como complemento a los diagramas de cajas anteriores, utilizando el siguiente código:

```
# Calcular la media de las métricas R-squared por modelo
media_r2 = results.groupby('Modelo')['Rsquared'].mean()
# Calcular la desviación estándar de las métricas R-squared por modelo
std_r2 = results.groupby('Modelo')['Rsquared'].std()
```