lab_1

June 24, 2019

- Miguel Vargas (lvc0107@protonmail.com)

1 Laboratorio 1: Conceptos básicos de aprendizaje automático

En este laboratorio les tocará probar con distintos parámetros de los algoritmos de aprendizaje automático aprendidos hasta ahora. La idea es que vean como la selección de atributos, el cambio de hiperparámetros, y los distintos algoritmos afectan los resultados de un regresor o clasificador sobre un conjunto de datos.

```
In [1]: import matplotlib.pyplot as plt
        import numpy as np

        from matplotlib.colors import ListedColormap
        from sklearn.datasets import load_boston, load_breast_cancer, load_iris
        from sklearn.linear_model import LinearRegression, LogisticRegression, Perceptron, Rid
        from sklearn.metrics import accuracy_score, confusion_matrix, mean_squared_error
        from sklearn.neighbors import KNeighborsClassifier
        from sklearn.preprocessing import PolynomialFeatures

        from ml.visualization import plot_confusion_matrix, classifier_boundary

        np.random.seed(1234)  # Setup seed to be more deterministic

        %matplotlib inline
```

1.1 Regresión

1.1.1 Carga de datos

```
In [2]: boston_data = load_boston()
        len(boston_data['data'])

Out[2]: 506
```

Como el dataset de boston tiene 506 muestras, generamos 506 indices aleatoriamente para luego obtener dos subsets de 80% y 20% del dataset desordenados

```
In [3]: np.random.permutation(506)
```

```
Out[3]: array([ 64, 100, 400, 485, 454, 288, 112, 478,  66, 187, 474,  71, 205,
               133, 247, 333, 444, 207, 393, 398, 200, 352, 363,   7,  92,  59,
               358, 213, 349, 124,  11,  27, 131, 441, 329, 409, 265, 319, 387,
               417, 404,  29, 280, 307, 421, 301, 351, 268,  22, 285, 362, 304,
               160,  73, 369,  57, 405, 341, 353, 480,  40, 448, 111, 192, 403,
                67, 264, 310, 166, 224, 156, 321, 477, 218,  70, 395, 225, 355,
               123, 138, 226, 262,  74, 129, 181, 406, 296, 252,  13,  44,  25,
               491,  91, 496, 149, 228, 424, 350, 191, 239, 109, 219, 221, 375,
               450, 163, 402, 378,   0, 315, 437, 194, 486, 342, 148,  51, 189,
               347, 141,  97,  78, 391,  21,  55, 104, 253,  43, 255, 115, 230,
               122, 169, 427, 386, 366, 373,  24, 306, 411, 263, 118, 443, 305,
               492, 300, 234,  90, 314, 287, 384, 108, 338,  99, 102, 416, 266,
               101, 501, 426,  33, 140, 198,  54, 447, 432,  36, 298,  37, 215,
               323, 299, 494, 493, 408, 308, 388, 418, 214, 270, 498, 110, 483,
                69, 229, 324, 153, 344, 337, 433, 348, 475, 309, 220, 451, 463,
                42, 466,  58, 473, 407,  20, 188, 394, 173, 168,  52, 425, 327,
               144, 167, 176, 146, 134, 410, 273, 132, 185, 206, 278,   6, 436,
               186, 376,  23, 468, 413, 223, 222, 499, 261, 439, 502, 162, 467,
               202, 125,  95,  31,  17, 330,  80, 445,   9, 157, 193, 203, 289,
               248, 254, 381,  93, 479, 180,  77, 453, 179, 297, 461, 382, 446,
               271, 216, 356, 302, 227, 455, 184,  72, 335,   5, 464,  63,  94,
               360, 357, 127, 462, 260, 471, 199, 106, 178,  60, 137, 245,  83,
               397,  48,  32, 414, 209, 489, 415, 283, 217,  65, 232, 364, 390,
               488,  35, 147, 312, 482, 182,  38,  87, 322, 165, 210, 295,  39,
               435,  19, 385, 419, 237, 281, 429, 392,  12,  56,  98, 470, 379,
               145, 190, 267, 257,  28, 317, 438, 164, 484, 155, 339, 318,  49,
               359, 504, 126,  61, 272, 276, 367,  75,   8, 208, 458, 117, 332,
               274, 291, 212,   2, 170, 161,  88, 292, 313,  46, 420,  79, 242,
               130, 452, 434,  15, 465, 241, 172, 136, 150, 320,  68,  26, 460,
               423, 201, 286, 259, 142,   1, 114, 361, 500, 250, 119, 196, 277,
               377, 370, 346, 401, 354, 121, 256, 105, 340, 183,  62, 440, 151,
               334, 235, 311, 345, 103,  82,   4, 284,  10,  34, 399, 175, 487,
               244,  47, 238, 389, 469, 135, 503, 365, 472, 174,  85,  18, 497,
               505, 428,  86, 326, 490, 328,  41,  16,  45,  84,  89, 249, 159,
               113, 197, 459,  96, 116, 269, 343,  81, 412, 430, 456, 316, 243,
                14, 107, 371, 449, 396, 275, 258,   3, 481, 331, 495, 128, 139,
               231, 195, 422, 290, 293,  76, 251, 240,  50, 431, 246, 383, 457,
               336, 325, 476, 368, 120, 442, 374, 282, 380, 236, 158, 171,  30,
               154, 233, 279, 177, 143, 152, 372, 204,  53, 294, 211, 303])
```

```
In [4]: # Utilizamos aproximadamente 80% de los datos para entrenamiento y 20% para validación
        shuff_data = np.random.permutation(506)
        shuff_train = shuff_data[:400]
        shuff_val = shuff_data[400:]

        X_train = boston_data['data'][shuff_train]
        X_val = boston_data['data'][shuff_val]
```

```
y_train = boston_data['target'][shuff_train]
y_val = boston_data['target'][shuff_val]

# Necesario para poder hacer un regresor por feature
feature_map = {feature: idx for idx, feature in enumerate(boston_data['feature_names'])
feature_map
```

Out[4]:
```
{'CRIM': 0,
 'ZN': 1,
 'INDUS': 2,
 'CHAS': 3,
 'NOX': 4,
 'RM': 5,
 'AGE': 6,
 'DIS': 7,
 'RAD': 8,
 'TAX': 9,
 'PTRATIO': 10,
 'B': 11,
 'LSTAT': 12}
```

In [5]:
```
print(boston_data['DESCR'])
```

```
.. _boston_dataset:

Boston house prices dataset
---------------------------

**Data Set Characteristics:**

    :Number of Instances: 506

    :Number of Attributes: 13 numeric/categorical predictive. Median Value (attribute 14) is us

    :Attribute Information (in order):
        - CRIM     per capita crime rate by town
        - ZN       proportion of residential land zoned for lots over 25,000 sq.ft.
        - INDUS    proportion of non-retail business acres per town
        - CHAS     Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
        - NOX      nitric oxides concentration (parts per 10 million)
        - RM       average number of rooms per dwelling
        - AGE      proportion of owner-occupied units built prior to 1940
        - DIS      weighted distances to five Boston employment centres
        - RAD      index of accessibility to radial highways
        - TAX      full-value property-tax rate per $10,000
        - PTRATIO  pupil-teacher ratio by town
        - B        1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
        - LSTAT    % lower status of the population
```

```
        - MEDV     Median value of owner-occupied homes in $1000's

    :Missing Attribute Values: None

    :Creator: Harrison, D. and Rubinfeld, D.L.

This is a copy of UCI ML housing dataset.
https://archive.ics.uci.edu/ml/machine-learning-databases/housing/


This dataset was taken from the StatLib library which is maintained at Carnegie Mellon Univers

The Boston house-price data of Harrison, D. and Rubinfeld, D.L. 'Hedonic
prices and the demand for clean air', J. Environ. Economics & Management,
vol.5, 81-102, 1978.   Used in Belsley, Kuh & Welsch, 'Regression diagnostics
...', Wiley, 1980.   N.B. Various transformations are used in the table on
pages 244-261 of the latter.

The Boston house-price data has been used in many machine learning papers that address regress
problems.

.. topic:: References

   - Belsley, Kuh & Welsch, 'Regression diagnostics: Identifying Influential Data and Sources
   - Quinlan,R. (1993). Combining Instance-Based and Model-Based Learning. In Proceedings on t
```

1.1.2 Regresión sin regularización

Para revisar cómo afecta el cambio de parámetros y los distintos tipos de regresores y atributos (características) al resultado final del algoritmo de aprendizaje automático, lo que se va a hacer es entrenar el regresor tomando sólo un atributo y visualizar eso.

Se busca entrenar utilizando el conjunto de entrenamiento (el terminado en `train`) y evaluar utilizando el conjunto de validación (el terminado en `val`). Luego se visualiza la función calculada para cada conjunto y se la compara.

Los atributos posibles están listados en la descripción del conjunto de datos en la celda anterior. No todos son útiles para visualizar, en particular solo nos interesan los atributos numéricos y descartamos los atributos que se listan a continuación:

- `CHAS`: Atributo categórico (toma valor 0 o 1).
- `RAD`: Atributo categórico (índice).
- `MEDV`: Este valor se lo lista como atributo en la descripción del conjunto de datos pero en realidad es el valor de y, i.e. es el valor que tratamos de aproximar con el algoritmo de aprendizaje automático.

In [6]:
```
# Seleccionamos un atributo de los listados en la descripción que no sea categórico
selected_feature = 'AGE'
```

```
In [7]: numeric_features = ['AGE', 'CRIM', 'ZN', 'INDUS', 'NOX', 'RM', 'DIS', 'TAX', 'PTRATIO'
```

```
In [8]: feature_col = feature_map[selected_feature]
        feature_col
```

```
Out[8]: 6
```

```
In [9]: X_train[:, feature_col] # Vemos datos de AGE para entrenamiento
```

```
Out[9]: array([ 29.1,  96.6,  97.4,  67. , 100. ,  30.8,  89. ,  78.1, 100. ,
                45.4,  22.3,  97.2,  79.7,  65.2,  96.7,  58.8,  31.3,  21.8,
                37.8,  91.9,  84.7,  15.8, 100. ,  62.5,  97.9,  45.6,  89.1,
                28.1,  58.5,  38.3,  73.1,  30.2,  98.8, 100. ,   6. ,  40.3,
                66.2, 100. ,  53.2,  31.9,  36.8,  96.4, 100. , 100. ,  33.1,
                87.3, 100. ,  97.3,  95.8,  98.9,  70.6,  29.7,  49.3,  88. ,
                 8.4,  68.2,  65.1,  93. ,  94.7,  54.2,  84.2,  86.3,  63.1,
                92.6,  52.3,  87.9,  17.5,  68.7,  81.3,  97.1,  53.7,   9.8,
                92.9,  33.5,  97.3,  81.8,  90.8,  93.6, 100. ,  92.4,  49.9,
                54.4,  31.5,  98.8,  74.9,  88.5, 100. ,  44.4,  92.6,  15.7,
                21.9,  65.4,  15.3,  85.7,  92.7,  19.1,  32.2,  52.8,  80.3,
                92.9,  34.2,  85.1,  79.9,  97. , 100. ,  53.8, 100. ,  27.6,
               100. , 100. ,  89.5,  88.8,  98.9,  89.2,  95. ,  82.6,  97.4,
               100. ,  98.8,  71.7,  58.7,  86.9,  71.9,  76.5,  94. ,  33.8,
                91.2, 100. ,  91.1,  28.9, 100. ,  98.8,  64.7,  94.1,  77.3,
               100. ,  95.6,   7.8, 100. ,  85.9,  74.5,  24.8,  78.3,  89.9,
                48.5, 100. ,  34.9,  52.6,  98.2,  69.7,  98.4,  90.4,  52.5,
                88.4,  31.9,  94.1,  21.4,  77.7,  98.4,  93.8,  26.3,  68.1,
                63. ,  71.6,  94.7,  96.2,  96.2,  74.4,  40.4,  59.6,  76.7,
                90. ,   7.8,  36.1,  70.4,  91.5,  78.9,  87.3,  91. , 100. ,
                56.5,  27.7,  23.4,  94.5,  42.8,  86.5,  49.1, 100. ,  93.6,
                91.8,  96.8,   2.9,  27.9,  82.9,  70.3,  85.1,  66.5,  42.1,
                67.6,  40.1,  51.8,  76.5,  46.7,  28.9,  91.2,  70.2, 100. ,
               100. ,  42.6,  17.8, 100. ,  22.9,   6.6,  29.2,  66.6,  80.8,
                96.1,  32.2,  67.8,  18.4,  82. , 100. ,  93.4,  52.3,  18.4,
                93.3,  49.7,  36.6,  21.4,  93.5,  77.8,  64.5,  82.9,  93.3,
                86.5,  91. ,  95.4,  72.5,  97.7,  17. ,  65.2,  95.2,  20.8,
                97.9,  45.8,  10. ,  21.9,  46.3,  84.6,  98.9,  16.3,  37.2,
                94.1,  77.7,  72.7,  77. ,  47.2,  92.4,  17.7,  19.5,  72.9,
                42.4,  94.5,  95.6,  83. ,  83.7,  93.9,  95.4,  88. ,  33. ,
                79.9,  42.3,  83.2, 100. ,   8.9,  94.8,  66.1,  61.1, 100. ,
                56.4,  31.1, 100. ,  85.5,  95.7,  54.3, 100. ,  38.9,  94.3,
                49. ,  58. ,  56.1,  69.1,  36.6, 100. ,  98.2,  69.5, 100. ,
                18.8,  18.5,  97.3,  28.8,  40. ,  98.3,  98.7,  41.1,  82.5,
                56. ,  83.4,  69.6,  39. ,  93.8,  32.3,  94.6,  33.2,  20.1,
                96.6,  41.1, 100. ,  90.3, 100. ,   6.6,  97.4,  85.4, 100. ,
                84.1,  76.7,  96. ,  14.7,  89.3,  61.4,  76.9,  59.5,  38.4,
                41.5,  84. ,  59.7,   6.8,  91.4,  71. ,  53.6,  98. ,  96. ,
                85.4,  91.7,  92.6,  45. , 100. ,  96. ,  43.4,  96.7,  94.4,
                18.5,  85.2,   6.2,  84.4,  48. ,  57.8,  95.3,  98.5,  97.9,
```

```
                 84.5,   91.3, 100. ,   34.1,   79.2,   28.4,   35.7, 100. ,   45.1,
                 58.7,   74.8,   21.1,   56.7,   87.4,   62.8,   45.8,   95.3,   96.2,
                100. ,   13.9,   56.8,   91.6,   88.2,   51.9,   73.3,   99.3,   34.5,
                100. ,   95. ,   87.9,   97.9,   21.4,   59.7,   58.1,   70.4,   73.5,
                 79.2,   17.2,   27.7,   21.5])
```

```
In [10]: X_train_feature = X_train[:, feature_col].reshape(-1, 1)  # Hay que ser que sea una m
         type(X_train_feature)
```

```
Out[10]: numpy.ndarray
```

```
In [11]: X_train_feature.shape
         # reshape(-1, 1) fransforma el vector en una matriz de una columna y (-1) filas.
         # -1 significa indeterminado, entonces numpy resuelve que la cantidad de filas es igu
         # en el vector, en este caso 400.
```

```
Out[11]: (400, 1)
```

```
In [12]: X_val_feature = X_val[:, feature_col].reshape(-1, 1)
```

1.1.3 Regresión lineal

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html

```
In [13]: # Entrenamos un clasificador utilizando sólo ese atributo sobre el conjunto de entren
         model = LinearRegression()
         model.fit(X_train_feature, y_train)

         # Evaluamos el desempeño del clasificador utilizando la media del error cuadrado (MSE
         # sobre el conjunto de datos de entrenamiento (X_train, y_train) y lo comparamos con
         # Mientras más cercano a cero mejor
         print('Media del error cuadrado para entrenamiento: %.2f' %
               mean_squared_error(y_train, model.predict(X_train_feature)))
         print('Media del error cuadrado para validación: %.2f' %
               mean_squared_error(y_val, model.predict(X_val_feature)))
```

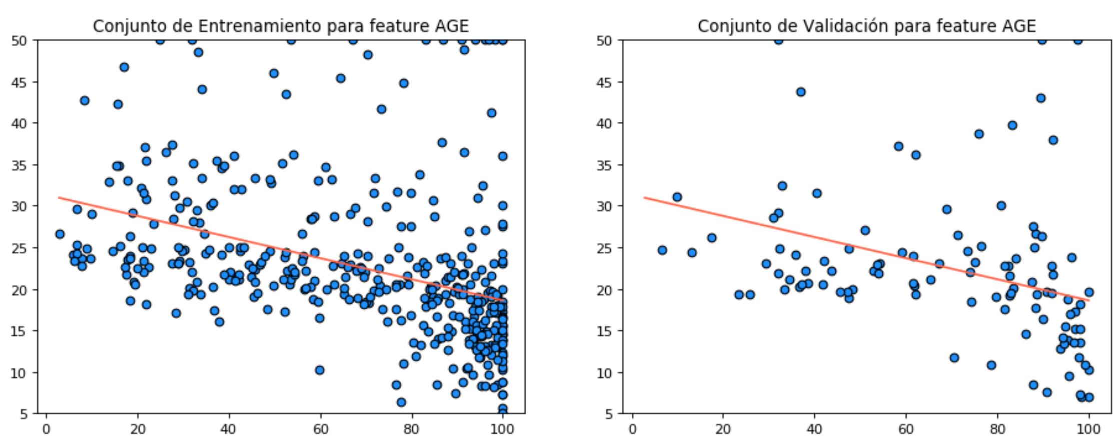
```
Media del error cuadrado para entrenamiento: 73.47
Media del error cuadrado para validación: 68.56
```

Visualización de la regresión lineal

```
In [14]: def plot_lineal_regression(X_train_feature, y_train, X_val_feature, y_val, selected_fe

             plt.figure(figsize=(14, 5), dpi= 80, facecolor='w', edgecolor='k')

             X_range_start = np.min(np.r_[X_train_feature, X_val_feature])
             X_range_stop = np.max(np.r_[X_train_feature, X_val_feature])
             y_range_start = np.min(np.r_[y_train, y_val])
```

```
    y_range_stop = np.max(np.r_[y_train, y_val])
    X_linspace = np.linspace(X_range_start, X_range_stop, 200).reshape(-1, 1)

    # Conjunto de entrenamiento
    plt.subplot(1, 2, 1)
    plt.scatter(X_train_feature, y_train, facecolor="dodgerblue", edgecolor="k", label
    plt.plot(X_linspace, model.predict(X_linspace), color="tomato", label="modelo")
    plt.ylim(y_range_start, y_range_stop)
    plt.title(f"Conjunto de Entrenamiento para feature {selected_feature}")

    # Conjunto de validación
    plt.subplot(1, 2, 2)
    plt.scatter(X_val_feature, y_val, facecolor="dodgerblue", edgecolor="k", label="da
    plt.plot(X_linspace, model.predict(X_linspace), color="tomato", label="modelo")
    plt.ylim(y_range_start, y_range_stop)
    plt.title(f"Conjunto de Validación para feature {selected_feature}")

    plt.show()

    mean_squared_training_error = mean_squared_error(y_train, model.predict(X_train_fe
    print(f'Media del error cuadrado para entrenamiento del feature {selected_feature}

    mean_squared_val_error = mean_squared_error(y_val, model.predict(X_val_feature))
    print(f'Media del error cuadrado para validación del feature {selected_feature}
```

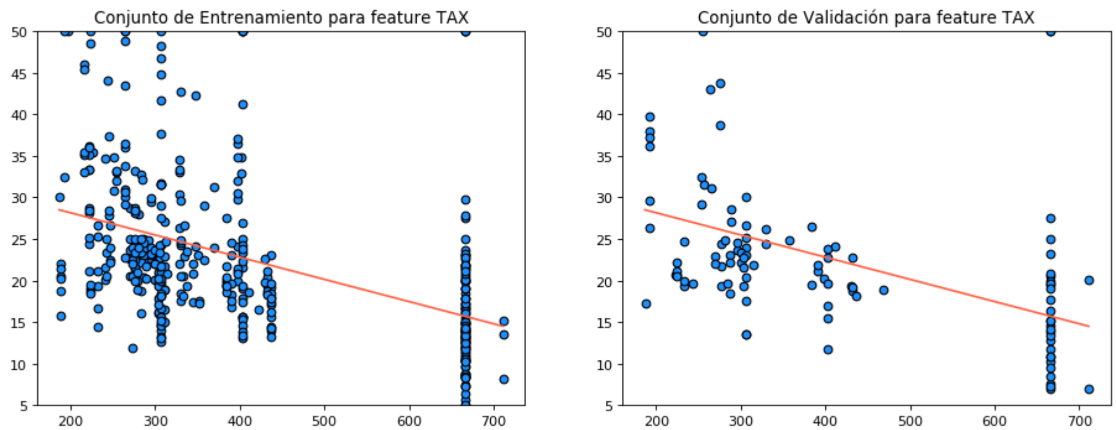
In [15]:
```
plot_lineal_regression(X_train_feature, y_train, X_val_feature, y_val, selected_featu
```



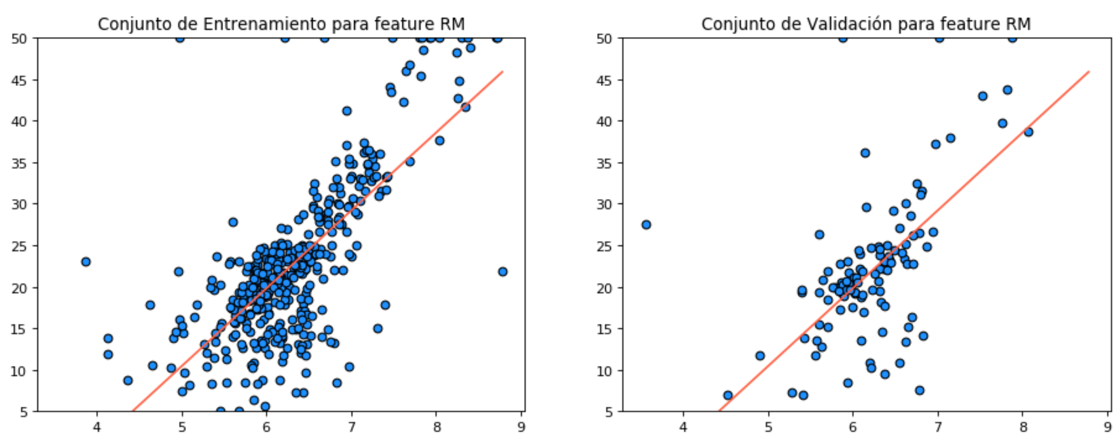

```
Media del error cuadrado para entrenamiento del feature AGE : 73.47
Media del error cuadrado para validación del feature AGE    : 68.56
```

El scatter plot muestra que el feature **AGE** tiene datos demasiado dispersos y que en principio no se ajusta bien con un polinomio de grado 1 (recta). No parece que se puedan ajustar bien tampoco con una regression polinomial. Analizamos el resto de los features:

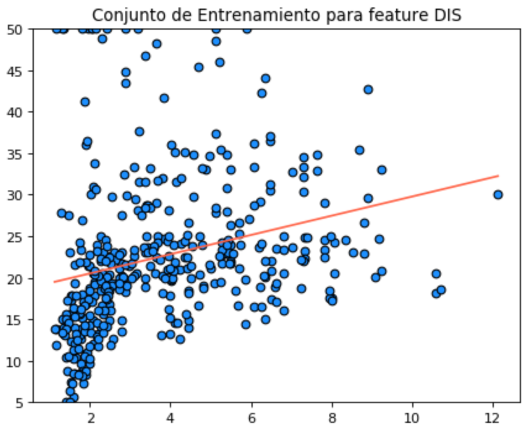
```
In [16]: for selected_feature in set(numeric_features) - {'AGE'}:
             feature_col = feature_map[selected_feature]
             X_train_feature = X_train[:, feature_col].reshape(-1, 1)  # Hay que haceer que se
             X_val_feature = X_val[:, feature_col].reshape(-1, 1)

             model = LinearRegression()
             model.fit(X_train_feature, y_train)

             plot_lineal_regression(X_train_feature, y_train, X_val_feature, y_val, selected_fe
```

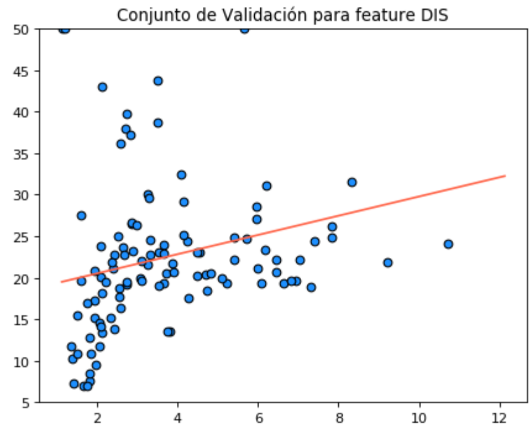



```
Media del error cuadrado para entrenamiento del feature TAX : 67.09
Media del error cuadrado para validación del feature TAX    : 61.54
```

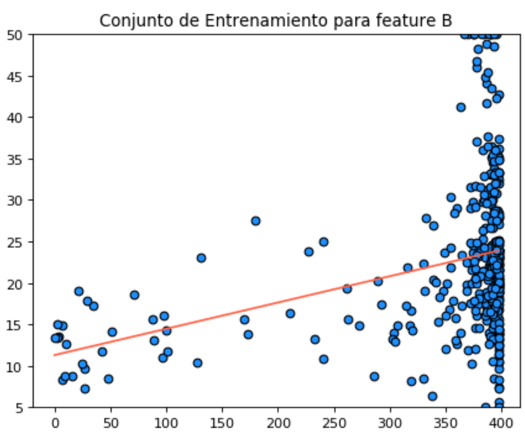


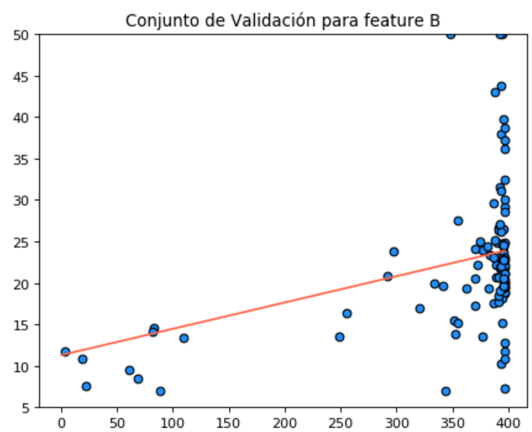
```
Media del error cuadrado para entrenamiento del feature RM : 41.17
Media del error cuadrado para validación del feature RM    : 53.02
```



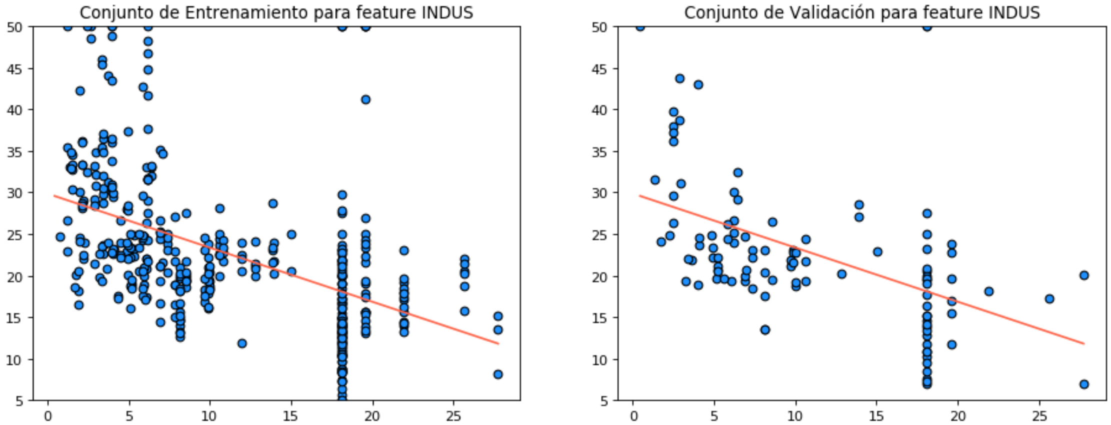



```
Media del error cuadrado para entrenamiento del feature DIS : 80.67
Media del error cuadrado para validación del feature DIS    : 73.50
```

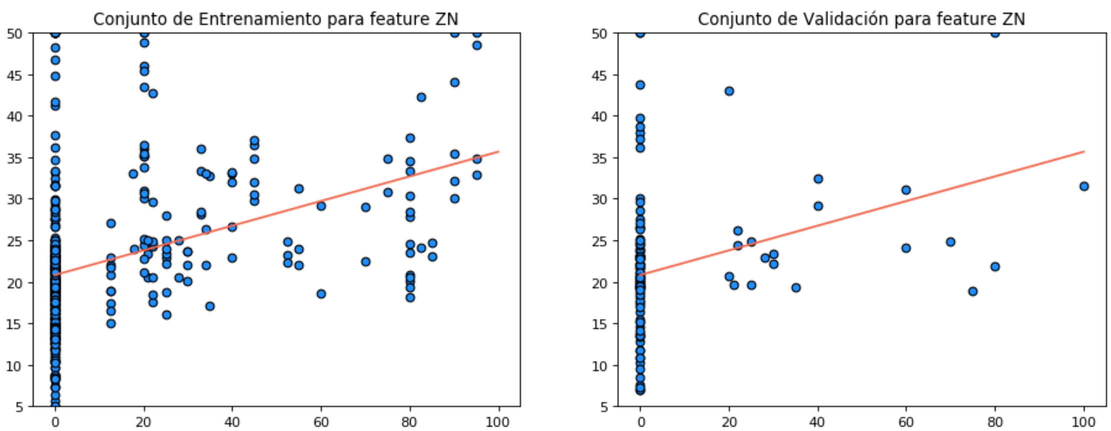




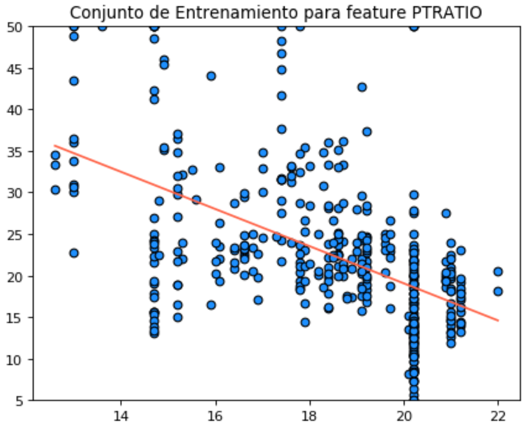

```
Media del error cuadrado para entrenamiento del feature B : 78.60
Media del error cuadrado para validación del feature B    : 61.73
```

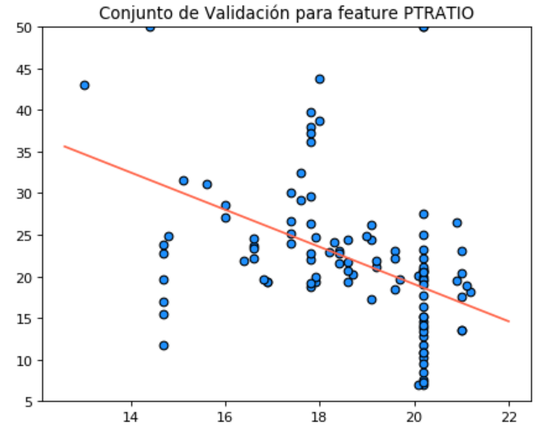



```
Media del error cuadrado para entrenamiento del feature INDUS : 66.81
Media del error cuadrado para validación del feature INDUS    : 56.63
```

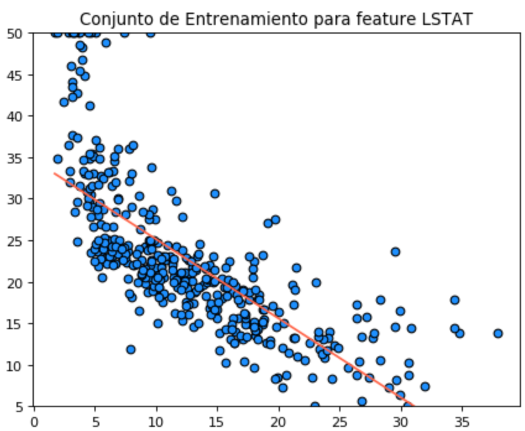


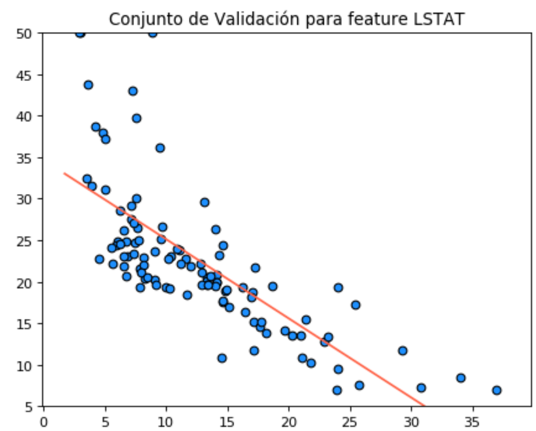

```
Media del error cuadrado para entrenamiento del feature ZN : 74.13
Media del error cuadrado para validación del feature ZN    : 71.00
```



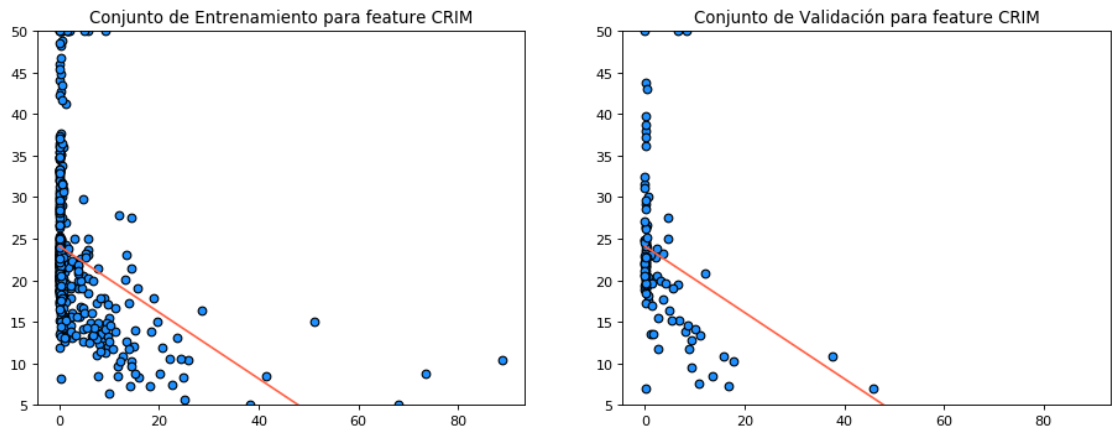



```
Media del error cuadrado para entrenamiento del feature PTRATIO : 62.06
Media del error cuadrado para validación del feature PTRATIO    : 65.02
```

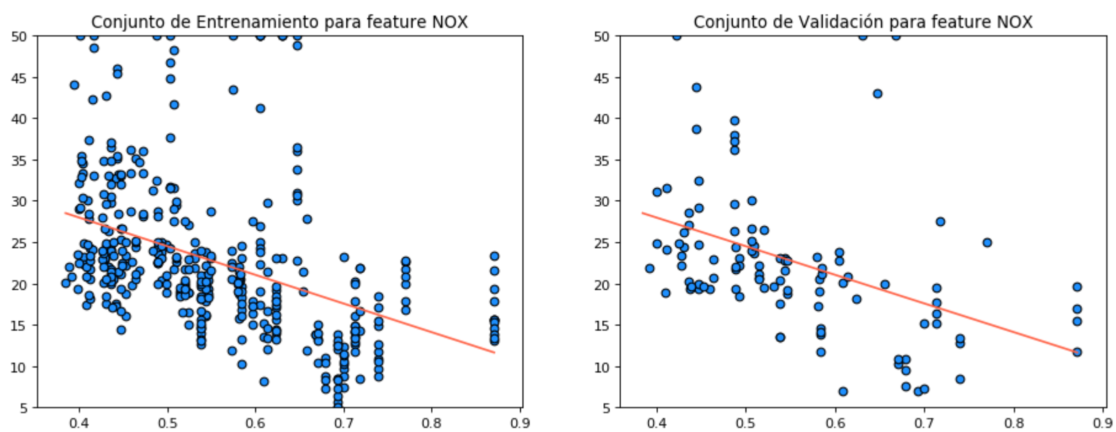





```
Media del error cuadrado para entrenamiento del feature LSTAT : 39.77
Media del error cuadrado para validación del feature LSTAT    : 33.66
```




```
Media del error cuadrado para entrenamiento del feature CRIM : 73.88
Media del error cuadrado para validación del feature CRIM    : 63.58
```




```
Media del error cuadrado para entrenamiento del feature NOX : 70.54
Media del error cuadrado para validación del feature NOX    : 63.28
```

Se observa que los features no se ajustan a una función lineal

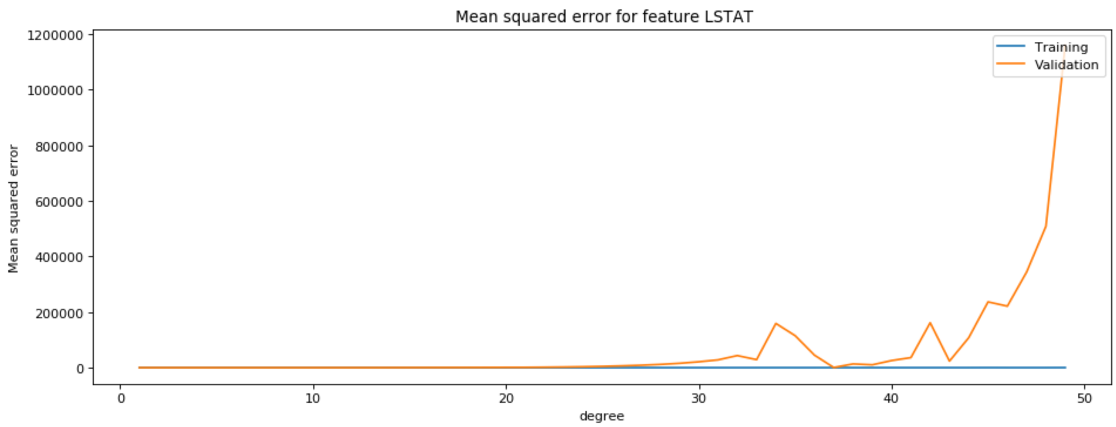
1.1.4 Regresión polinomial

https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.PolynomialFeatures.html
Analizamos el feature **LSTAT**

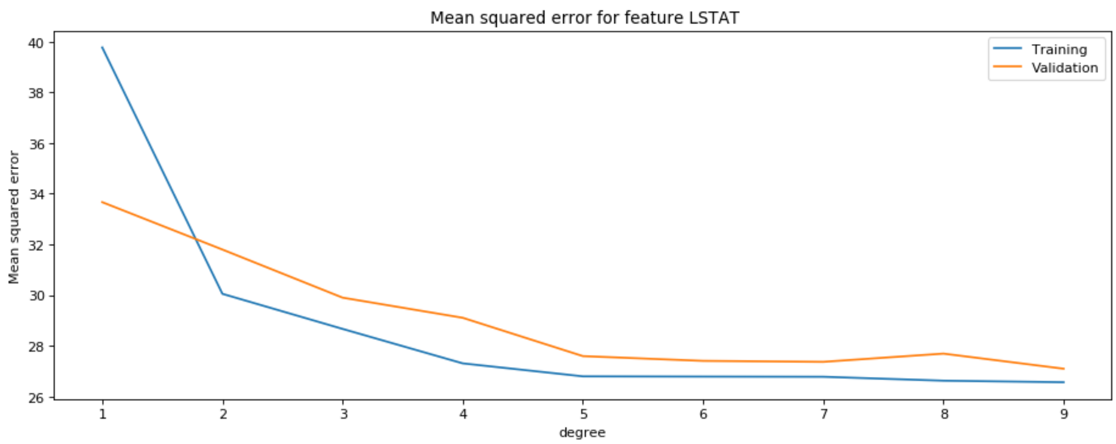
```
In [17]: selected_feature = 'LSTAT'
         feature_col = feature_map[selected_feature]
         X_train_feature = X_train[:, feature_col].reshape(-1, 1)
         X_val_feature = X_val[:, feature_col].reshape(-1, 1)
```

In [18]:
```
def get_mean_squared_error_for_lineal_regression(X_train_feature, y_train, X_val_featu
    """
    Get mean squared error for traning and validation set in a linear regression mode

    """
    poly_features = PolynomialFeatures(polynomial_degree)
    poly_features.fit(X_train_feature)
    X_poly_train = poly_features.transform(X_train_feature)
    X_poly_val = poly_features.transform(X_val_feature)

    model = LinearRegression()
    model.fit(X_poly_train, y_train)

    mean_squared_training_error = mean_squared_error(y_train, model.predict(X_poly_tra
    mean_squared_val_error = mean_squared_error(y_val, model.predict(X_poly_val))
    return mean_squared_training_error, mean_squared_val_error
```

In [19]:
```
# Evaluate until large polinomial degree in order to detect overfitting
x = range(1, 50)
errors_training = []
errors_val = []
for degree in x:

    train_error, val_error = get_mean_squared_error_for_lineal_regression(X_train_fea
                                                                          y_train,
                                                                          X_val_feature
                                                                          y_val,
                                                                          degree)
    errors_training.append(train_error)
    errors_val.append(val_error)


plt.figure(figsize=(14, 5), dpi= 80, facecolor='w', edgecolor='k')
plt.plot(x, errors_training, label='Training')
plt.plot(x, errors_val, label='Validation')
plt.title(f" Mean squared error for feature {selected_feature}")
plt.legend(loc='upper right')
plt.xlabel("degree")
plt.ylabel("Mean squared error")
plt.show()
```




```
In [20]: x = range(1, 10)
         errors_training = []
         errors_val = []
         for degree in x:
             train_error, val_error = get_mean_squared_error_for_lineal_regression(X_train_fea
                                                                                     y_train,
                                                                                     X_val_feature
                                                                                     y_val,
                                                                                     degree)
             errors_training.append(train_error)
             errors_val.append(val_error)

         plt.figure(figsize=(14, 5), dpi= 80, facecolor='w', edgecolor='k')
         plt.plot(x, errors_training, label='Training')
         plt.plot(x, errors_val, label='Validation')
         plt.title(f" Mean squared error for feature {selected_feature}")
         plt.legend(loc='upper right')
         plt.xlabel("degree")
         plt.ylabel("Mean squared error")
         plt.show()
```

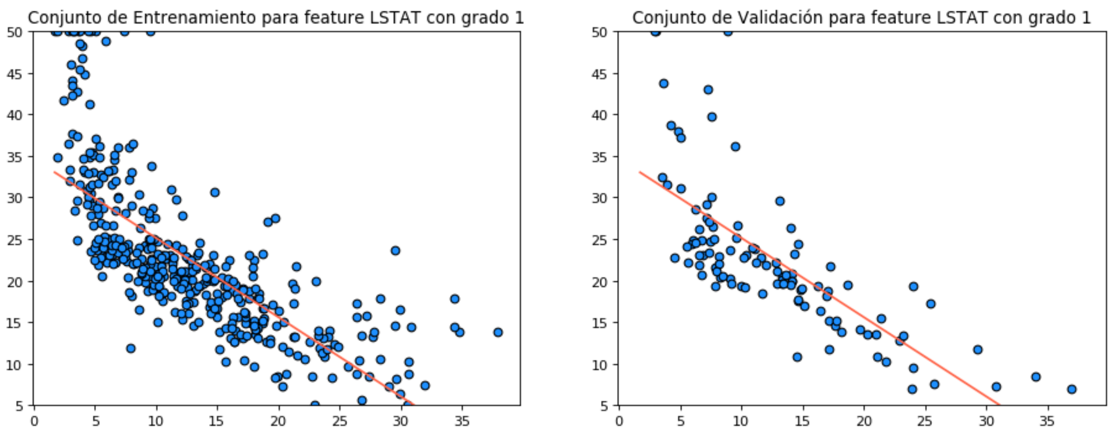


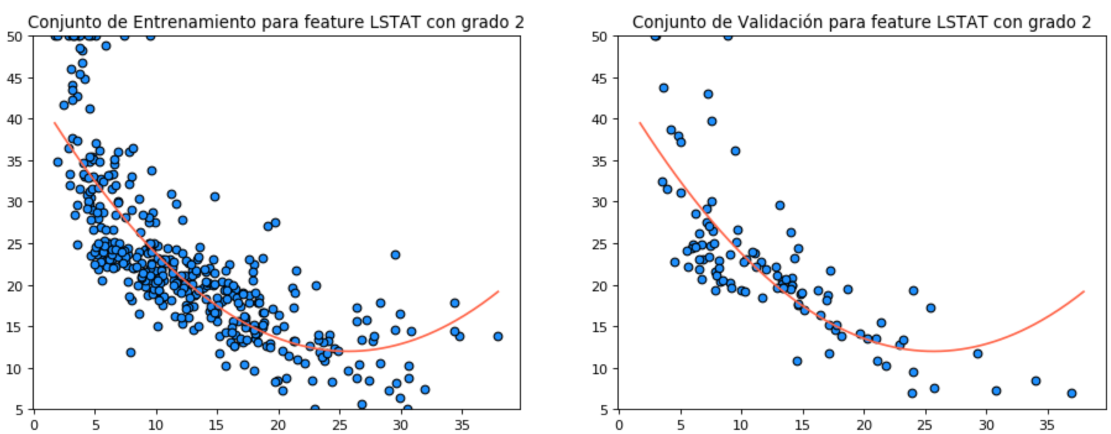
Se observa que el error medio para el conjunto de validación alcanza su minímo para un polinomio de grado 7 No es necesario ajustar al modelo con polinomios mayores a ese grado.

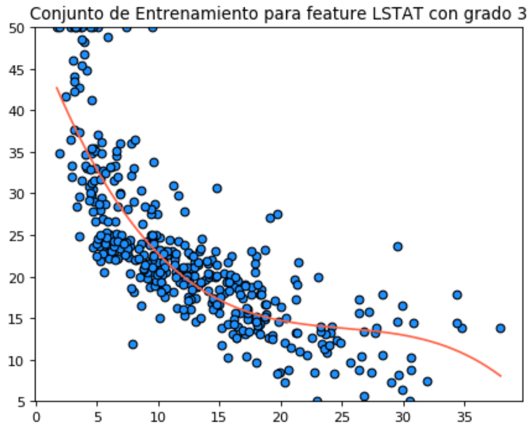
Visualización de la regresión polinomial

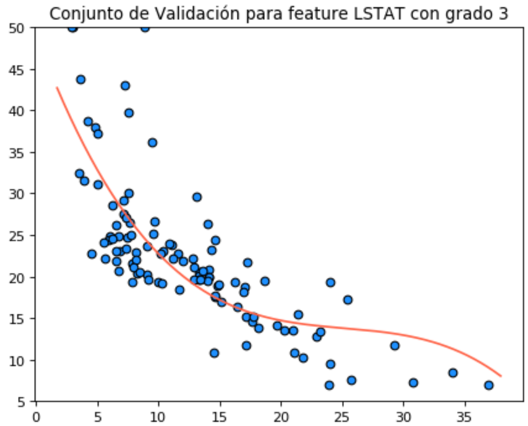
```
In [21]: for polynomial_degree in range(1, 9):

             # Probamos distintos grados del polinomio

             poly_features = PolynomialFeatures(polynomial_degree)
             poly_features.fit(X_train_feature)
             X_poly_train = poly_features.transform(X_train_feature)
             X_poly_val = poly_features.transform(X_val_feature)

             model = LinearRegression()
             model.fit(X_poly_train, y_train)

             plt.figure(figsize=(14, 5), dpi=80, facecolor='w', edgecolor='k')

             X_range_start = np.min(np.r_[X_train_feature, X_val_feature])
             X_range_stop = np.max(np.r_[X_train_feature, X_val_feature])
             y_range_start = np.min(np.r_[y_train, y_val])
             y_range_stop = np.max(np.r_[y_train, y_val])
             X_linspace = np.linspace(X_range_start, X_range_stop, 200).reshape(-1, 1)
             X_linspace_poly = poly_features.transform(X_linspace)

             # Conjunto de entrenamiento
             plt.subplot(1, 2, 1)
             plt.scatter(X_train_feature, y_train, facecolor="dodgerblue", edgecolor="k", label
             plt.plot(X_linspace, model.predict(X_linspace_poly), color="tomato", label="model
             plt.ylim(y_range_start, y_range_stop)
             plt.title(f"Conjunto de Entrenamiento para feature {selected_feature} con grado {
```

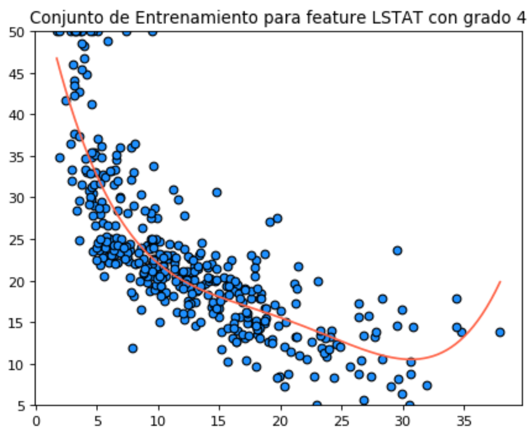
```python
# Conjunto de validación
plt.subplot(1, 2, 2)
plt.scatter(X_val_feature, y_val, facecolor="dodgerblue", edgecolor="k", label="da
plt.plot(X_linspace, model.predict(X_linspace_poly), color="tomato", label="model
plt.ylim(y_range_start, y_range_stop)
plt.title(f"Conjunto de Validación para feature {selected_feature} con grado {poly

plt.show()
```

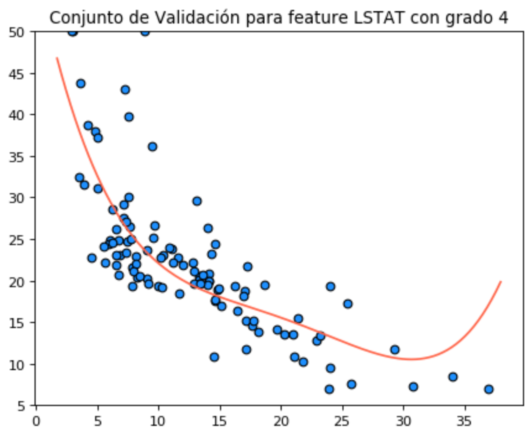


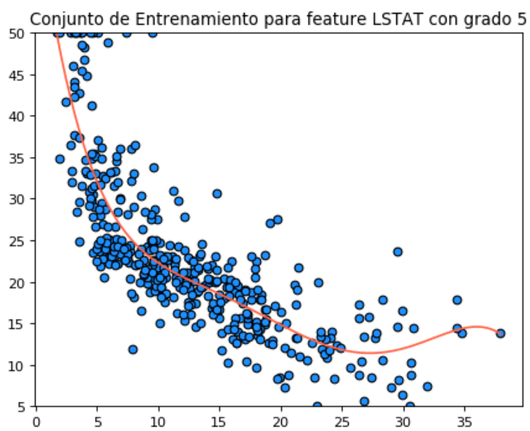


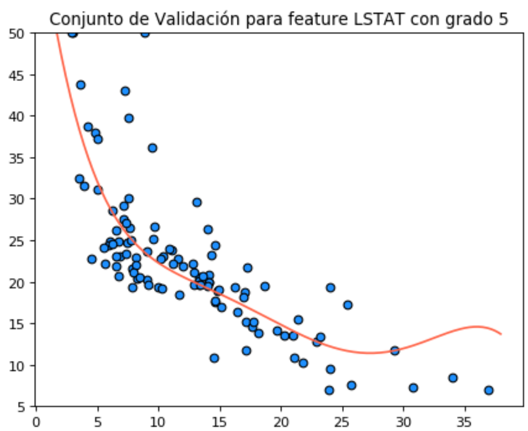

Conjunto de Entrenamiento para feature LSTAT con grado 3

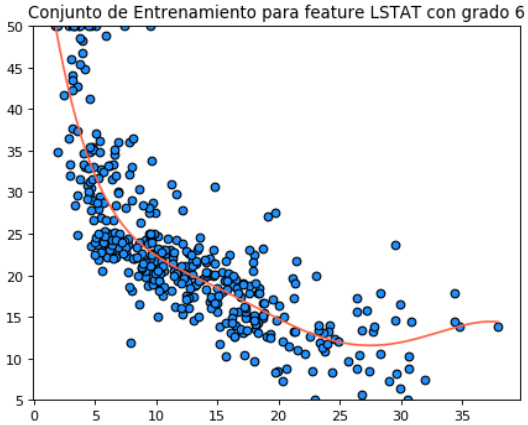

Conjunto de Validación para feature LSTAT con grado 3

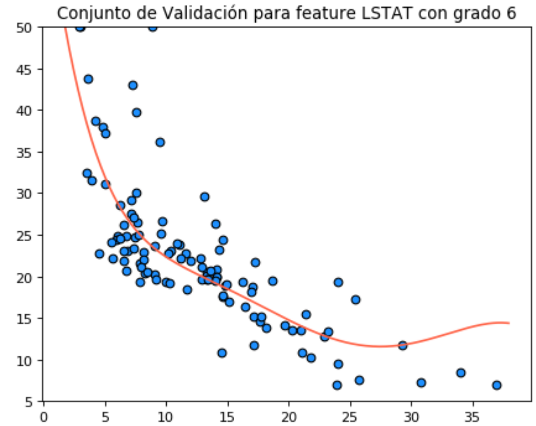

Conjunto de Entrenamiento para feature LSTAT con grado 4

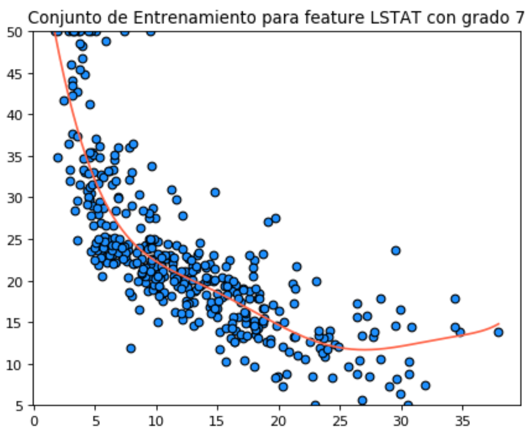

Conjunto de Validación para feature LSTAT con grado 4

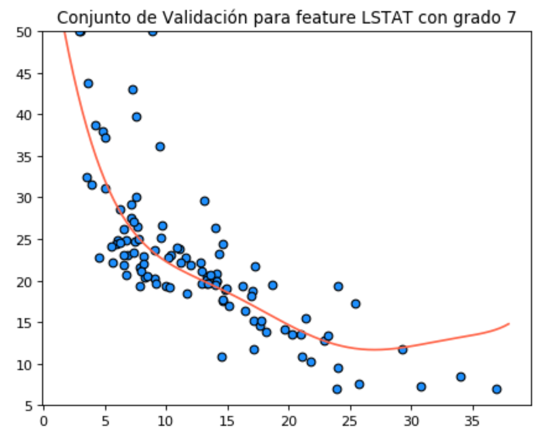

Conjunto de Entrenamiento para feature LSTAT con grado 5

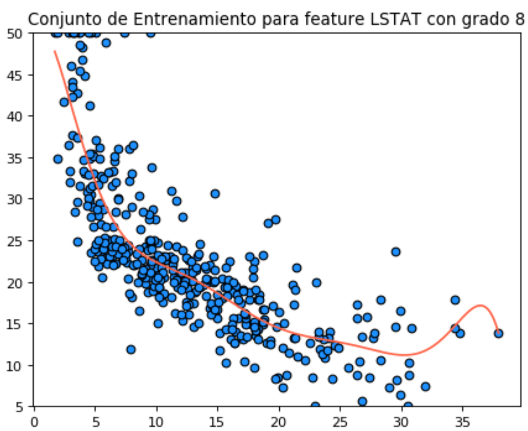

Conjunto de Validación para feature LSTAT con grado 5

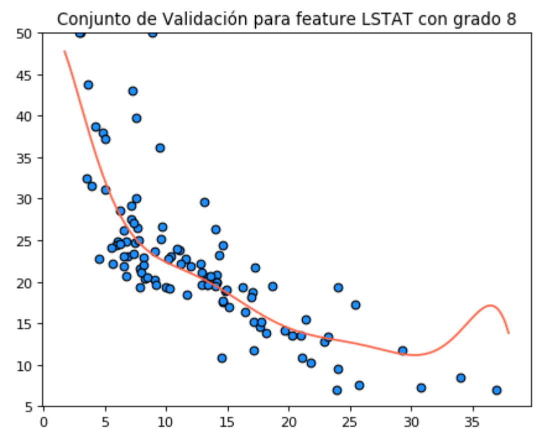

Conjunto de Entrenamiento para feature LSTAT con grado 6
50 45 40 35 30 25 20 15 10 5
0 5 10 15 20 25 30 35


Conjunto de Validación para feature LSTAT con grado 6
50 45 40 35 30 25 20 15 10 5
0 5 10 15 20 25 30 35


Conjunto de Entrenamiento para feature LSTAT con grado 7
50 45 40 35 30 25 20 15 10 5
0 5 10 15 20 25 30 35


Conjunto de Validación para feature LSTAT con grado 7
50 45 40 35 30 25 20 15 10 5
0 5 10 15 20 25 30 35


Conjunto de Entrenamiento para feature LSTAT con grado 8
50 45 40 35 30 25 20 15 10 5
0 5 10 15 20 25 30 35


Conjunto de Validación para feature LSTAT con grado 8
50 45 40 35 30 25 20 15 10 5
0 5 10 15 20 25 30 35

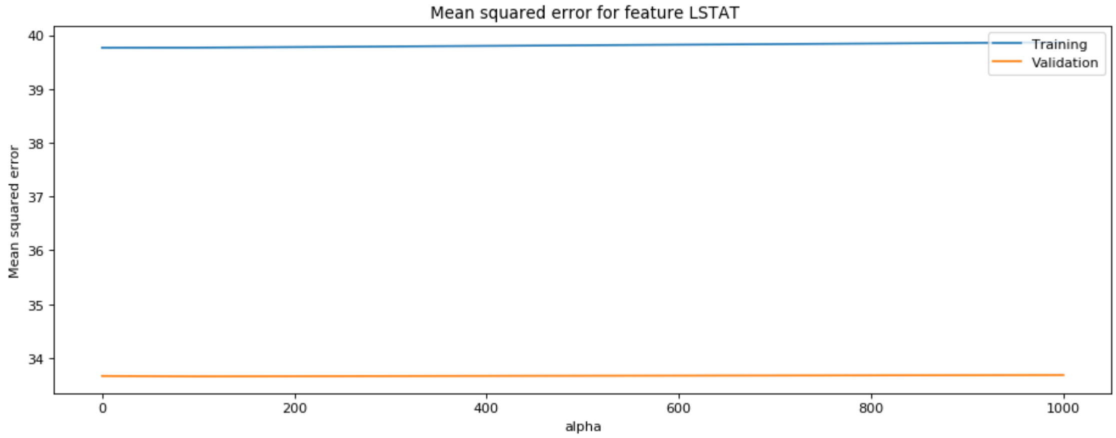
1.1.5 Regresión lineal con regularización

- https://scikit-learn.org/stable/modules/linear_model.html#ridge-regression
- https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html

In [22]:
```
def get_mean_squared_error_for_lineal_regression_gridge(X_train_feature, y_train, X_va
    """
    Get mean squared error for traning and validation set in a linear regression mode

    """
    model = Ridge(alpha=alpha)
    model.fit(X_train_feature, y_train)
    mean_squared_training_error = mean_squared_error(y_train, model.predict(X_train_fe
    mean_squared_val_error = mean_squared_error(y_val, model.predict(X_val_feature))
    return mean_squared_training_error, mean_squared_val_error
```

In [23]:
```
x = [pow(10, x) for x in list(range(-5, 5))]
errors_training = []
errors_val = []
for alpha in x:
    train_error, val_error = get_mean_squared_error_for_lineal_regression_gridge(X_tra
                                                                        y_train,
                                                                        X_val_feature
                                                                        y_val,
                                                                        alpha)
    errors_training.append(train_error)
    errors_val.append(val_error)


plt.figure(figsize=(14, 5), dpi= 80, facecolor='w', edgecolor='k')
plt.plot(x, errors_training, label='Training')
plt.plot(x, errors_val, label='Validation')
plt.title(f" Mean squared error for feature {selected_feature}")
plt.legend(loc='upper right')
plt.xlabel("alpha")
plt.ylabel("Mean squared error")
plt.show()
```

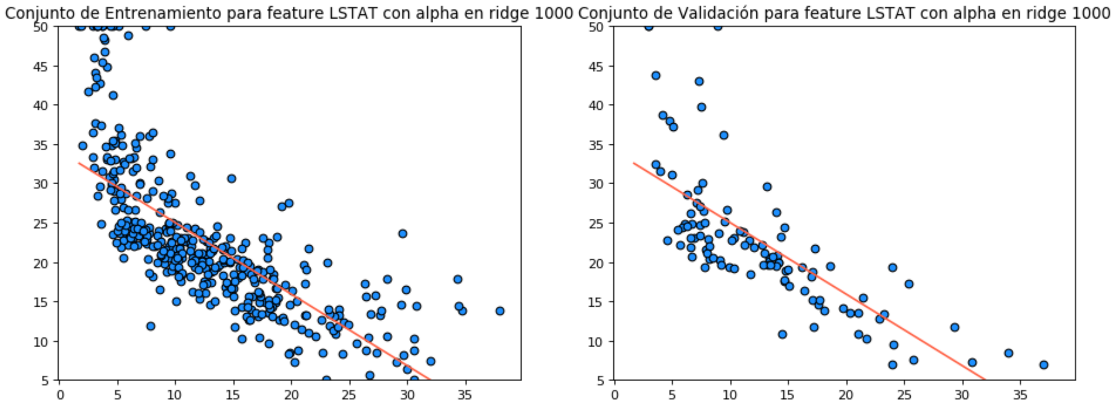

```
In [24]: x = [pow(10, x) for x in list(range(-1, 4))]
         errors_training = []
         errors_val = []
         for alpha in x:
             errors_training.append(get_mean_squared_error_for_lineal_regression_gridge(X_trai
                                                                                         y_train,
                                                                                         X_val_feature
                                                                                         y_val,
                                                                                         alpha)[0])
             errors_val.append(get_mean_squared_error_for_lineal_regression_gridge(X_train_fea
                                                                                    y_train,
                                                                                    X_val_feature,
                                                                                    y_val,
                                                                                    alpha)[1])

         plt.figure(figsize=(14, 5), dpi= 80, facecolor='w', edgecolor='k')
         plt.plot(x, errors_training, label='Training')
         plt.plot(x, errors_val, label='Validation')
         plt.title(f" Mean squared error for feature {selected_feature}")
         plt.legend(loc='upper right')
         plt.xlabel("alpha")
         plt.ylabel("Mean squared error")
         plt.show()
```



Visualización de la regresión lineal

```
In [25]: def plot_lineal_regression_ridge(X_train_feature, y_train, X_val_feature, y_val, selec

             plt.figure(figsize=(14, 5), dpi= 80, facecolor='w', edgecolor='k')

             X_range_start = np.min(np.r_[X_train_feature, X_val_feature])
             X_range_stop = np.max(np.r_[X_train_feature, X_val_feature])
             y_range_start = np.min(np.r_[y_train, y_val])
             y_range_stop = np.max(np.r_[y_train, y_val])
             X_linspace = np.linspace(X_range_start, X_range_stop, 200).reshape(-1, 1)

             # Conjunto de entrenamiento
             plt.subplot(1, 2, 1)
             plt.scatter(X_train_feature, y_train, facecolor="dodgerblue", edgecolor="k", label
             plt.plot(X_linspace, model.predict(X_linspace), color="tomato", label="modelo")
             plt.ylim(y_range_start, y_range_stop)
             plt.title(f"Conjunto de Entrenamiento para feature {selected_feature} con alpha en


             # Conjunto de validación
             plt.subplot(1, 2, 2)
             plt.scatter(X_val_feature, y_val, facecolor="dodgerblue", edgecolor="k", label="da
             plt.plot(X_linspace, model.predict(X_linspace), color="tomato", label="modelo")
             plt.ylim(y_range_start, y_range_stop)
             plt.title(f"Conjunto de Validación para feature {selected_feature} con alpha en ri
             plt.show()

In [26]: alpha =  1000 # Parámetro de regularización. También denominado como parámetro `lambda
         model = Ridge(alpha=alpha)
         model.fit(X_train_feature, y_train)
```

```
plot_lineal_regression_ridge(X_train_feature, y_train, X_val_feature, y_val, selected_
```



No tiene mucho sentido aplicar el ridge a un polinomio de grado 1(pues no estamos suavizando nada) De echo con una recta tenemos un problema de underfitting y el ridge sirve para suavizar polinomios con grados altos, que ajustan muy bien en etapa de entrenamiento pero no en etapa de validación(overfitting)

1.1.6 Regresión polinomial con regularización

Probamos con grado polinomio de grado 7 que fue el mejor resultado para regresion polinomial sin regularizacion

Probamos los siguientes valores para lambda(o alpha)

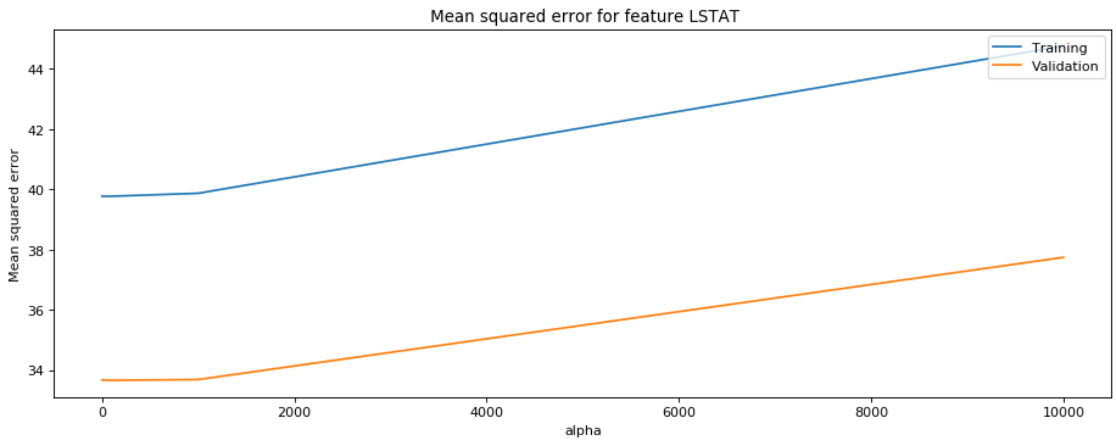
```
In [27]: x = [pow(10, x) for x in list(range(-4, 5))]
         x
```

```
Out[27]: [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000]
```

```
In [28]: polynomial_degree = 7
         alpha = 100

         poly_features = PolynomialFeatures(polynomial_degree)
         poly_features.fit(X_train_feature)
         X_poly_train = poly_features.transform(X_train_feature)
         X_poly_val = poly_features.transform(X_val_feature)

         model = Ridge(alpha=alpha)
         model.fit(X_poly_train, y_train)

         errors_training = []
         errors_val = []
         for alpha in x:
             errors_training.append(get_mean_squared_error_for_lineal_regression_gridge(X_trai
```

```
                                                                                          y_train,
                                                                                          X_val_feature
                                                                                          y_val,
                                                                                          alpha)[0])
            errors_val.append(get_mean_squared_error_for_lineal_regression_gridge(X_train_fea
                                                                                     y_train,
                                                                                     X_val_feature,
                                                                                     y_val,
                                                                                     alpha)[1])

        plt.figure(figsize=(14, 5), dpi= 80, facecolor='w', edgecolor='k')
        plt.plot(x, errors_training, label='Training')
        plt.plot(x, errors_val, label='Validation')
        plt.title(f" Mean squared error for feature {selected_feature}")
        plt.legend(loc='upper right')
        plt.xlabel("alpha")
        plt.ylabel("Mean squared error")
        plt.show()
```

```
/Users/luisvargas/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/ridge.py:125: Lin
  overwrite_a=True).T
```



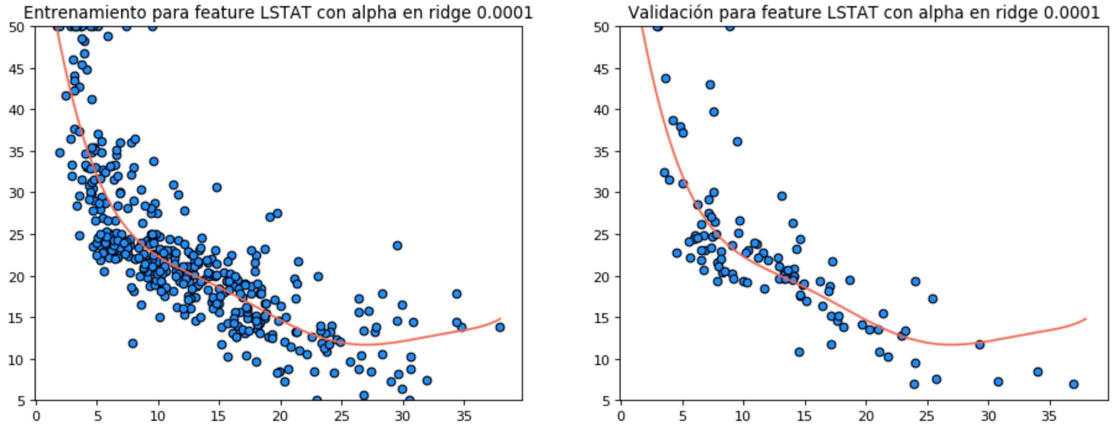
Se observa el mismo comportamiento del error con ridge para regression polinomial. No parece que el termino de regularización ayude mucho al ajuste de este feature.

Visualización de la regresión polinomial

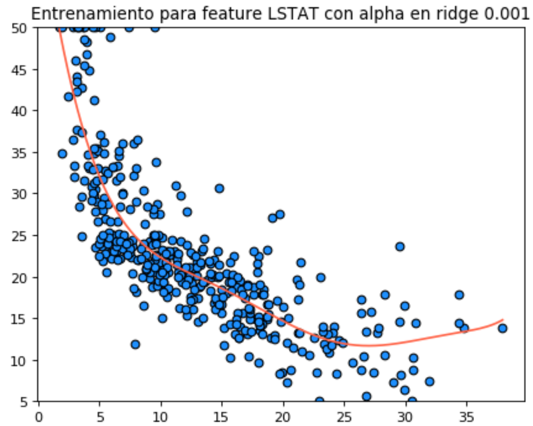
```
In [29]: alphas = [pow(10, x) for x in list(range(-4, 4))]
         for alpha in alphas:
             plt.figure(figsize=(14, 5), dpi= 80, facecolor='w', edgecolor='k')

             polynomial_degree = 7
```

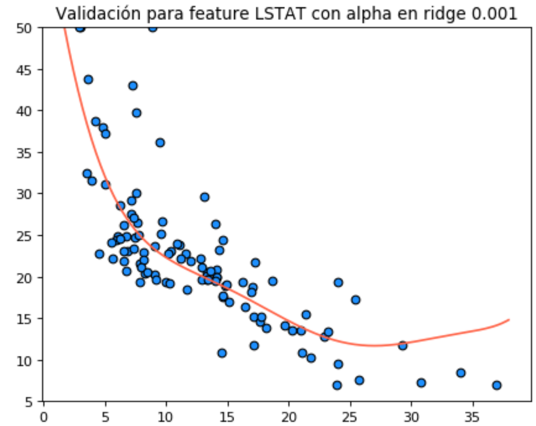
```
    poly_features = PolynomialFeatures(polynomial_degree)
    poly_features.fit(X_train_feature)
    X_poly_train = poly_features.transform(X_train_feature)
    X_poly_val = poly_features.transform(X_val_feature)

    model = Ridge(alpha=alpha)
    model.fit(X_poly_train, y_train)

    X_range_start = np.min(np.r_[X_train_feature, X_val_feature])
    X_range_stop = np.max(np.r_[X_train_feature, X_val_feature])
    y_range_start = np.min(np.r_[y_train, y_val])
    y_range_stop = np.max(np.r_[y_train, y_val])
    X_linspace = np.linspace(X_range_start, X_range_stop, 200).reshape(-1, 1)
    X_linspace_poly = poly_features.transform(X_linspace)

    # Conjunto de entrenamiento
    plt.subplot(1, 2, 1)
    plt.scatter(X_train_feature, y_train, facecolor="dodgerblue", edgecolor="k", label
    plt.plot(X_linspace, model.predict(X_linspace_poly), color="tomato", label="model
    plt.ylim(y_range_start, y_range_stop)
    plt.title(f"Entrenamiento para feature {selected_feature} con alpha en ridge {alph

    # Conjunto de validación
    plt.subplot(1, 2, 2)
    plt.scatter(X_val_feature, y_val, facecolor="dodgerblue", edgecolor="k", label="da
    plt.plot(X_linspace, model.predict(X_linspace_poly), color="tomato", label="model
    plt.ylim(y_range_start, y_range_stop)
    plt.title(f"Validación para feature {selected_feature} con alpha en ridge {alpha}"

    plt.show()
```

```
/Users/luisvargas/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/ridge.py:125: Lin
  overwrite_a=True).T
```

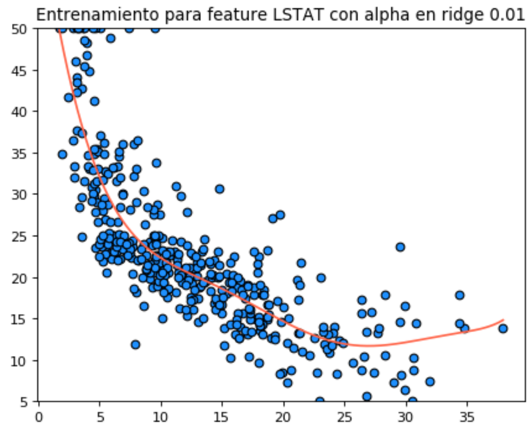


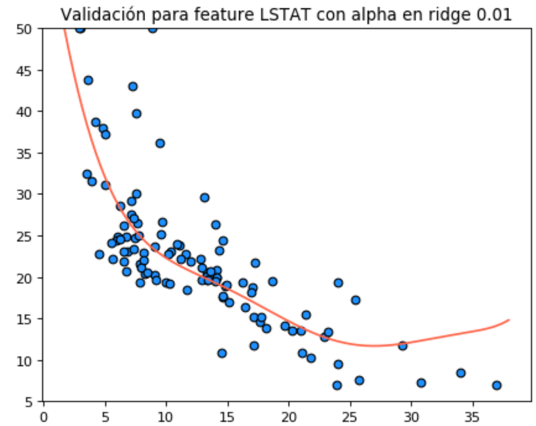
```
/Users/luisvargas/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/ridge.py:125: Lin
  overwrite_a=True).T
```



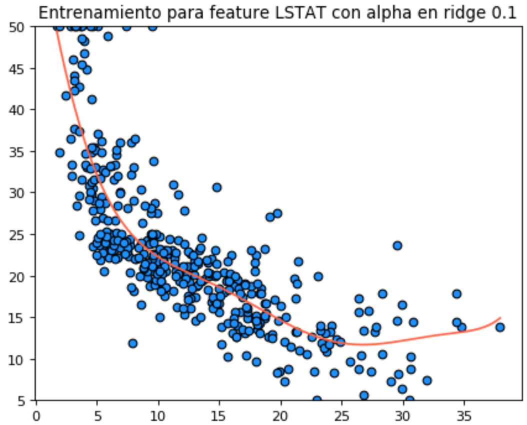


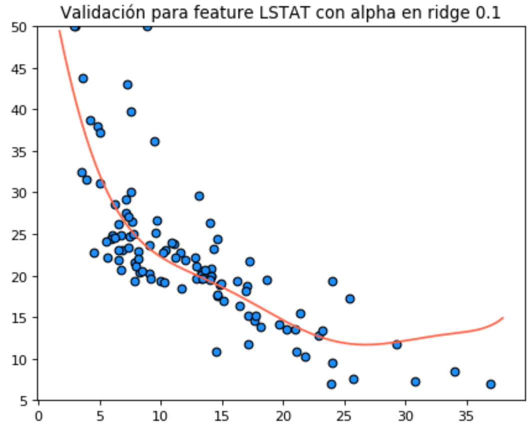

```
/Users/luisvargas/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/ridge.py:125: Lin
  overwrite_a=True).T
```



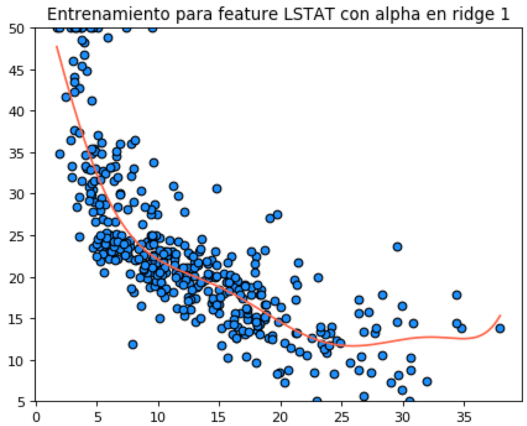


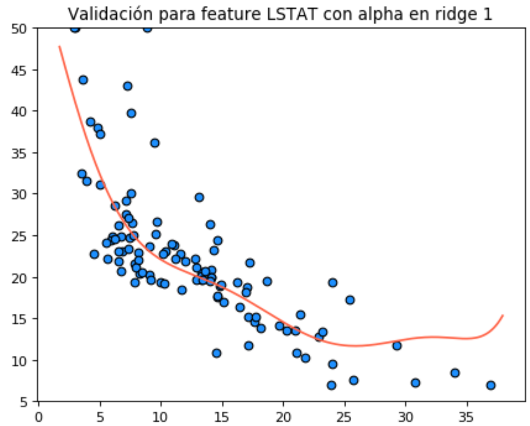

```
/Users/luisvargas/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/ridge.py:125: Lin
  overwrite_a=True).T
```



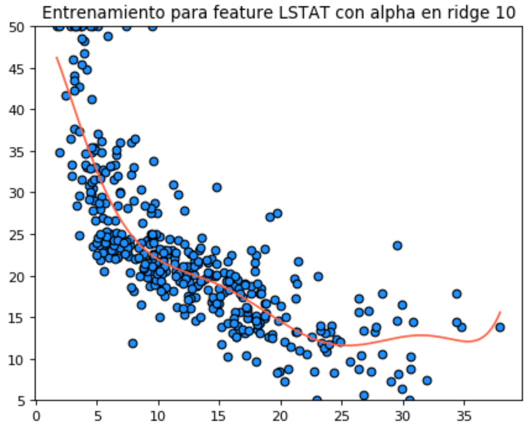


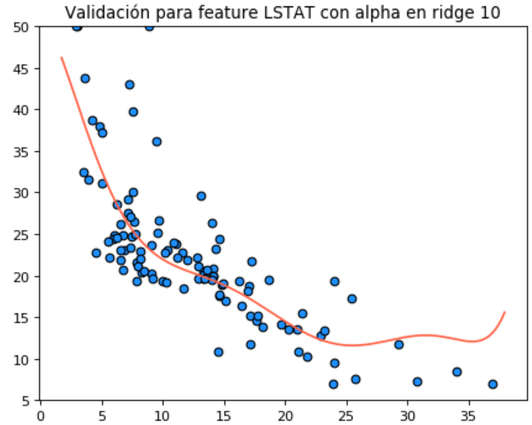

```
/Users/luisvargas/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/ridge.py:125: Lin
  overwrite_a=True).T
```



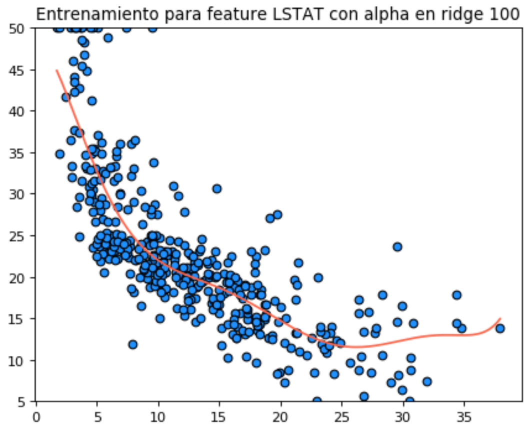


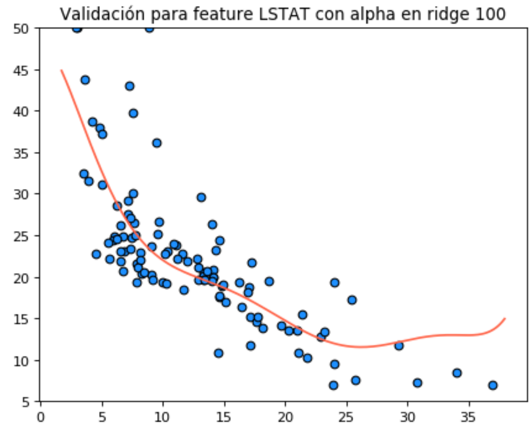

```
/Users/luisvargas/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/ridge.py:125: Lin
  overwrite_a=True).T
```



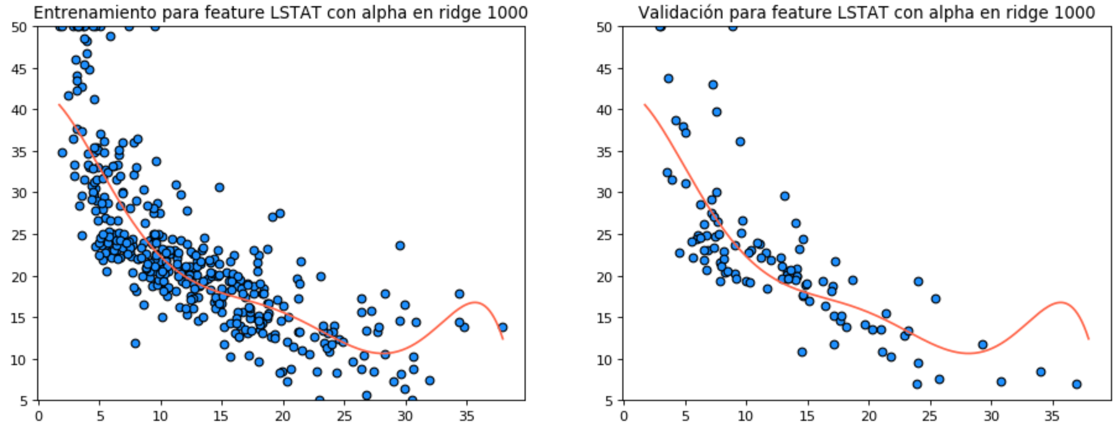



```
/Users/luisvargas/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/ridge.py:125: Lin
  overwrite_a=True).T
```






```
/Users/luisvargas/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/ridge.py:125: Lin
  overwrite_a=True).T
```



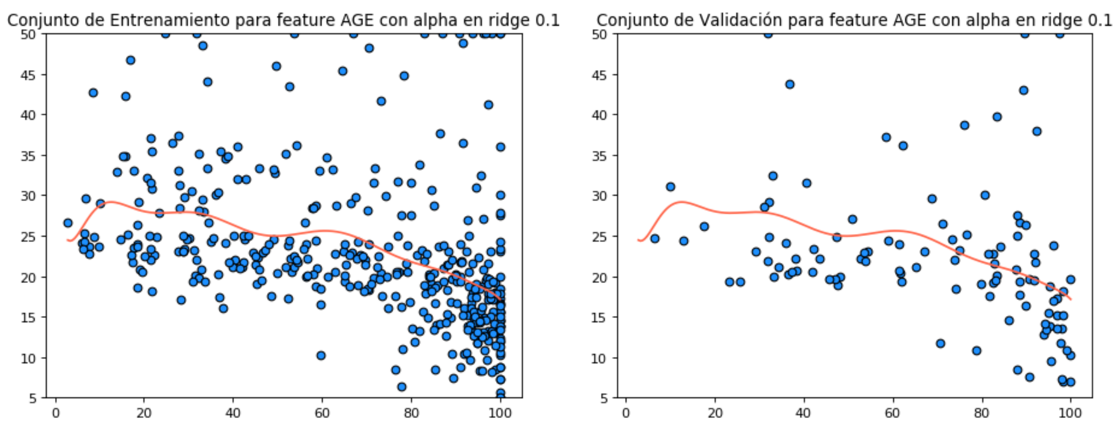
Se observa que para un polinomio de grado 7 un α bajo, parece ajustar un poco mejor que para valores altos

Analizamos otros features:

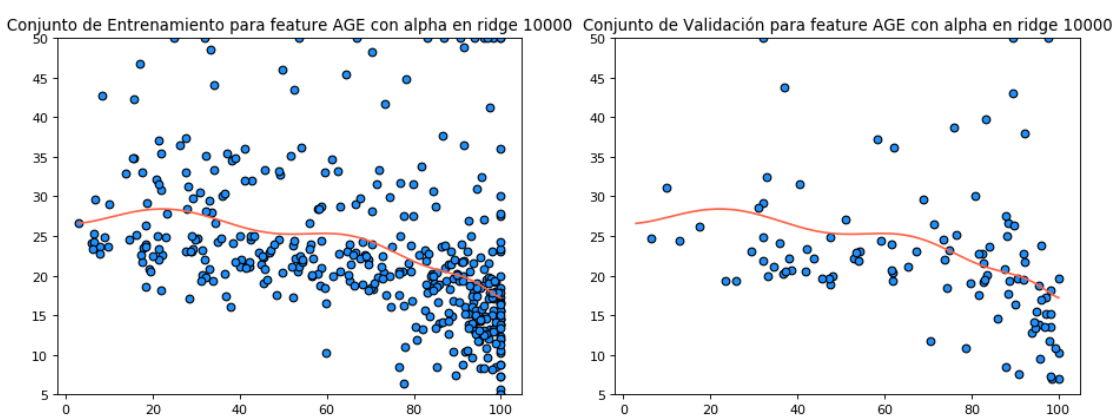
```
In [30]: def plot_polinomial_regression_ridge(X_train_feature, y_train, X_val_feature, y_val, s
             plt.figure(figsize=(14, 5), dpi= 80, facecolor='w', edgecolor='k')

             X_range_start = np.min(np.r_[X_train_feature, X_val_feature])
             X_range_stop = np.max(np.r_[X_train_feature, X_val_feature])
             y_range_start = np.min(np.r_[y_train, y_val])
             y_range_stop = np.max(np.r_[y_train, y_val])
             X_linspace = np.linspace(X_range_start, X_range_stop, 200).reshape(-1, 1)
             X_linspace_poly = poly_features.transform(X_linspace)

             # Conjunto de entrenamiento
             plt.subplot(1, 2, 1)
             plt.scatter(X_train_feature, y_train, facecolor="dodgerblue", edgecolor="k", label
             plt.plot(X_linspace, model.predict(X_linspace_poly), color="tomato", label="modelo
             plt.ylim(y_range_start, y_range_stop)
             plt.title(f"Conjunto de Entrenamiento para feature {selected_feature} con alpha en

             # Conjunto de validación
             plt.subplot(1, 2, 2)
             plt.scatter(X_val_feature, y_val, facecolor="dodgerblue", edgecolor="k", label="da
             plt.plot(X_linspace, model.predict(X_linspace_poly), color="tomato", label="modelo
             plt.ylim(y_range_start, y_range_stop)
             plt.title(f"Conjunto de Validación para feature {selected_feature} con alpha en ri

             plt.show()

In [31]: for selected_feature in numeric_features:
             feature_col = feature_map[selected_feature]
             X_train_feature = X_train[:, feature_col].reshape(-1, 1)  # Hay que ser que sea u
```

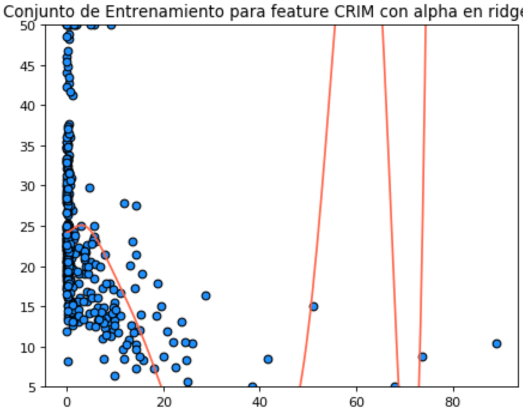
```
    X_val_feature = X_val[:, feature_col].reshape(-1, 1)
    for alpha in (0.1, 10000):
        polynomial_degree = 11
        poly_features = PolynomialFeatures(polynomial_degree)
        poly_features.fit(X_train_feature)
        X_poly_train = poly_features.transform(X_train_feature)
        X_poly_val = poly_features.transform(X_val_feature)

        model = Ridge(alpha=alpha)
        model.fit(X_poly_train, y_train)

        plot_polinomial_regression_ridge(X_train_feature, y_train, X_val_feature, y_va
```

```
/Users/luisvargas/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/ridge.py:125: Lin
  overwrite_a=True).T
```

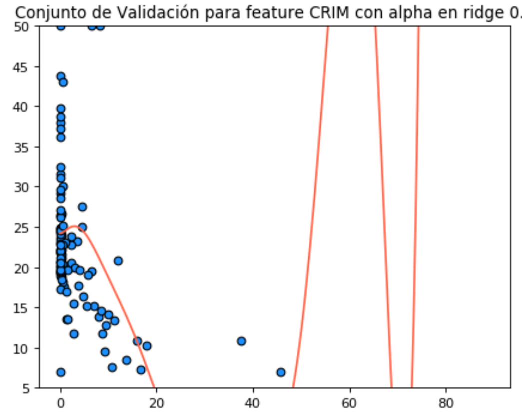



```
/Users/luisvargas/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/ridge.py:125: Lin
  overwrite_a=True).T
```

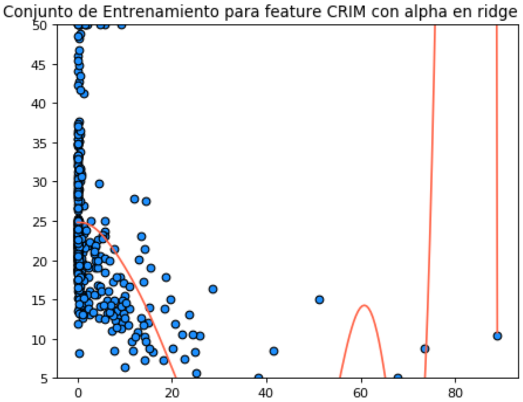




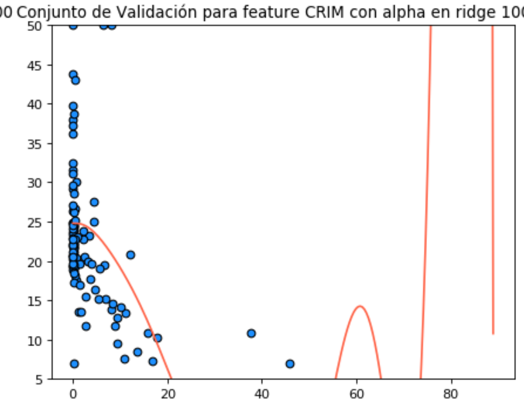
Conjunto de Entrenamiento para feature CRIM con alpha en ridge 0.1



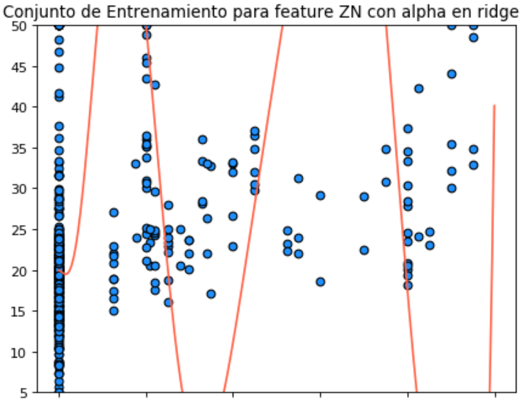
Conjunto de Validación para feature CRIM con alpha en ridge 0.1



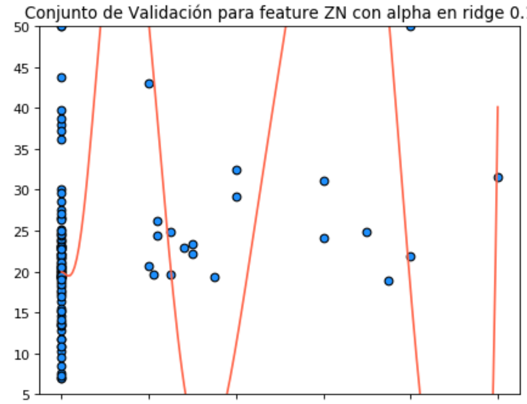
Conjunto de Entrenamiento para feature CRIM con alpha en ridge 10000



Conjunto de Validación para feature CRIM con alpha en ridge 10000

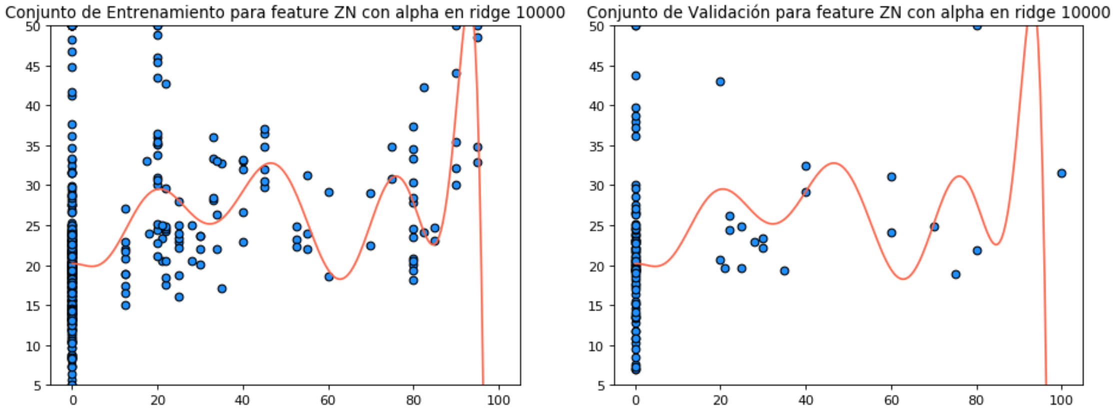


Conjunto de Entrenamiento para feature ZN con alpha en ridge 0.1

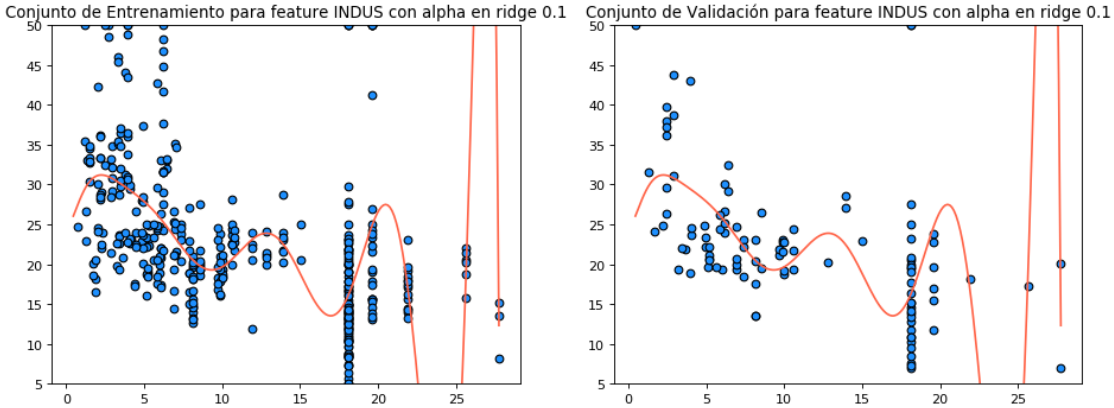


Conjunto de Validación para feature ZN con alpha en ridge 0.1

```
/Users/luisvargas/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/ridge.py:125: Lin
  overwrite_a=True).T
```

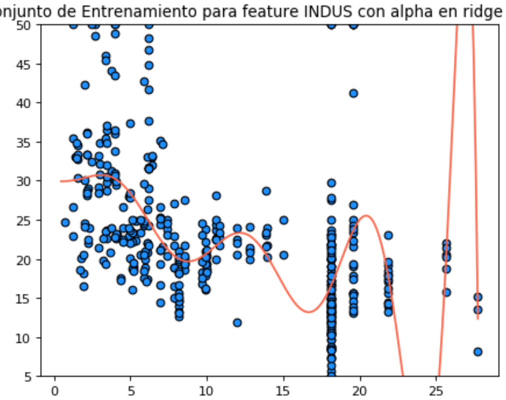



```
/Users/luisvargas/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/ridge.py:125: Lin
  overwrite_a=True).T
```

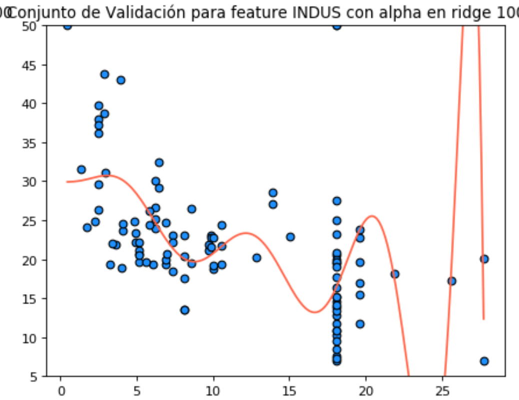



```
/Users/luisvargas/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/ridge.py:125: Lin
  overwrite_a=True).T
```

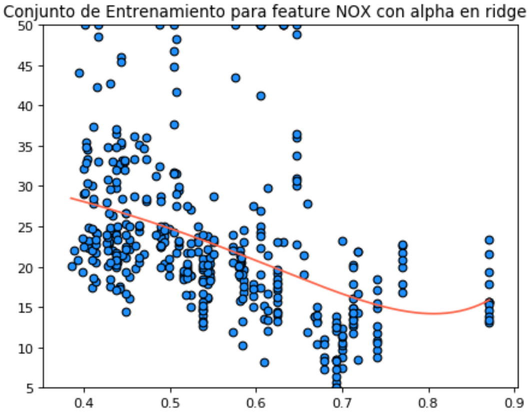
Conjunto de Entrenamiento para feature INDUS con alpha en ridge 10000



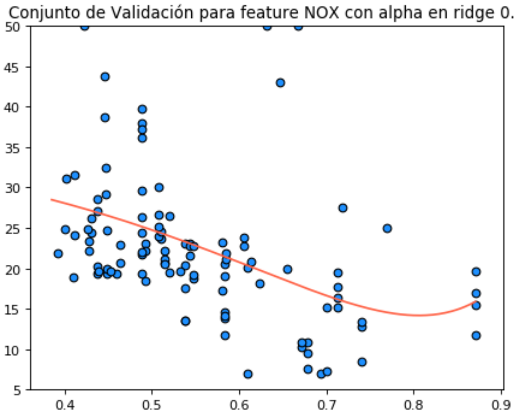
Conjunto de Validación para feature INDUS con alpha en ridge 10000



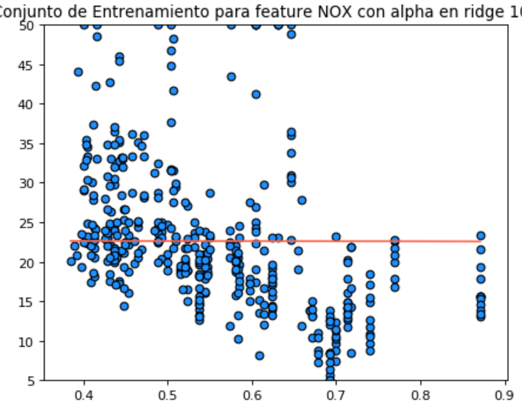
Conjunto de Entrenamiento para feature NOX con alpha en ridge 0.1



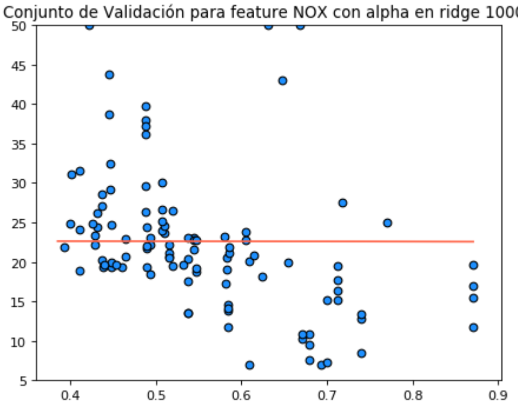
Conjunto de Validación para feature NOX con alpha en ridge 0.1



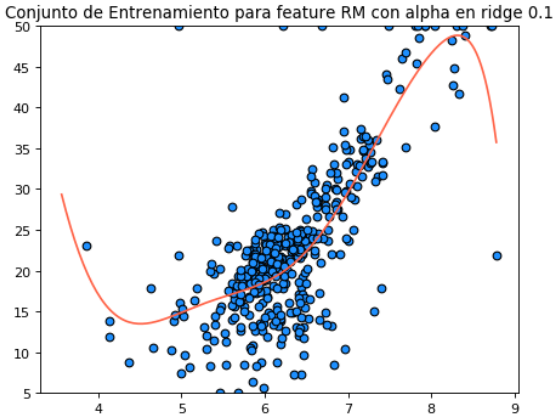
Conjunto de Entrenamiento para feature NOX con alpha en ridge 10000

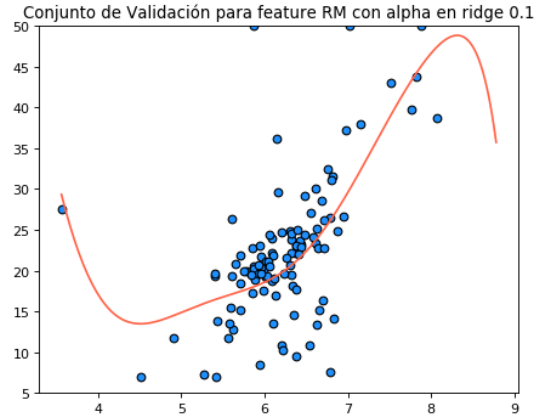


Conjunto de Validación para feature NOX con alpha en ridge 10000

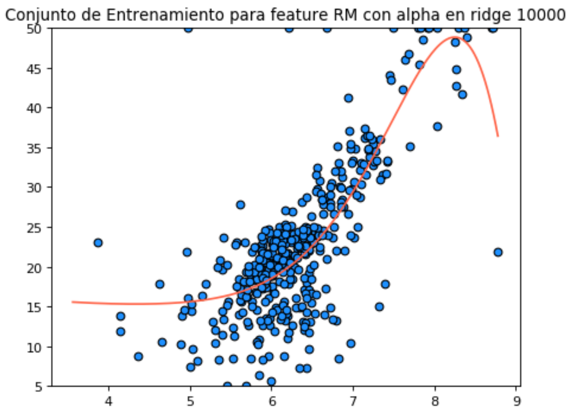


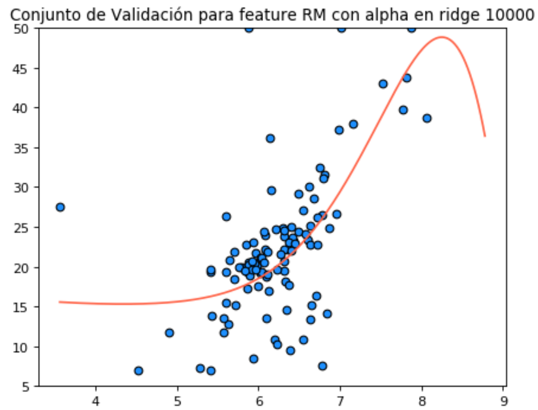
```
/Users/luisvargas/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/ridge.py:125: Lin
  overwrite_a=True).T
```



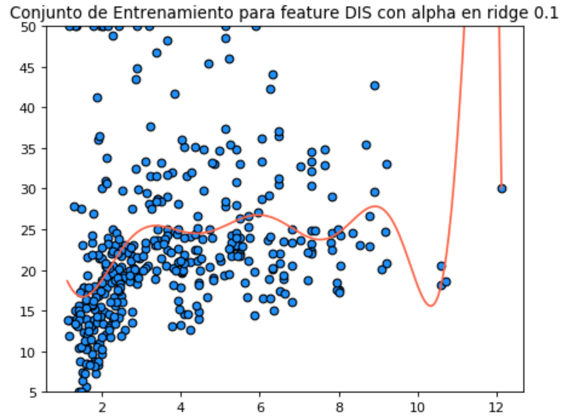


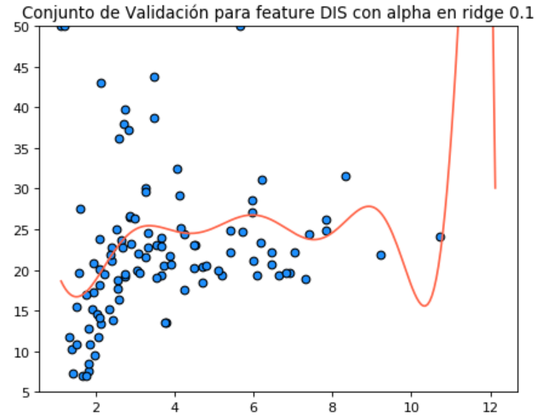

```
/Users/luisvargas/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/ridge.py:125: Lin
  overwrite_a=True).T
```



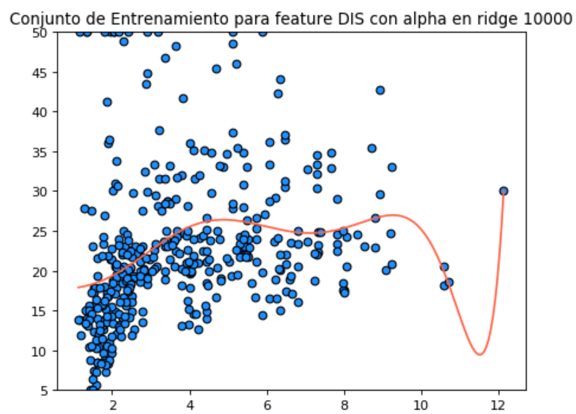


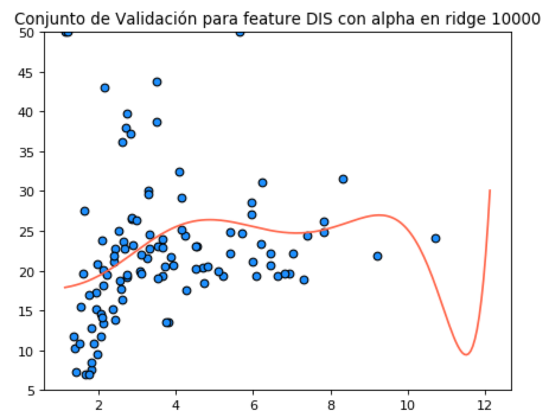

```
/Users/luisvargas/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/ridge.py:125: Lin
  overwrite_a=True).T
```

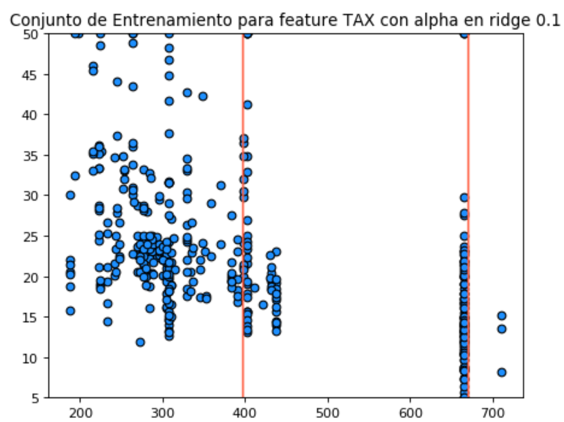


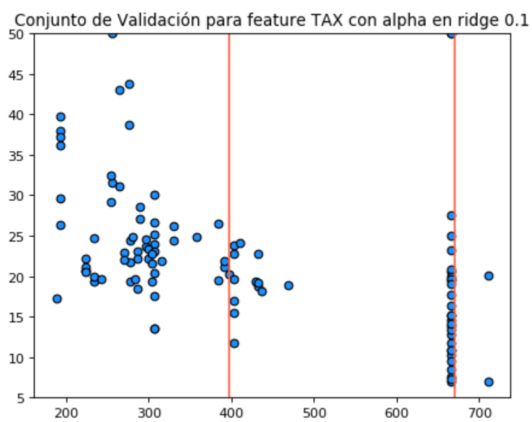


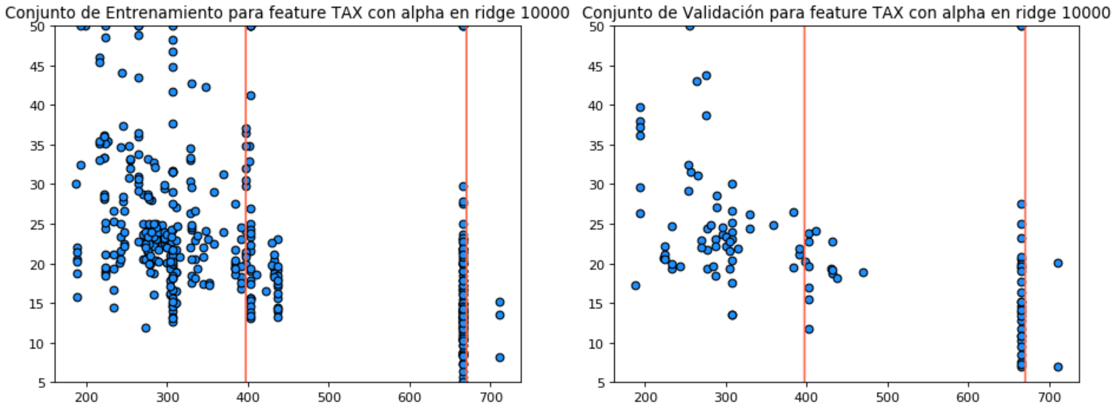

```
/Users/luisvargas/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/ridge.py:125: Lin
  overwrite_a=True).T
```

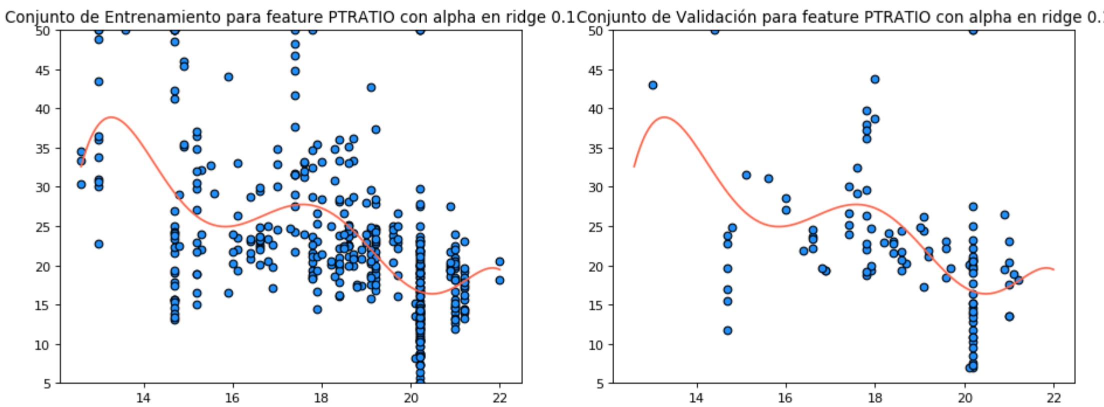




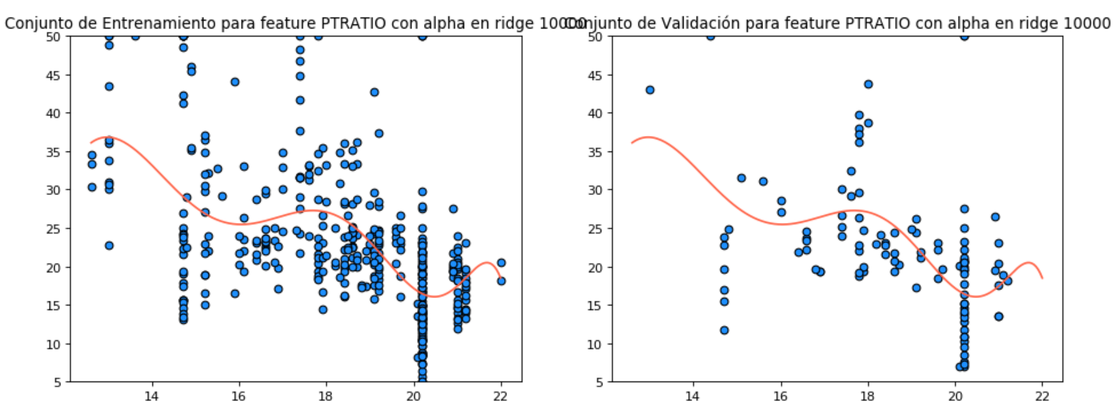




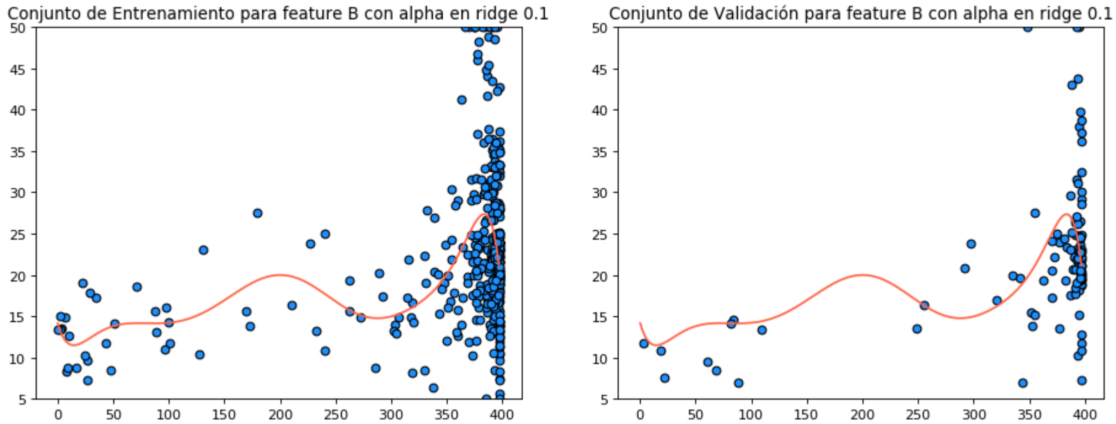




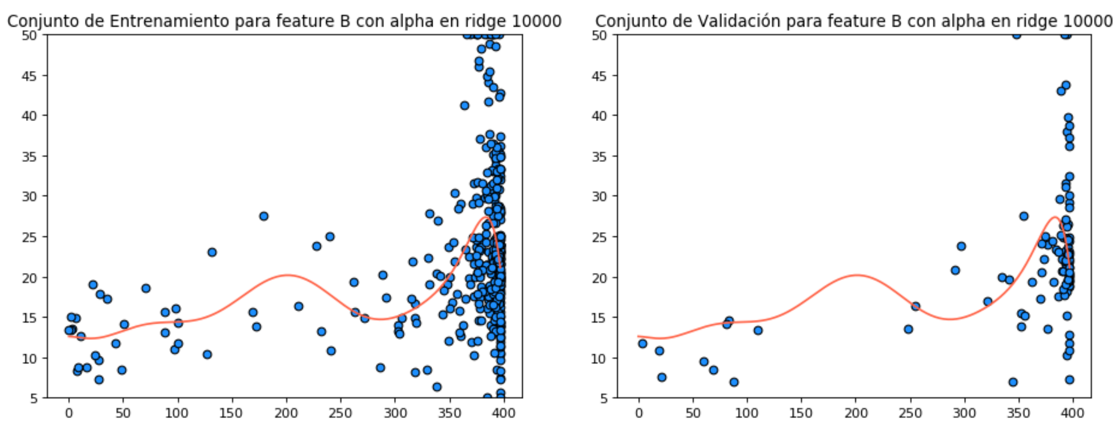

```
/Users/luisvargas/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/ridge.py:125: Lin
  overwrite_a=True).T
```



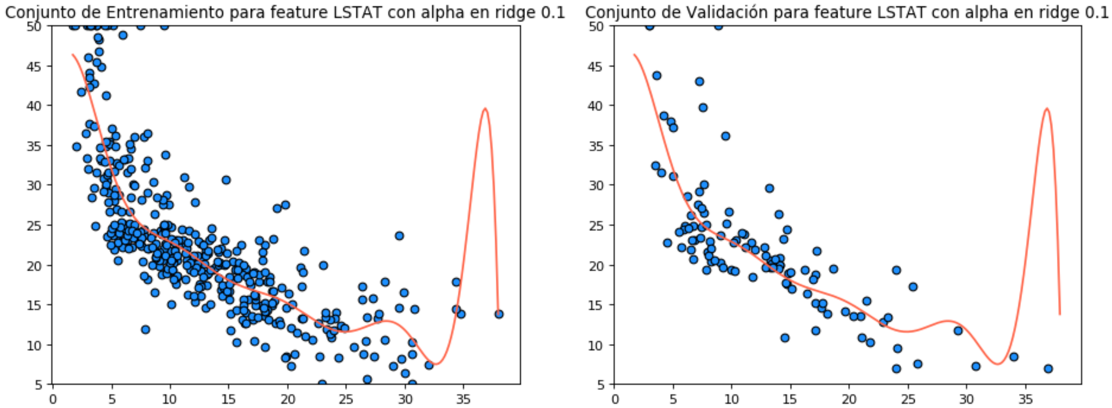
```
/Users/luisvargas/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/ridge.py:125: Lin
  overwrite_a=True).T
```



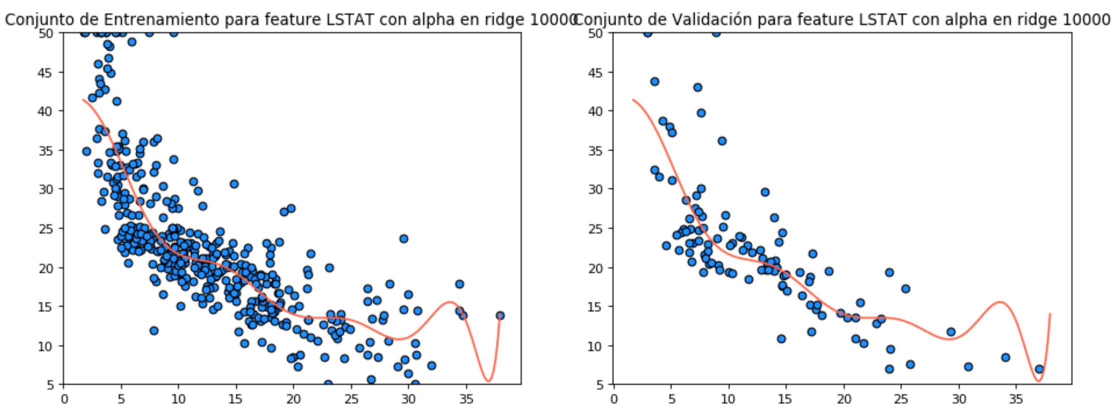

```
/Users/luisvargas/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/ridge.py:125: Lin
  overwrite_a=True).T
```




```
/Users/luisvargas/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/ridge.py:125: Lin
  overwrite_a=True).T
```




```
/Users/luisvargas/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/ridge.py:125: Lin
  overwrite_a=True).T
```



Luego de probar con polinomios de grado 11 y terminos de regularización pequeños y grandes, observamos que algunos features pueden ajustarse con un polinomio (por ejemplo RM) y otros features no se pueden ajustar correctamente ni alterando el ridge o variando el grado del polinomio, dado a que estan muy dispersos.

1.2 Clasificación binaria

La clasificación binaria tiene dos posibles etiquetas para su clasificación: SI y NO (o 0 y 1, o -1 y 1). Nuevamente, se busca entrenar utilizando el conjunto de entrenamiento (el terminado en `train`) y evaluar utilizando el conjunto de validación (el terminado en `val`). Luego se visualiza la función calculada para cada conjunto y se la compara.

Similar al caso anterior, para poder visualizar los distintos atributos y cómo estos afectan el modelo, debemos hacer uso de una selección de atributos a mano. En este caso todos los atributos son válidos, puesto que todos son numéricos. Como tenemos una clasificación, lo que buscamos ver es la frontera de decisión eligiendo distintos atributos y parámetros para distintos clasificadores. En este caso elegimos 2 atributos ya que la clase se representará por color dentro del gráfico.

1.2.1 Carga de datos

In [32]:
```python
breast_cancer_data = load_breast_cancer()

# Utilizamos aproximadamente 80% de los datos para entrenamiento y 20% para validació
shuff_data = np.random.permutation(569)
shuff_train = shuff_data[:400]
shuff_val = shuff_data[400:]

X_train = breast_cancer_data['data'][shuff_train]
X_val = breast_cancer_data['data'][shuff_val]

y_train = breast_cancer_data['target'][shuff_train]
y_val = breast_cancer_data['target'][shuff_val]

feature_map = {feature: idx for idx, feature in enumerate(breast_cancer_data['feature_

print(breast_cancer_data['DESCR'])
```

```
.. _breast_cancer_dataset:

Breast cancer wisconsin (diagnostic) dataset
--------------------------------------------

**Data Set Characteristics:**

    :Number of Instances: 569

    :Number of Attributes: 30 numeric, predictive attributes and the class

    :Attribute Information:
        - radius (mean of distances from center to points on the perimeter)
        - texture (standard deviation of gray-scale values)
        - perimeter
        - area
        - smoothness (local variation in radius lengths)
        - compactness (perimeter^2 / area - 1.0)
        - concavity (severity of concave portions of the contour)
        - concave points (number of concave portions of the contour)
        - symmetry
        - fractal dimension ("coastline approximation" - 1)

        The mean, standard error, and "worst" or largest (mean of the three
        largest values) of these features were computed for each image,
```

resulting in 30 features. For instance, field 3 is Mean Radius, field 13 is Radius SE, field 23 is Worst Radius.

- class:
    - WDBC-Malignant
    - WDBC-Benign

:Summary Statistics:

	Min	Max
radius (mean):	6.981	28.11
texture (mean):	9.71	39.28
perimeter (mean):	43.79	188.5
area (mean):	143.5	2501.0
smoothness (mean):	0.053	0.163
compactness (mean):	0.019	0.345
concavity (mean):	0.0	0.427
concave points (mean):	0.0	0.201
symmetry (mean):	0.106	0.304
fractal dimension (mean):	0.05	0.097
radius (standard error):	0.112	2.873
texture (standard error):	0.36	4.885
perimeter (standard error):	0.757	21.98
area (standard error):	6.802	542.2
smoothness (standard error):	0.002	0.031
compactness (standard error):	0.002	0.135
concavity (standard error):	0.0	0.396
concave points (standard error):	0.0	0.053
symmetry (standard error):	0.008	0.079
fractal dimension (standard error):	0.001	0.03
radius (worst):	7.93	36.04
texture (worst):	12.02	49.54
perimeter (worst):	50.41	251.2
area (worst):	185.2	4254.0
smoothness (worst):	0.071	0.223
compactness (worst):	0.027	1.058
concavity (worst):	0.0	1.252
concave points (worst):	0.0	0.291
symmetry (worst):	0.156	0.664
fractal dimension (worst):	0.055	0.208

:Missing Attribute Values: None

:Class Distribution: 212 - Malignant, 357 - Benign

```
    :Creator:  Dr. William H. Wolberg, W. Nick Street, Olvi L. Mangasarian

    :Donor: Nick Street

    :Date: November, 1995

This is a copy of UCI ML Breast Cancer Wisconsin (Diagnostic) datasets.
https://goo.gl/U2Uwz2

Features are computed from a digitized image of a fine needle
aspirate (FNA) of a breast mass.  They describe
characteristics of the cell nuclei present in the image.

Separating plane described above was obtained using
Multisurface Method-Tree (MSM-T) [K. P. Bennett, "Decision Tree
Construction Via Linear Programming." Proceedings of the 4th
Midwest Artificial Intelligence and Cognitive Science Society,
pp. 97-101, 1992], a classification method which uses linear
programming to construct a decision tree.  Relevant features
were selected using an exhaustive search in the space of 1-4
features and 1-3 separating planes.

The actual linear program used to obtain the separating plane
in the 3-dimensional space is that described in:
[K. P. Bennett and O. L. Mangasarian: "Robust Linear
Programming Discrimination of Two Linearly Inseparable Sets",
Optimization Methods and Software 1, 1992, 23-34].

This database is also available through the UW CS ftp server:

ftp ftp.cs.wisc.edu
cd math-prog/cpo-dataset/machine-learn/WDBC/

.. topic:: References

   - W.N. Street, W.H. Wolberg and O.L. Mangasarian. Nuclear feature extraction
     for breast tumor diagnosis. IS&T/SPIE 1993 International Symposium on
     Electronic Imaging: Science and Technology, volume 1905, pages 861-870,
     San Jose, CA, 1993.
   - O.L. Mangasarian, W.N. Street and W.H. Wolberg. Breast cancer diagnosis and
     prognosis via linear programming. Operations Research, 43(4), pages 570-577,
     July-August 1995.
   - W.H. Wolberg, W.N. Street, and O.L. Mangasarian. Machine learning techniques
     to diagnose breast cancer from fine-needle aspirates. Cancer Letters 77 (1994)
     163-171.
```

```
In [33]: print("Listado de atributos\n====================")
```

```
for feature in breast_cancer_data['feature_names']:
    print("- %s" % feature)
```

```
Listado de atributos
====================
- mean radius
- mean texture
- mean perimeter
- mean area
- mean smoothness
- mean compactness
- mean concavity
- mean concave points
- mean symmetry
- mean fractal dimension
- radius error
- texture error
- perimeter error
- area error
- smoothness error
- compactness error
- concavity error
- concave points error
- symmetry error
- fractal dimension error
- worst radius
- worst texture
- worst perimeter
- worst area
- worst smoothness
- worst compactness
- worst concavity
- worst concave points
- worst symmetry
- worst fractal dimension
```

```
In [34]: # Seleccionamos dos atributo de los listados en el apartado anterior, uno para el eje
         x_feature = 'mean radius'
         y_feature = 'mean texture'

         x_feature_col = feature_map[x_feature]
         y_feature_col = feature_map[y_feature]
         X_train_feature = X_train[:, [x_feature_col, y_feature_col]]
         X_val_feature = X_val[:, [x_feature_col, y_feature_col]]
```

1.2.2 Perceptrón

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Perceptron.html

```
In [35]: penalty = 'l1' # Tipo de regularización: l1 (valor absoluto), l2 (cuadrados), elastic
         alpha = 0.1 # Parámetro de regularización. También denominado como parámetro `lambda`
         max_iter = 100 # Cantidad máxima de iteraciones del algoritmo

         model = Perceptron(penalty=penalty, alpha=alpha, max_iter=max_iter)
         model.fit(X_train_feature, y_train)

         # Evaluamos el desempeño del clasificador utilizando la exactitud (accuracy) sobre el
         # de datos de entrenamiento (X_train, y_train) y lo comparamos con el de validación (
         # La exactitud toma valor en el rango [0, 1] donde más alto es mejor
         print('Exactitud para entrenamiento: %.2f' %  accuracy_score(y_train, model.predict(X_
         print('Exactitud para validación: %.2f' % accuracy_score(y_val, model.predict(X_val_fe
```

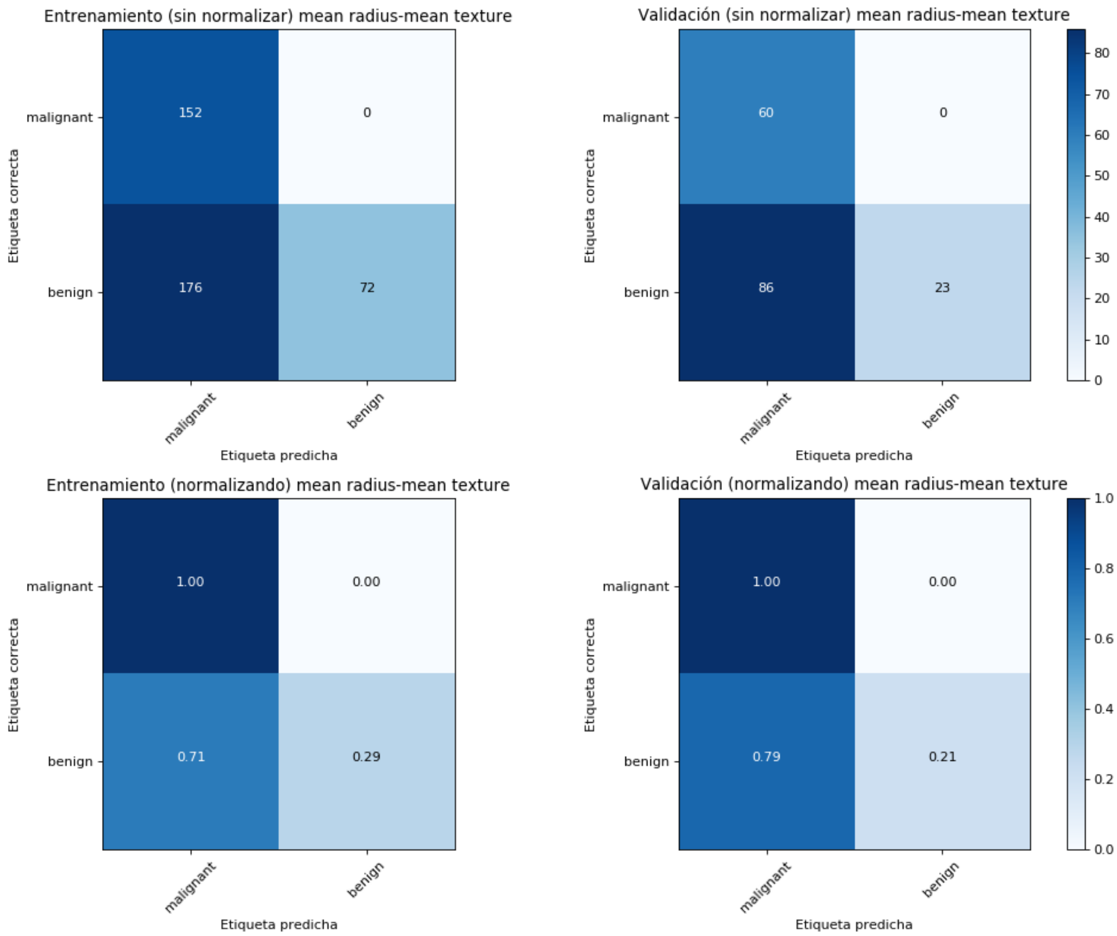
```
Exactitud para entrenamiento: 0.56
Exactitud para validación: 0.49

/Users/luisvargas/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/stochastic_gradien
  FutureWarning)
```

Matriz de confusión La matriz de confusión sirve en clasificación para ver que tanto se desviaron las instancias (de entrenamiento o de validación) de su valor real.

```
In [36]: def _plot_confusion_matrix(X_train_feature, y_train, X_val_feature, y_val, x_feature,

             plt.figure(figsize=(14, 10), dpi= 80, facecolor='w', edgecolor='k')

             plt.subplot(2, 2, 1)
             title = f'Entrenamiento (sin normalizar) {x_feature}-{y_feature}'
             plot_confusion_matrix(confusion_matrix(y_train, model.predict(X_train_feature)),
                                   classes=breast_cancer_data.target_names,
                                   title=title)
             plt.subplot(2, 2, 3)
             title = f'Entrenamiento (normalizando) {x_feature}-{y_feature}'
             plot_confusion_matrix(confusion_matrix(y_train, model.predict(X_train_feature)),
                                   classes=breast_cancer_data.target_names, normalize=True,
                                   title=title)

             plt.subplot(2, 2, 2)
             title = f'Validación (sin normalizar) {x_feature}-{y_feature}'
             plot_confusion_matrix(confusion_matrix(y_val, model.predict(X_val_feature)),
                                   classes=breast_cancer_data.target_names,
                                   title=title)
             plt.subplot(2, 2, 4)
             title = f'Validación (normalizando) {x_feature}-{y_feature}'
             plot_confusion_matrix(confusion_matrix(y_val, model.predict(X_val_feature)),
                                   classes=breast_cancer_data.target_names, normalize=True,
```

```
                            title=title)

    plt.show()
```

```
In [37]: _plot_confusion_matrix(X_train_feature, y_train, X_val_feature, y_val, x_feature, y_fe
```

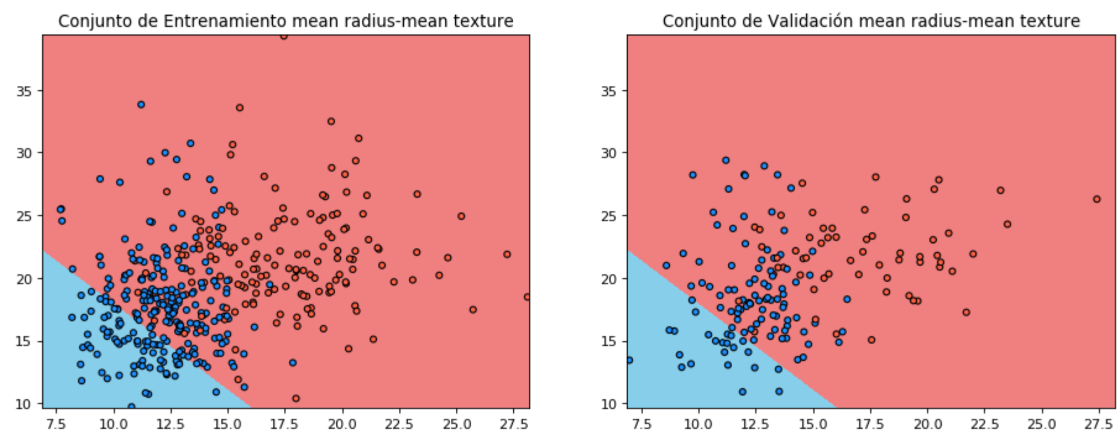


Visualización de la frontera de decisión

```
In [38]: def plot_frontera_decision(X_train_feature, y_train, X_val_feature, y_val, x_feature,
             plt.figure(figsize=(14, 5), dpi=80, facecolor='w', edgecolor='k')

             xx, yy, Z = classifier_boundary(np.r_[X_train_feature, X_val_feature], model)

             cmap_dots = ListedColormap(['tomato', 'dodgerblue'])
             cmap_back = ListedColormap(['lightcoral', 'skyblue'])

             # Conjunto de entrenamiento
```

```
    plt.subplot(1, 2, 1)
    plt.pcolormesh(xx, yy, Z, cmap=cmap_back)
    plt.scatter(X_train_feature[:, 0], X_train_feature[:, 1], c=y_train, cmap=cmap_do
    plt.xlim(xx.min(), xx.max())
    plt.ylim(yy.min(), yy.max())
    plt.title(f"Conjunto de Entrenamiento {x_feature}-{y_feature}")

    # Conjunto de validación
    plt.subplot(1, 2, 2)
    plt.pcolormesh(xx, yy, Z, cmap=cmap_back)
    plt.scatter(X_val_feature[:, 0], X_val_feature[:, 1], c=y_val, cmap=cmap_dots, edg
    plt.xlim(xx.min(), xx.max())
    plt.ylim(yy.min(), yy.max())
    plt.title(f"Conjunto de Validación {x_feature}-{y_feature}")

    plt.show()
```

```
In [39]: plot_frontera_decision(X_train_feature, y_train, X_val_feature, y_val, x_feature, y_fe
```



Probamos Perceptron aumentando la cantidad de iteraciones maximas(pues el algoritmo original itera hasta que no hay datos en el set de entrenamiento mal clasificados)

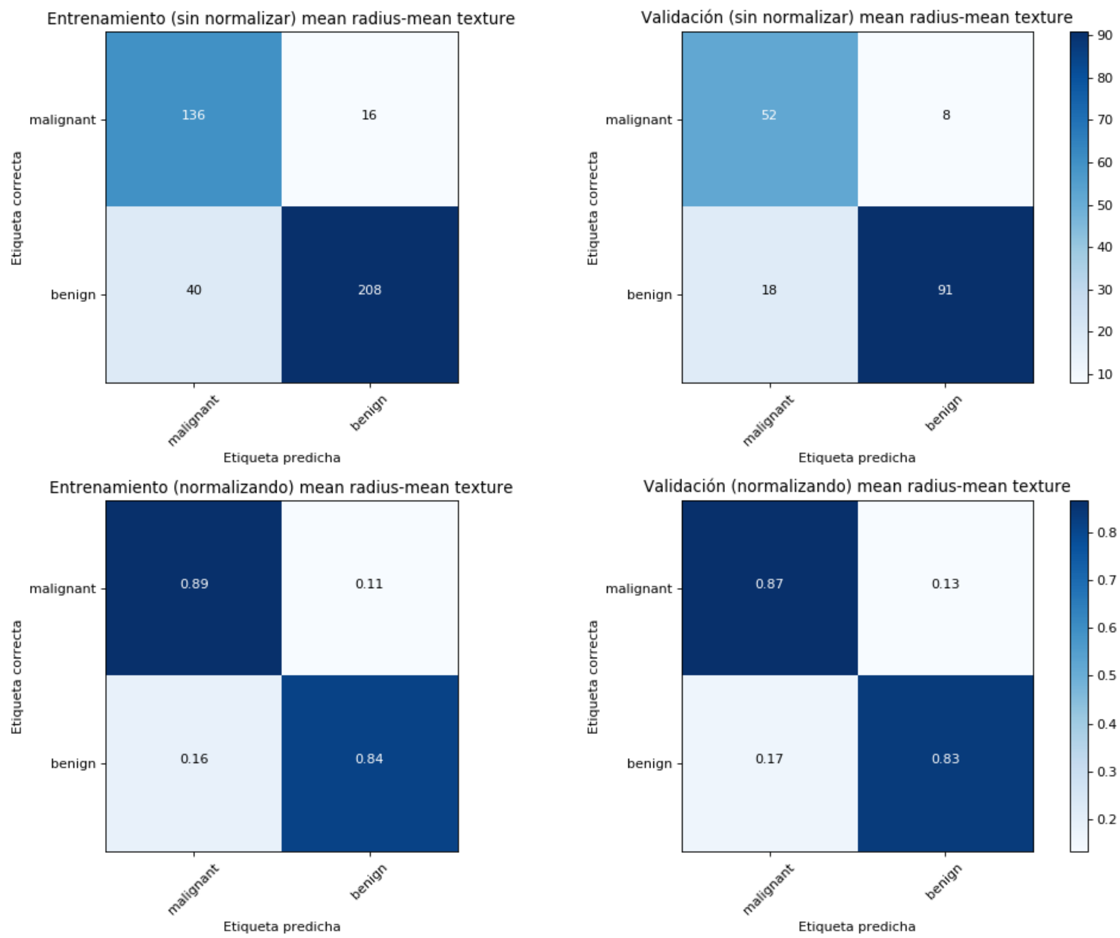
```
In [40]: penalty = 'l1' # Tipo de regularización: l1 (valor absoluto), l2 (cuadrados), elastic
         alpha = 0.1 # Parámetro de regularización. También denominado como parámetro `lambda`
         max_iter = 1000 # Cantidad máxima de iteraciones del algoritmo
         model = Perceptron(penalty=penalty, alpha=alpha, max_iter=max_iter)
         model.fit(X_train_feature, y_train)

         print('Exactitud para entrenamiento: %.2f' %  accuracy_score(y_train, model.predict(X_
         print('Exactitud para validación: %.2f' % accuracy_score(y_val, model.predict(X_val_fe
```

```
Exactitud para entrenamiento: 0.86
Exactitud para validación: 0.85
```

```
/Users/luisvargas/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/stochastic_gradie
  FutureWarning)
```

Se observa que la precicio del ajuste mejora de 0.56 a 0.86 y 0.49 a 0.85 para los set de entrenamiento y validación respectivamente

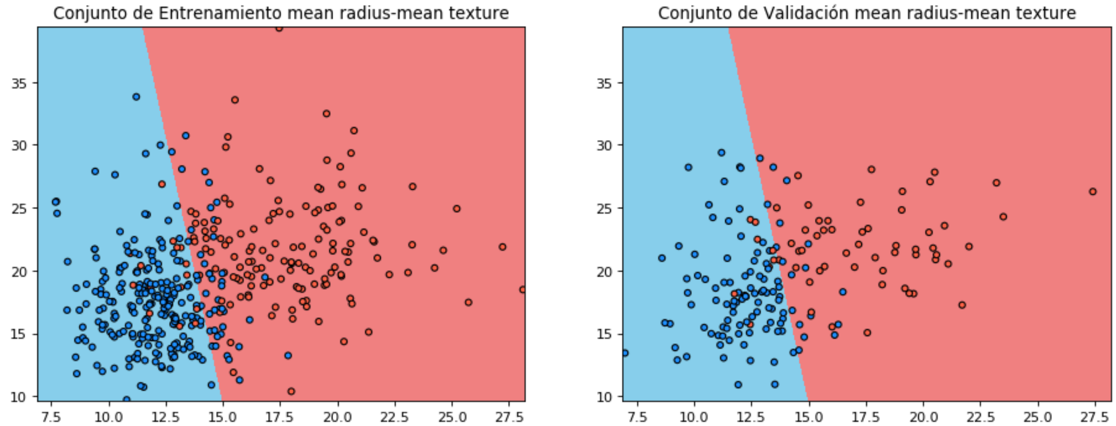
```
In [41]: _plot_confusion_matrix(X_train_feature, y_train, X_val_feature, y_val, x_feature, y_f
```



Las matrices de confusion muestran menos errores

Visualización de la frontera de decisión

```
In [42]: plot_frontera_decision(X_train_feature, y_train, X_val_feature, y_val, x_feature, y_f
```



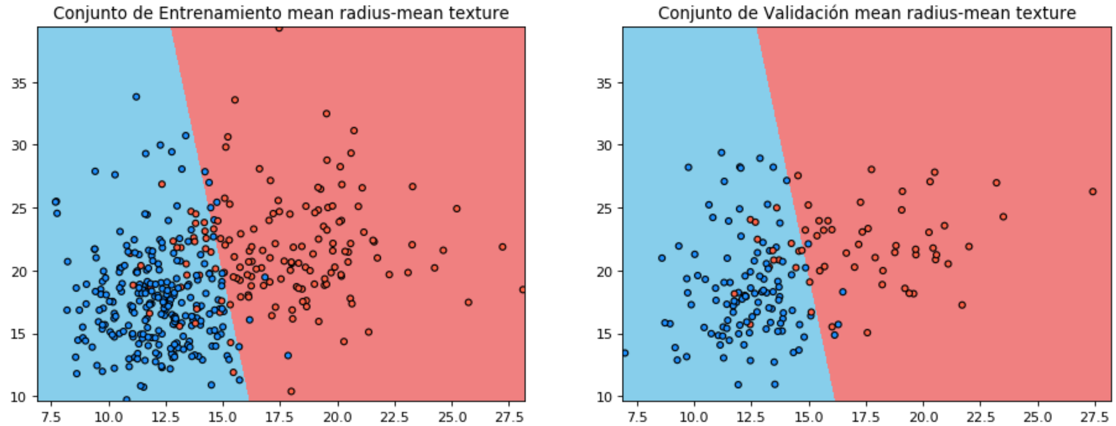
Las fronteras tambien separan de manera mas clara

Tras varias pruebas encontramos que el termino de regularización $\alpha = 0.5$ da una mejor precisión

In [43]:
```
penalty = 'l1' # Tipo de regularización: l1 (valor absoluto), l2 (cuadrados), elastic
max_iter = 1000 # Cantidad máxima de iteraciones del algoritmo
model = Perceptron(penalty=penalty, alpha=0.5, max_iter=max_iter)
model.fit(X_train_feature, y_train)

print('Exactitud para entrenamiento: %.2f' %  accuracy_score(y_train, model.predict(X_
print('Exactitud para validación: %.2f' % accuracy_score(y_val, model.predict(X_val_fe
```

```
Exactitud para entrenamiento: 0.89
Exactitud para validación: 0.88


/Users/luisvargas/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/stochastic_gradie
  FutureWarning)
```

In [44]:
```
plot_frontera_decision(X_train_feature, y_train, X_val_feature, y_val, x_feature, y_fe
```



Por ultimo probamos otros valores para el parametro penalty, pero no se logro mejorar la precición

```
In [45]: penalty = 'l2' # Tipo de regularización: l1 (valor absoluto), l2 (cuadrados), elastic
         model = Perceptron(penalty=penalty, alpha=0.5, max_iter=1000)
         model.fit(X_train_feature, y_train)

         print('Exactitud para entrenamiento: %.2f' %  accuracy_score(y_train, model.predict(X_
         print('Exactitud para validación: %.2f' % accuracy_score(y_val, model.predict(X_val_fe
```

```
Exactitud para entrenamiento: 0.62
Exactitud para validación: 0.64


/Users/luisvargas/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/stochastic_gradie
  FutureWarning)
```

```
In [46]: penalty = 'elasticnet' # Tipo de regularización: l1 (valor absoluto), l2 (cuadrados),
         model = Perceptron(penalty=penalty, alpha=0.5, max_iter=1000)
         model.fit(X_train_feature, y_train)

         print('Exactitud para entrenamiento: %.2f' %  accuracy_score(y_train, model.predict(X_
         print('Exactitud para validación: %.2f' % accuracy_score(y_val, model.predict(X_val_fe
```

```
Exactitud para entrenamiento: 0.62
Exactitud para validación: 0.64


/Users/luisvargas/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/stochastic_gradie
  FutureWarning)
```

1.2.3 Regresión logística

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

```
In [47]: penalty =  'l1'# Tipo de regularización: l1 (valor absoluto), l2 (cuadrados).
         alpha = 0.01 # Parámetro de regularización. También denominado como parámetro `lambda

         model = LogisticRegression(penalty=penalty, C=1./alpha)
         model.fit(X_train_feature, y_train)

         print('Exactitud para entrenamiento: %.2f' %  accuracy_score(y_train, model.predict(X_
         print('Exactitud para validación: %.2f' % accuracy_score(y_val, model.predict(X_val_fe
```

```
Exactitud para entrenamiento: 0.90
Exactitud para validación: 0.88
```

```
/Users/luisvargas/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:433: F
  FutureWarning)
/Users/luisvargas/anaconda3/lib/python3.7/site-packages/sklearn/svm/base.py:931: ConvergenceWar
  "the number of iterations.", ConvergenceWarning)
```

```
In [48]: penalty = 'l1'# Tipo de regularización: l1 (valor absoluto), l2 (cuadrados).
         alpha = 1000 # Parámetro de regularización. También denominado como parámetro `lambda

         model = LogisticRegression(penalty=penalty, C=1./alpha)
         model.fit(X_train_feature, y_train)

         print('Exactitud para entrenamiento: %.2f' %  accuracy_score(y_train, model.predict(X_
         print('Exactitud para validación: %.2f' % accuracy_score(y_val, model.predict(X_val_fe
```

```
Exactitud para entrenamiento: 0.38
Exactitud para validación: 0.36
```

```
/Users/luisvargas/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:433: F
  FutureWarning)
```

Un valor de α muy alto genera una baja en la precición

```
In [49]: penalty = 'l2' #l2 (cuadrados).
         alpha = 0.001

         model = LogisticRegression(penalty=penalty, C=1./alpha)
         model.fit(X_train_feature, y_train)

         print('Exactitud para entrenamiento: %.2f' %  accuracy_score(y_train, model.predict(X_
         print('Exactitud para validación: %.2f' % accuracy_score(y_val, model.predict(X_val_fe
```

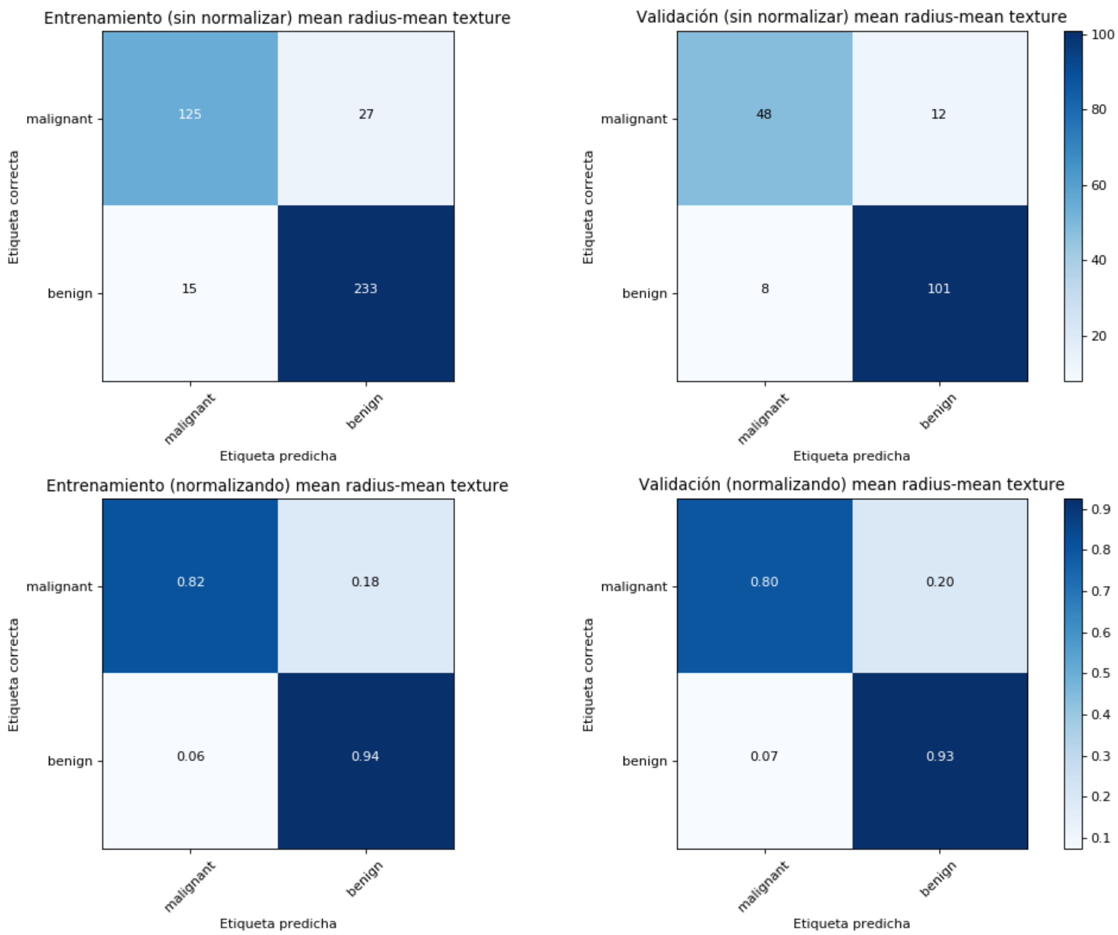
```
Exactitud para entrenamiento: 0.90
Exactitud para validación: 0.88
```

```
/Users/luisvargas/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:433: F
  FutureWarning)
```

Cambiar el tipo de penalty no afecta el mejor resultado de precición

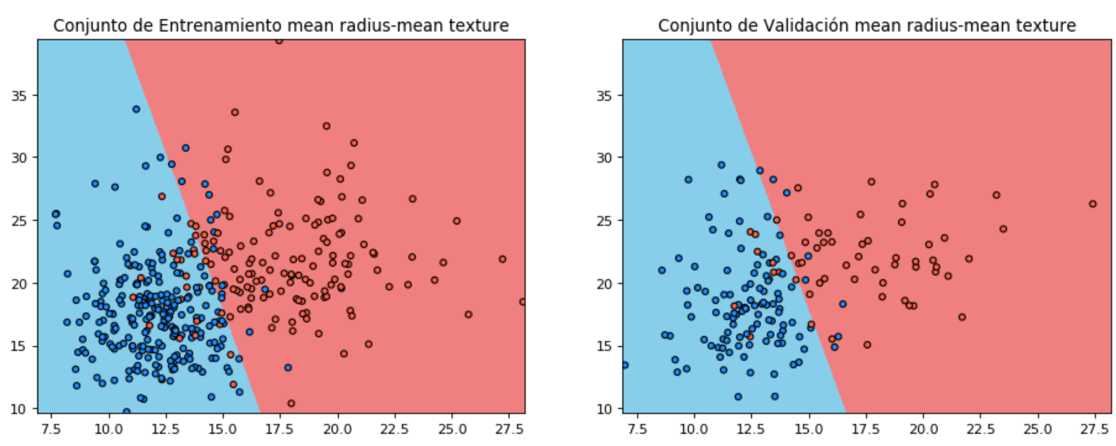
Matriz de confusión

```
In [50]: _plot_confusion_matrix(X_train_feature, y_train, X_val_feature, y_val, x_feature, y_fe
```



Visualización de la frontera de decisión

```
In [51]: plot_frontera_decision(X_train_feature, y_train, X_val_feature, y_val, x_feature, y_fe
```



1.2.4 Vecinos más cercanos

https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

```
In [52]: n_neighbors = 4    # Cantidad de vecinos a tener en cuenta
         metric = 'cosine'  # Medida de distancia. Algunas opciones: cosine, euclidean, manhat

         model = KNeighborsClassifier(n_neighbors=n_neighbors, metric=metric)
         model.fit(X_train_feature, y_train)

         print('Exactitud para entrenamiento: %.2f' %  accuracy_score(y_train, model.predict(X_
         print('Exactitud para validación: %.2f' % accuracy_score(y_val, model.predict(X_val_fe
```

```
Exactitud para entrenamiento: 0.73
Exactitud para validación: 0.60
```

```
In [53]: n_neighbors = 10
         metric = 'manhattan'

         model = KNeighborsClassifier(n_neighbors=n_neighbors, metric=metric)
         model.fit(X_train_feature, y_train)

         print('Exactitud para entrenamiento: %.2f' %  accuracy_score(y_train, model.predict(X_
         print('Exactitud para validación: %.2f' % accuracy_score(y_val, model.predict(X_val_fe
```

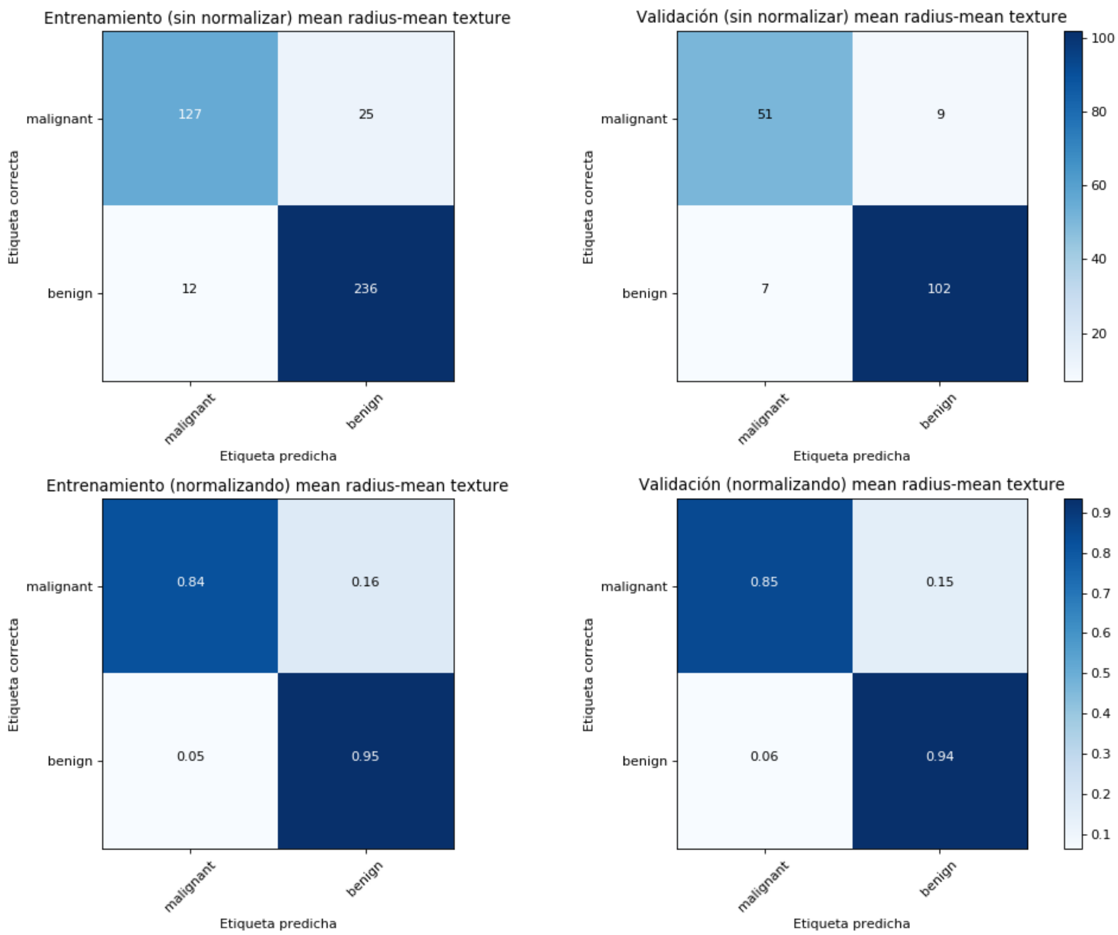
```
Exactitud para entrenamiento: 0.90
Exactitud para validación: 0.89
```

```
In [54]: n_neighbors = 15
         metric = 'euclidean'

         model = KNeighborsClassifier(n_neighbors=n_neighbors, metric=metric)
         model.fit(X_train_feature, y_train)

         print('Exactitud para entrenamiento: %.2f' %  accuracy_score(y_train, model.predict(X_
         print('Exactitud para validación: %.2f' % accuracy_score(y_val, model.predict(X_val_fe
```

```
Exactitud para entrenamiento: 0.91
Exactitud para validación: 0.91
```

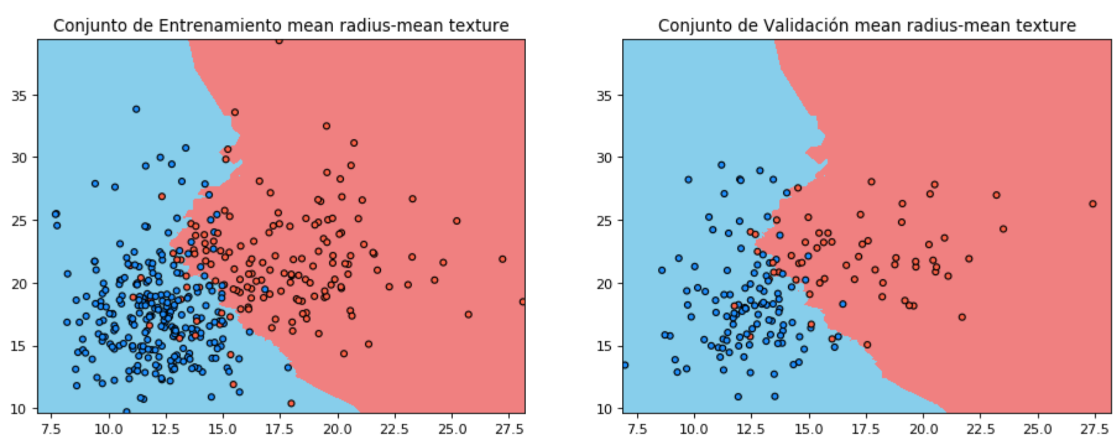
Matriz de confusión

```
In [55]: _plot_confusion_matrix(X_train_feature, y_train, X_val_feature, y_val, x_feature, y_fe
```



Visualización de la frontera de decisión

```
In [56]: plot_frontera_decision(X_train_feature, y_train, X_val_feature, y_val, x_feature, y_fe
```



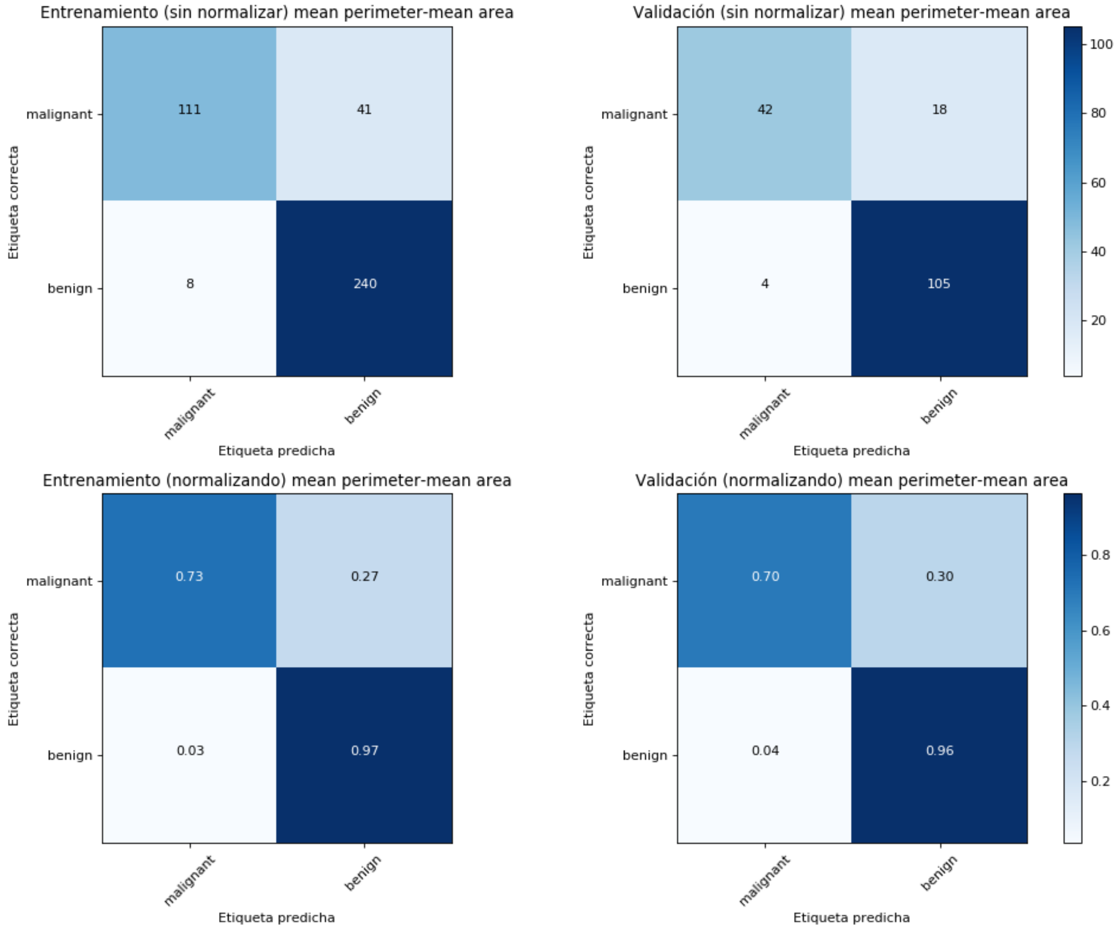
Probamos con otros features - mean perimeter - mean area
Probamos KNeighborsClassifier

In [57]:
```
# Seleccionamos dos atributo de los listados en el apartado anterior, uno para el eje
x_feature = 'mean perimeter'
y_feature = 'mean area'

x_feature_col = feature_map[x_feature]
y_feature_col = feature_map[y_feature]
X_train_feature = X_train[:, [x_feature_col, y_feature_col]]
X_val_feature = X_val[:, [x_feature_col, y_feature_col]]
```

In [58]:
```
n_neighbors = 15
metric = 'cosine'

model = KNeighborsClassifier(n_neighbors=n_neighbors, metric=metric)
model.fit(X_train_feature, y_train)

print('Exactitud para entrenamiento: %.2f' %  accuracy_score(y_train, model.predict(X_
print('Exactitud para validación: %.2f' % accuracy_score(y_val, model.predict(X_val_fe
```

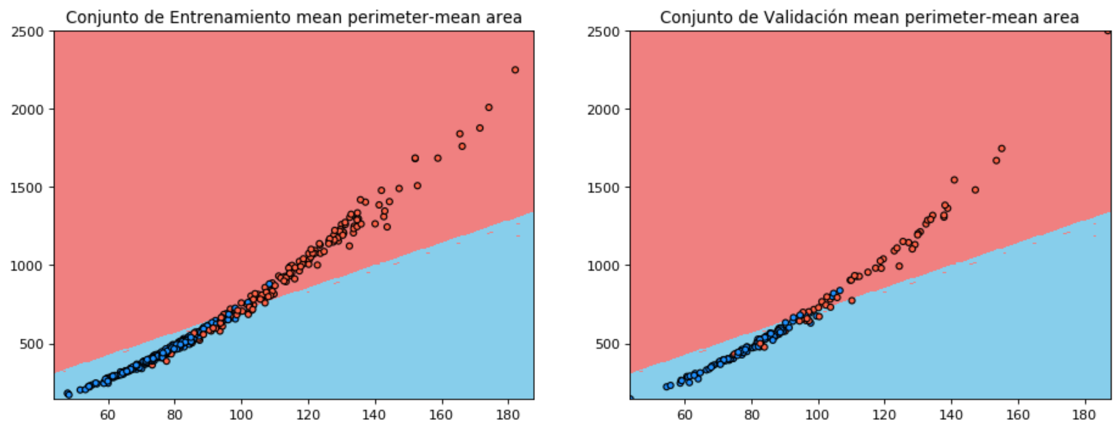
```
Exactitud para entrenamiento: 0.88
Exactitud para validación: 0.87
```

In [59]:
```
_plot_confusion_matrix(X_train_feature, y_train, X_val_feature, y_val, x_feature, y_fe
```




```
In [60]: plt.figure(figsize=(14, 5), dpi=80, facecolor='w', edgecolor='k')

         xx, yy, Z = classifier_boundary(np.r_[X_train_feature, X_val_feature], model, h=1)

         cmap_dots = ListedColormap(['tomato', 'dodgerblue'])
         cmap_back = ListedColormap(['lightcoral', 'skyblue'])

         # Conjunto de entrenamiento
         plt.subplot(1, 2, 1)
         plt.pcolormesh(xx, yy, Z, cmap=cmap_back)
         plt.scatter(X_train_feature[:, 0], X_train_feature[:, 1], c=y_train, cmap=cmap_dots,
         plt.xlim(xx.min(), xx.max())
         plt.ylim(yy.min(), yy.max())
         plt.title(f"Conjunto de Entrenamiento {x_feature}-{y_feature}")

         # Conjunto de validación
         plt.subplot(1, 2, 2)
         plt.pcolormesh(xx, yy, Z, cmap=cmap_back)
```

```
plt.scatter(X_val_feature[:, 0], X_val_feature[:, 1], c=y_val, cmap=cmap_dots, edgecol
plt.xlim(xx.min(), xx.max())
plt.ylim(yy.min(), yy.max())
plt.title(f"Conjunto de Validación {x_feature}-{y_feature}")

plt.show()
```



1.3 Clasificación multiclase

Ahora veremos clasificación multiclase. Muy similar al caso anterior, con la diferencia de que en este caso hay más de dos etiquetas posibles para clasificación. Se utilizará el método `one-vs-all` (o también `one-vs-rest`) para hacer posible la clasificación.

Una vez más tenemos que decidir dos features para poder visualizar los modelos.

1.3.1 Carga de datos

```
In [61]: iris_data = load_iris()

         # Utilizamos aproximadamente 80% de los datos para entrenamiento y 20% para validació
         shuff_data = np.random.permutation(150)
         shuff_train = shuff_data[:120]
         shuff_val = shuff_data[120:]

         X_train = iris_data['data'][shuff_train]
         X_val = iris_data['data'][shuff_val]

         y_train = iris_data['target'][shuff_train]
         y_val = iris_data['target'][shuff_val]

         feature_map = {feature: idx for idx, feature in enumerate(iris_data['feature_names'])}

         print(iris_data['DESCR'])
```

```
.. _iris_dataset:

Iris plants dataset
--------------------

**Data Set Characteristics:**

    :Number of Instances: 150 (50 in each of three classes)
    :Number of Attributes: 4 numeric, predictive attributes and the class
    :Attribute Information:
        - sepal length in cm
        - sepal width in cm
        - petal length in cm
        - petal width in cm
        - class:
                - Iris-Setosa
                - Iris-Versicolour
                - Iris-Virginica

    :Summary Statistics:

    ============== ==== ==== ======= ===== ====================
                    Min  Max   Mean    SD   Class Correlation
    ============== ==== ==== ======= ===== ====================
    sepal length:   4.3  7.9   5.84   0.83    0.7826
    sepal width:    2.0  4.4   3.05   0.43   -0.4194
    petal length:   1.0  6.9   3.76   1.76    0.9490  (high!)
    petal width:    0.1  2.5   1.20   0.76    0.9565  (high!)
    ============== ==== ==== ======= ===== ====================

    :Missing Attribute Values: None
    :Class Distribution: 33.3% for each of 3 classes.
    :Creator: R.A. Fisher
    :Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
    :Date: July, 1988

The famous Iris database, first used by Sir R.A. Fisher. The dataset is taken
from Fisher's paper. Note that it's the same as in R, but not as in the UCI
Machine Learning Repository, which has two wrong data points.

This is perhaps the best known database to be found in the
pattern recognition literature.  Fisher's paper is a classic in the field and
is referenced frequently to this day.  (See Duda & Hart, for example.)  The
data set contains 3 classes of 50 instances each, where each class refers to a
type of iris plant.  One class is linearly separable from the other 2; the
latter are NOT linearly separable from each other.

.. topic:: References
```

```
   - Fisher, R.A. "The use of multiple measurements in taxonomic problems"
     Annual Eugenics, 7, Part II, 179-188 (1936); also in "Contributions to
     Mathematical Statistics" (John Wiley, NY, 1950).
   - Duda, R.O., & Hart, P.E. (1973) Pattern Classification and Scene Analysis.
     (Q327.D83) John Wiley & Sons.  ISBN 0-471-22361-1.  See page 218.
   - Dasarathy, B.V. (1980) "Nosing Around the Neighborhood: A New System
     Structure and Classification Rule for Recognition in Partially Exposed
     Environments".  IEEE Transactions on Pattern Analysis and Machine
     Intelligence, Vol. PAMI-2, No. 1, 67-71.
   - Gates, G.W. (1972) "The Reduced Nearest Neighbor Rule".  IEEE Transactions
     on Information Theory, May 1972, 431-433.
   - See also: 1988 MLC Proceedings, 54-64.  Cheeseman et al"s AUTOCLASS II
     conceptual clustering system finds 3 classes in the data.
   - Many, many more ...
```

In [62]:
```python
print("Listado de atributos\n====================")
for feature in iris_data['feature_names']:
    print("- %s" % feature)
```

```
Listado de atributos
====================
- sepal length (cm)
- sepal width (cm)
- petal length (cm)
- petal width (cm)
```

In [63]:
```python
# Seleccionamos dos atributo de los listados en el apartado anterior, uno para el eje
# TODO: Cambiar los atributos y ver como se modifica el resultado
x_feature = 'sepal length (cm)'
y_feature = 'sepal width (cm)'

x_feature_col = feature_map[x_feature]
y_feature_col = feature_map[y_feature]
X_train_feature = X_train[:, [x_feature_col, y_feature_col]]
X_val_feature = X_val[:, [x_feature_col, y_feature_col]]
```

1.3.2 Regresión logística

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

In [64]:
```python
penalty = 'l1' # Tipo de regularización: l1 (valor absoluto), l2 (cuadrados).
alpha =  0.01 # Parámetro de regularización. También denominado como parámetro `lambd

model = LogisticRegression(penalty=penalty, C=1./alpha, multi_class='ovr')
model.fit(X_train_feature, y_train)
```

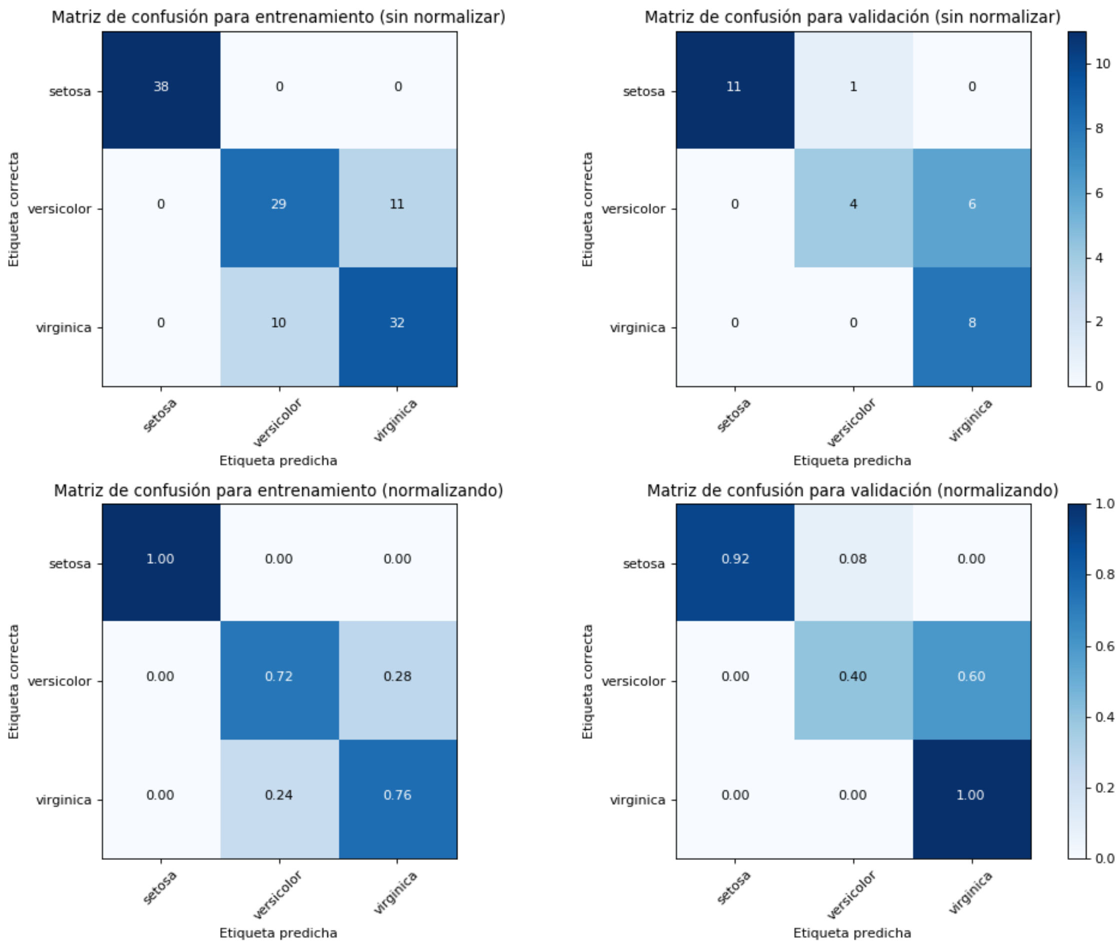
```
print('Exactitud para entrenamiento: %.2f' %  accuracy_score(y_train, model.predict(X_
print('Exactitud para validación: %.2f' % accuracy_score(y_val, model.predict(X_val_fe
```

```
Exactitud para entrenamiento: 0.82
Exactitud para validación: 0.77
```

```
/Users/luisvargas/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:433: F
  FutureWarning)
/Users/luisvargas/anaconda3/lib/python3.7/site-packages/sklearn/svm/base.py:931: ConvergenceWar
  "the number of iterations.", ConvergenceWarning)
```

Matriz de confusión

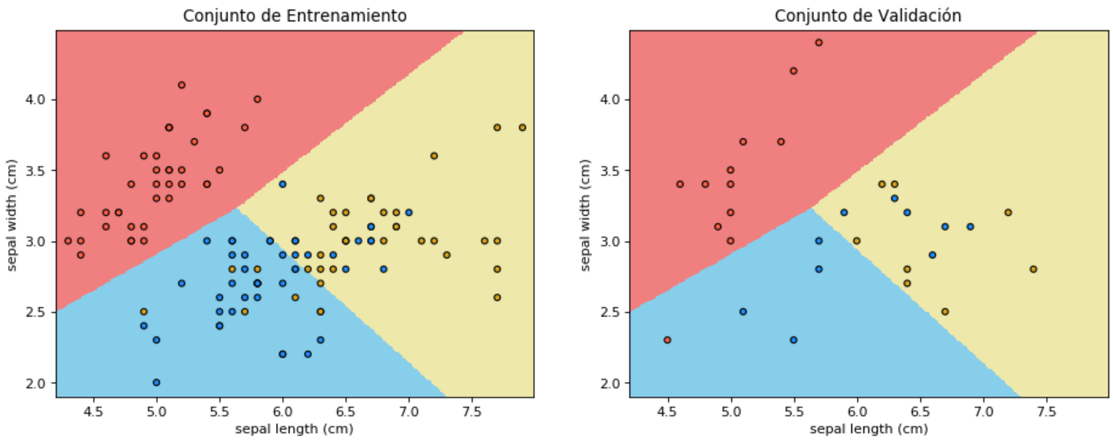
```
In [65]: plt.figure(figsize=(14, 10), dpi= 80, facecolor='w', edgecolor='k')

         plt.subplot(2, 2, 1)
         plot_confusion_matrix(confusion_matrix(y_train, model.predict(X_train_feature)),
                               classes=iris_data.target_names,
                               title='Matriz de confusión para entrenamiento (sin normalizar)')
         plt.subplot(2, 2, 3)
         plot_confusion_matrix(confusion_matrix(y_train, model.predict(X_train_feature)),
                               classes=iris_data.target_names, normalize=True,
                               title='Matriz de confusión para entrenamiento (normalizando)')

         plt.subplot(2, 2, 2)
         plot_confusion_matrix(confusion_matrix(y_val, model.predict(X_val_feature)),
                               classes=iris_data.target_names,
                               title='Matriz de confusión para validación (sin normalizar)')
         plt.subplot(2, 2, 4)
         plot_confusion_matrix(confusion_matrix(y_val, model.predict(X_val_feature)),
                               classes=iris_data.target_names, normalize=True,
                               title='Matriz de confusión para validación (normalizando)')

         plt.show()
```



Visualización de la frontera de decisión

In [66]:
```python
plt.figure(figsize=(14, 5), dpi=80, facecolor='w', edgecolor='k')

xx, yy, Z = classifier_boundary(np.r_[X_train_feature, X_val_feature], model)

cmap_dots = ListedColormap(['tomato', 'dodgerblue', 'goldenrod'])
cmap_back = ListedColormap(['lightcoral', 'skyblue', 'palegoldenrod'])

# Conjunto de entrenamiento
plt.subplot(1, 2, 1)
plt.pcolormesh(xx, yy, Z, cmap=cmap_back)
plt.scatter(X_train_feature[:, 0], X_train_feature[:, 1], c=y_train, cmap=cmap_dots, 
plt.xlim(xx.min(), xx.max())
plt.ylim(yy.min(), yy.max())
plt.title("Conjunto de Entrenamiento")
plt.xlabel(x_feature)
plt.ylabel(y_feature)
```

```
# Conjunto de validación
plt.subplot(1, 2, 2)
plt.pcolormesh(xx, yy, Z, cmap=cmap_back)
plt.scatter(X_val_feature[:, 0], X_val_feature[:, 1], c=y_val, cmap=cmap_dots, edgecol
plt.xlim(xx.min(), xx.max())
plt.ylim(yy.min(), yy.max())
plt.title("Conjunto de Validación")
plt.xlabel(x_feature)
plt.ylabel(y_feature)

plt.show()
```



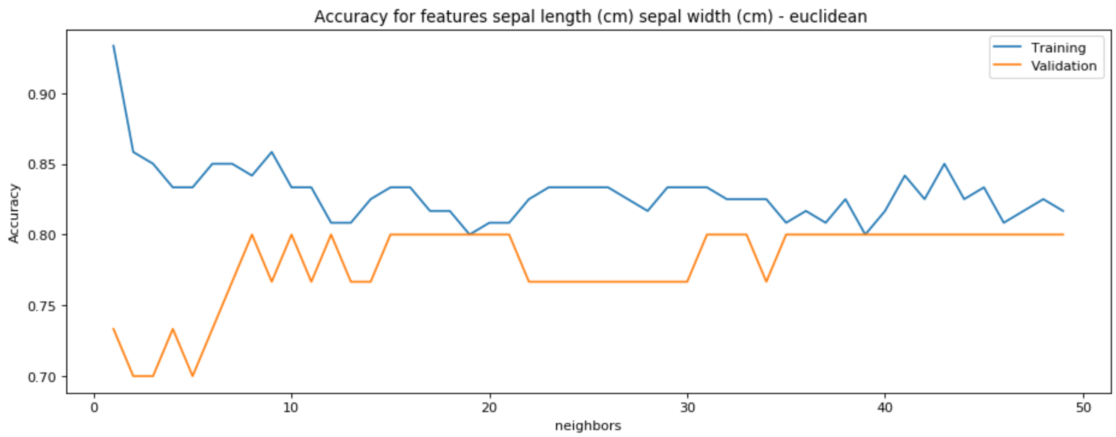
1.3.3 Vecinos más cercanos

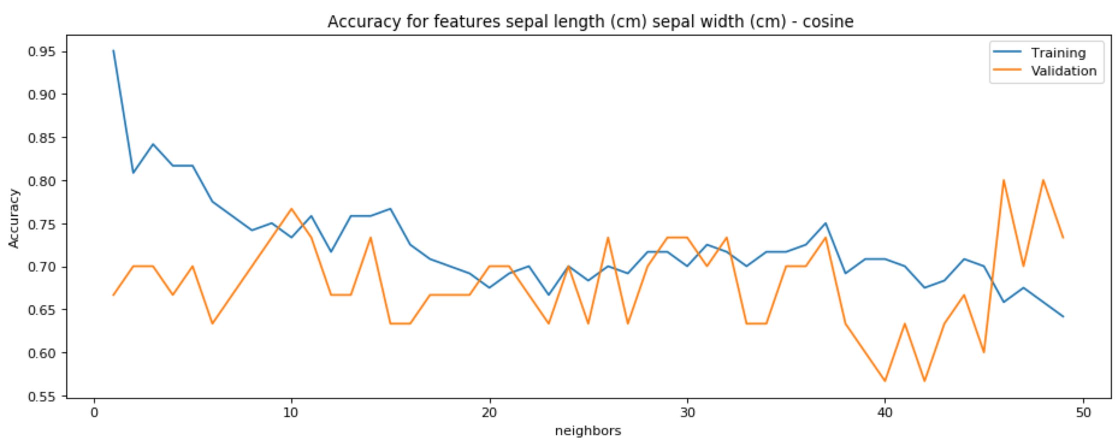
https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

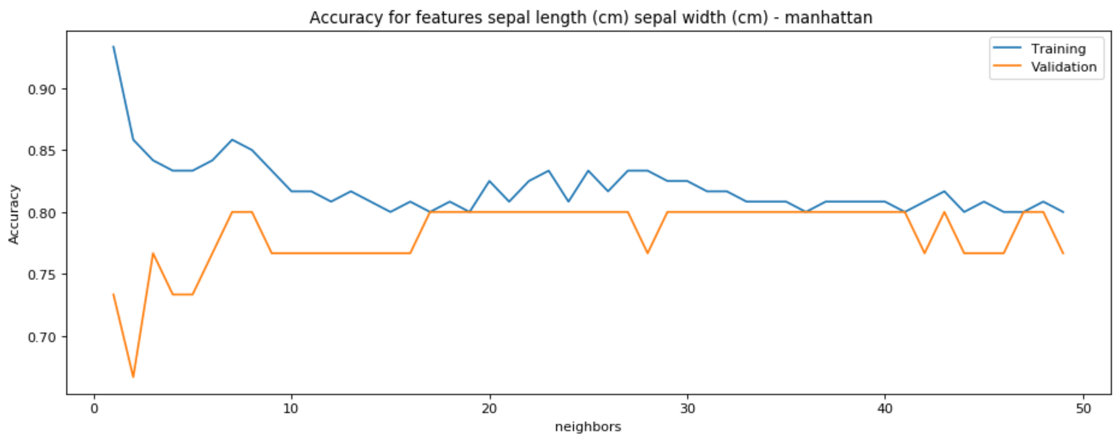
```
In [67]: def get_accuracy_knc(X_train_feature, y_train, X_val_feature, y_val, n_neighbors, met
             model = KNeighborsClassifier(n_neighbors=n_neighbors, metric=metric)
             model.fit(X_train_feature, y_train)
             training_accuracy_score =  accuracy_score(y_train, model.predict(X_train_feature)
             val_accuracy_score = accuracy_score(y_val, model.predict(X_val_feature))
             return training_accuracy_score, val_accuracy_score
```

```
In [68]: x = range(1, 50)
         for metric in ['euclidean', 'cosine', 'manhattan']:
             accuracy_training = []
             accuracy_val = []
             for n_neighbors in x:
                 accuracy_training.append(get_accuracy_knc(X_train_feature, y_train, X_val_fea
                 accuracy_val.append(get_accuracy_knc(X_train_feature, y_train, X_val_feature,
```

```python
plt.figure(figsize=(14, 5), dpi= 80, facecolor='w', edgecolor='k')
plt.plot(x, accuracy_training, label='Training')
plt.plot(x, accuracy_val, label='Validation')
plt.title(f" Accuracy for features {x_feature} {y_feature} - {metric}")
plt.legend(loc='upper right')
plt.xlabel("neighbors")
plt.ylabel("Accuracy")
plt.show()
```







20 neighbors + distancia 'euclidean' parece dar una buena preción

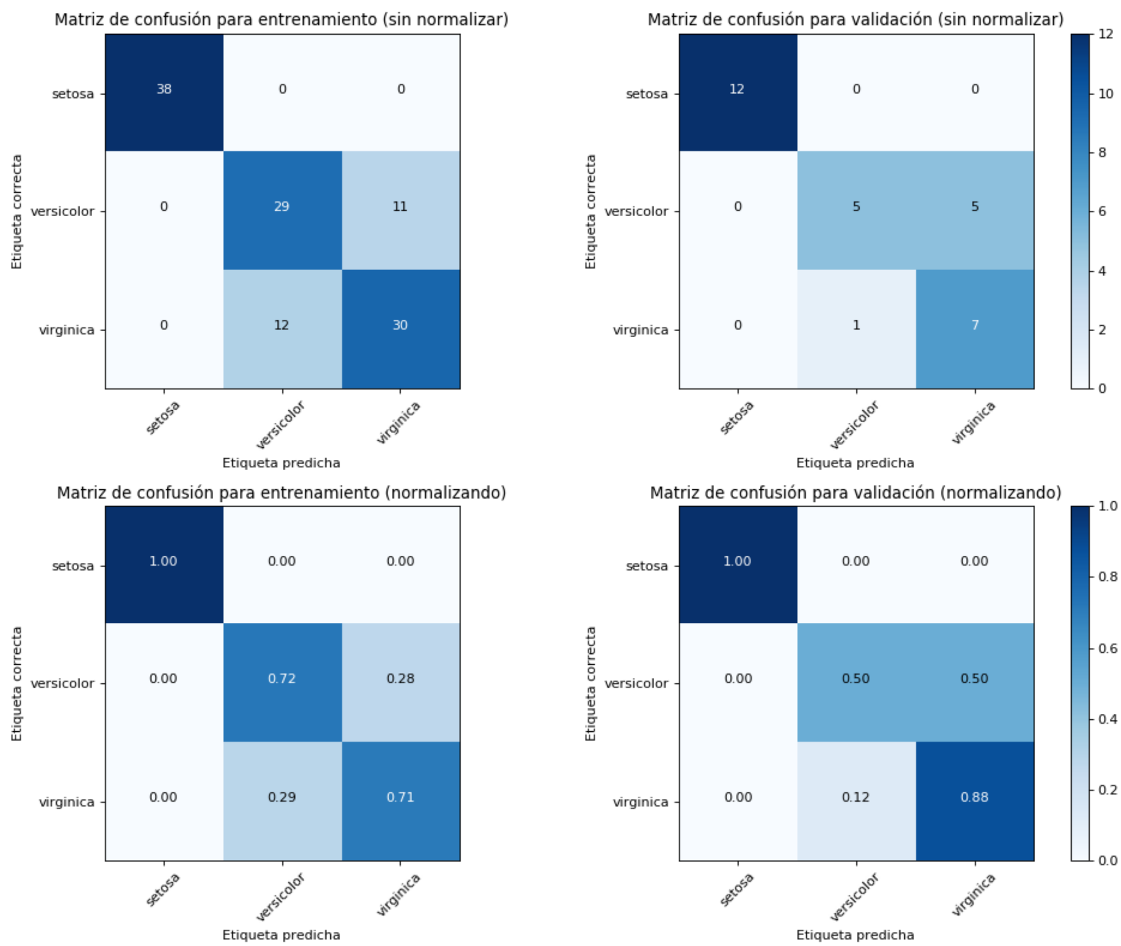
```
In [69]: n_neighbors = 20 # Cantidad de vecinos a tener en cuenta
         metric = 'euclidean' # Medida de distancia. Algunas opciones: cosine, euclidean, manha

         model = KNeighborsClassifier(n_neighbors=n_neighbors, metric=metric)
         model.fit(X_train_feature, y_train)

         print('Exactitud para entrenamiento: %.2f' %  accuracy_score(y_train, model.predict(X_
         print('Exactitud para validación: %.2f' % accuracy_score(y_val, model.predict(X_val_fe
```

```
Exactitud para entrenamiento: 0.81
Exactitud para validación: 0.80
```

Matriz de confusión

```
In [70]: plt.figure(figsize=(14, 10), dpi= 80, facecolor='w', edgecolor='k')

         plt.subplot(2, 2, 1)
         plot_confusion_matrix(confusion_matrix(y_train, model.predict(X_train_feature)),
                               classes=iris_data.target_names,
                               title='Matriz de confusión para entrenamiento (sin normalizar)')
         plt.subplot(2, 2, 3)
         plot_confusion_matrix(confusion_matrix(y_train, model.predict(X_train_feature)),
                               classes=iris_data.target_names, normalize=True,
                               title='Matriz de confusión para entrenamiento (normalizando)')

         plt.subplot(2, 2, 2)
         plot_confusion_matrix(confusion_matrix(y_val, model.predict(X_val_feature)),
                               classes=iris_data.target_names,
                               title='Matriz de confusión para validación (sin normalizar)')
         plt.subplot(2, 2, 4)
```

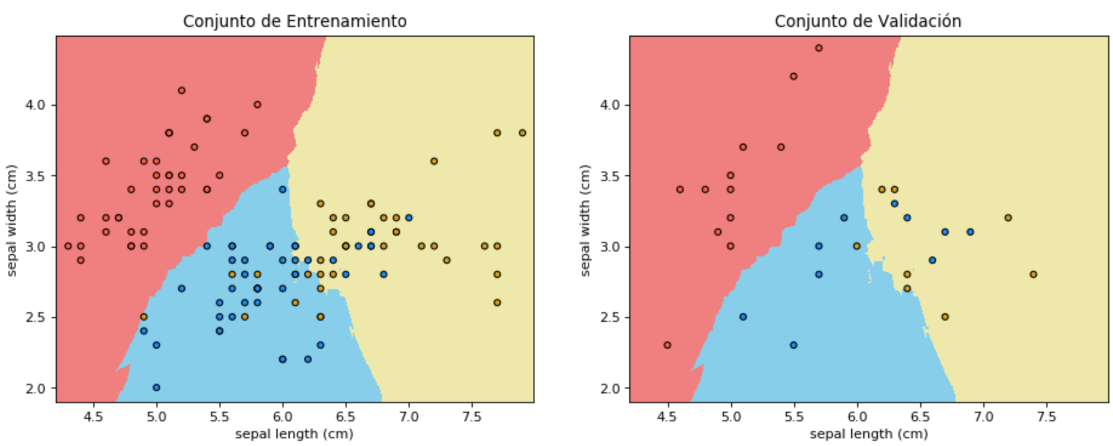
```
plot_confusion_matrix(confusion_matrix(y_val, model.predict(X_val_feature)),
                      classes=iris_data.target_names, normalize=True,
                      title='Matriz de confusión para validación (normalizando)')

plt.show()
```



Visualización de la frontera de decisión

```
In [71]: plt.figure(figsize=(14, 5), dpi=80, facecolor='w', edgecolor='k')

         xx, yy, Z = classifier_boundary(np.r_[X_train_feature, X_val_feature], model)

         cmap_dots = ListedColormap(['tomato', 'dodgerblue', 'goldenrod'])
         cmap_back = ListedColormap(['lightcoral', 'skyblue', 'palegoldenrod'])

         # Conjunto de entrenamiento
         plt.subplot(1, 2, 1)
         plt.pcolormesh(xx, yy, Z, cmap=cmap_back)
```

```
plt.scatter(X_train_feature[:, 0], X_train_feature[:, 1], c=y_train, cmap=cmap_dots, e
plt.xlim(xx.min(), xx.max())
plt.ylim(yy.min(), yy.max())
plt.title("Conjunto de Entrenamiento")
plt.xlabel(x_feature)
plt.ylabel(y_feature)

# Conjunto de validación
plt.subplot(1, 2, 2)
plt.pcolormesh(xx, yy, Z, cmap=cmap_back)
plt.scatter(X_val_feature[:, 0], X_val_feature[:, 1], c=y_val, cmap=cmap_dots, edgecol
plt.xlim(xx.min(), xx.max())
plt.ylim(yy.min(), yy.max())
plt.title("Conjunto de Validación")
plt.xlabel(x_feature)
plt.ylabel(y_feature)

plt.show()
```



In [ ]: