Expansiones Basales y No-linealidades

¡Comencemos!

Competencias

- Entender los conceptos asociados al tratamiento de no linealidades a través de expansiones de base.
- Ser capaz de implementar modelos aditivos generalizados.
- Comprender la importancia de los hiper parámetros de un modelo y las dificultades asociadas a su elección.
- Entender la importancia de las transformaciones no lineales de atributos.

Introducción

En vista a nuestros conocimientos sobre regresión lineal y regularizadores, una pregunta común es ¿Por qué a pesar de sus limitantes, existe tanta teoría asociada a la regresión lineal? Ya sabemos que la regresión busca presentar una recta de mejor ajuste que resume un punto característico en un intervalo de datos. Una de las principales limitantes de esta aproximación es la forma funcional que impone un alto sesgo en nuestros datos de entrenamiento. Existen medidas paliativas como implementar términos polinomiales y/o splines, que implica dedicar parte importante en investigar cuál es la mejor representación funcional en los datos de entrenamiento.

Un contrapunto asociado al uso de splines o términos polinomiales en el componente sistemático de nuestra regresión es la probabilidad de generar un ajuste perfecto en los datos que ya tenemos conocimiento. Lo que necesitamos es un método que permita describir el comportamiento general de un fenómeno con un margen de error controlado. Nuestro objetivo como investigadores no es generar predicciones perfectas con un desempeño perfecto, lo que intentamos es generar predicciones con un nivel de certeza cercano al 100%.

Tener un modelo entrenado con un desempeño alto no es condición suficiente para su validación externa y asegurar conocimiento sobre el fenómeno estudiado. Este comportamiento puede generar falsa seguridad en el investigador, generando supuestos falsos sobre el modelo. Aspectos como la causalidad entre atributos y el vector obetivo no se puede asegurar mediante las métricas de desempeño. Con el advenimiento de métodos de caja negra, esto se transforma en un arma de doble filo.

¡Vamos con todo!

Los Modelos Aditivos Generalizados buscan superar las limitantes impuestas de la forma funcional de los modelos lineales (sean en su variante generalizada o normal). La rigidez de asumir un comportamiento lineal fijo para todo atributo en nuestro conjunto de datos genera situaciones donde la forma funcional presenta un alto sesgo y no captura de manera adecuada el comportamiento del modelo.

Sabemos que en una regresión lineal buscamos resolver una ecuación con forma:

$$y = \beta_0 + X\overline{\beta}$$

El principal problema con esa forma es que asume monotonicidad estricta tanto en X como en Y. Una de las primeras soluciones es especificar la forma funcional en el modelo mediante la inclusión de términos polinomiales. El problema con la implementación de términos polinomiales es que la optimización de la función objetivo dependerá fuertemente de los datos, restringiendo su capacidad de generalización.

```
import pandas as pd
import numpy as np
import seaborn as sn
import matplotlib.pyplot as plt
import lec2_graphs as afx
import warnings
warnings.filterwarnings(action='ignore')
plt.rcParams["figure.figsize"] = (10, 6) # Tamaño gráficos
plt.rcParams["figure.dpi"] = 80 # resolución gráficos
sn.set_style('darkgrid')

afx.polynomial_degrees()
```

La figura generada con `afx.polynomial_degrees()` presenta la recta de ajuste en una serie de datos con una fuerte no linealidad. Cada uno de los gráficos representa la inclusión de términos en el lado derecho de la ecuación. La cantidad de términos está representado con el operador.

$$\sum_{j=1}^{t}$$

Una regresión sin términos falla en capturar gran parte de la dinámica. En el extremo opuesto, cuando agregamos alrededor de 20 términos, nuestra curva se adapta muy bien a los datos existentes, pero no tenemos certeza sobre cómo serán los siguientes datos que deseamos predecir. El óptimo se encuentra entre 3 y 5 polinomios, dado que se ajustan relativamente bien a los datos mientras no exacerban la varianza (representada con el área sombreada de los intervalos de confianza).

Otra solución es mediante los Modelos Lineales Generalizados. Con esta familia de modelos podemos incorporar otras distribuciones en nuestro vector objetivo, e incluir una función de vínculo que permita asociada los valores esperados de

$$E\{y\}$$

con una combinación lineal de parámetros

$$X\overline{\beta}$$

Con esta estrategia podemos capturar un proceso de generación de datos distinta a la requerida por la regresión lineal, pero no hemos trabajado sobre la restricción monotónica de los parámetros.

Digresión: GAM vs OLS

Si los Modelos Aditivos Generalizados presentan tantas virtudes, ¿por qué seguimos implementando el modelo de regresión lineal?

Cosma Shalizi (NA) sugiere la existencia de dos razones por las cuales preferimos modelos lineales:

- Por lo general los análisis se guían por teoría científica que asume el comportamiento entre variables como lineal.

- Desde la eficiencia computacional, cuando incluímos términos distintos a los lineales estamos aumentando la complejidad del modelo, que muchas veces repercute en tiempo y recursos de procesamiento.

  En una gran parte de los casos cuando se implementan modelos, ninguno de los puntos son justificables: La teoría no nos dice que debemos esperar linealidad estricta (y muchas veces, un buen primer paso es la exploración de éste supuesto en los datos), y hoy en día el poder computacional es lo suficientemente alto como para ejecutar modelos con una mayor complejidad en minutos.

La gente implementa modelos lineales porque no tiene los elementos necesarios como para sentirse cómoda implementando GAM.

Modelos Aditivos Generalizados (Generalized Additive Models)

Los Modelos Aditivos Generalizados surgen como un compromiso entre ambas soluciones, partiendo del supuesto que nuestra función de regresión candidata es desconocida (Hastie et al. 2009).

La forma general de los GAMs es:

$$g(E(Y)) = \beta_0 + f_1(x_1) + f_2(x_2) + ... + f_m(x_m)$$

Donde:

- β_0 es el intercepto.
- $g(.)$ es llamada función de vínculo que permite relacionar el valor esperado de la variable objetivo con las variables predictivas.
- $f_i(x_i)$ es el output de la i-ésima función (spline) sobre el i-ésimo atributo.

La forma de los GAMs es bastante similar a la de un modelo lineal. La diferencia es que en un modelo lineal todas las funciones:

$$f_i(.)$$

Son las funciones de identidad. La potencia de los GAMs radica en la posibilidad de utilizar distintas funciones para tratar de distinta manera ciertos atributos, por ejemplo, a través de funciones no lineales. La no linealidad permite a la curva de regresión responder de forma mucho más flexible a oscilaciones periódicas o fluctuaciones en la variable objetivo.

Entrenamiento de GAMs

Ya tenemos la formulación del modelo, ahora tenemos que especificar como aprenderá, esto lo hacemos especificando una función de costo y pérdida. El método que utilizará el modelo para aprender sus parámetros por lo general no se considera parte del modelo pues es una de las decisiones que uno debe tomar. Un modelo puede aprender sus parámetros utilizando muchos métodos distintos, el resultado (en teoría), debiese ser el mismo.

Los GAMs clásicos buscan estimar las funciones:

$$f_i(x_i)$$

Mediante la minimización de la siguiente función objetivo:

$$argim \sum_{i=1}^{n} (y_1 - \sum_{j=1}^{p} f_j(x_{ij}))^2 + \lambda \sum_{j}^{p} \int f_j^{''}(t)^2 dt$$

Si nos fijamos bien, el término de penalización (el de la derecha) está penalizando la concavidad de la spline, si la spline es muy "exagerada" se penalizará más, de hecho, si la spline es una línea recta el modelo no es penalizado. El factor λ es el coeficiente de tradeoff entre la concavidad de la spline propuesta y la penalización asignada, mientras más alto este hiper parámetro, más se penalizará el modelo por las splines que sean curvas y tenderá a splines más rectas.

La penalización a la segunda derivada de la spline es una forma de evitar el overfitting al que suelen tender los splines.

Digresión: ¿Cómo obtenemos las funciones de identidad?

Hasta ahora asumimos que las funciones de vínculo existen y permiten suavizar cada término, ignorando cómo se obtienen.

Un problema frecuente con este modelo es que el término constante β_0 no es identificable (esto es que no tiene una solución única), dificultando la obtención de estimaciones y sus funciones correspondientes. Para facilitar la obtención, Hastie et al. (2009) reportan el uso del algoritmo **Backfitting**.

grandes rasgos, el algoritmo busca lo siguiente:

1. Asumimos que β_0 no es identificable, lo igualamos al promedio de las observaciones de **y**.
2. Aplicamos una función de suavización a x_1 para estimar los vectores objetivos y obtener un suavizador candidato $\widehat{f}_k$.
3. Actualizamos el valor de β_0 en base al paso 2.
4. Iteramos los pasos 2 y 3 hasta que $\widehat{f}_k$ se estabilice o satisfaga algún criterio de tolerancia.

Implementando un Modelo Aditivo Generalizado con `pygam`

Para implementar nuestro modelo GAM, utilizaremos la librería `pygam`. Por defecto, ésta no viene incluida por defecto en la distribución de Anaconda y debemos incorporar de la siguiente manera: `conda install -c conda-forge pygam`. Una vez instalada, procedemos con los pasos usuales:

Para este ejemplo trabajaremos con la base de datos sobre los precios de inmuebles en Boston que implementamos en la lectura de Regularización Paramétrica, y generaremos un procedimiento de limpieza idéntico al de esa lectura.

```
from sklearn.preprocessing import StandardScaler
df = pd.read_csv('kc_house_data.csv')
# Vamos a eliminar ciertas columnas que son irrelevantes para nuestro
analisis
df.drop(['zipcode', 'id', 'date'], axis = 1, inplace = True)

# Vamos a utilizar un subconjunto de las columnas para hacer el ejemplo
# mas expedito
sub = df[['bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot']]
```

También generamos las muestras de entrenamiento y validación, dejando un 30% de la muestra reservada para validar el modelo.

```
from sklearn.model_selection import train_test_split
# X será nuestro conjunto de atributos e y será nuestra variable
objetivo

X_train_pre, X_test_pre, y_train, y_test = train_test_split(sub,
df['price'], test_size =.3, random_state = 63)

# Ajustamos el estandarizador sobre el conjunto de entrenamiento (para
que aprenda la media y desv. est.)
scaler = StandardScaler().fit(X_train_pre)

# Con el estandarizador ajustado sobre entrenamiento, transformamos el
conjunto de entrenamiento con esta estandarizacion
X_train = pd.DataFrame(scaler.transform(X_train_pre), columns =
X_train_pre.columns)

# Transformamos el conjunto de pruebas con el estandarizador ajustado
sobre entrenamiento
X_test = pd.DataFrame(scaler.transform(X_test_pre), columns =
X_test_pre.columns)
```

En la lectura de regularización paramétrica aprendimos sobre la importancia de seleccionar hiper parámetros para obtener un modelo con mejor desempeño. Por suerte para nosotros, la librería `pyGAM` implementa un método de búsqueda de hiper parámetros que nos ahorra el trabajo de explorar a mano las combinaciones. El método `gridsearch` recibe tanto un diccionario que se pasa posteriormente como keyword argument, donde la llave es el nombre del parámetro (en este caso, `lam`) y un rango de valores a evaluar (en este caso, `np.logspace(-3, 3, 3)`), como una lista con los valores a evaluar para el hiper parámetros mencionado como argumento.

Importamos la clase `LinearGAM` de `pygam` la cual implementa una función de link lineal sobre la variable dependiente ***y***. Otras opciones incluyen la Binomial, Poisson, Gamma, Gaussiana Inversa y funciones de vínculo. La API ofrece también una clase prototipo `GAM` para desarrollar modelos personalizados.

```
from pygam import LinearGAM

# definimos el rango de hiperparametros a evaluar
lams = np.logspace(-3, 3, 3)

# Necesitamos generar copias de esta lista de valores a evaluar para
cada funcion f(X), que en nuestro caso serán splines
lams = [lams]* len(X_train.columns)

# Definimos el modelo indicandole qué tipo de función debe tratar de
ajustar a cada atributo/variable independiente en la matriz que le
entregaremos: s(0) significa que se debe ajustar un término de tipo
spline a la primera columna entregada

gam = LinearGAM(s(0) + s(1) + s(2)+ s(3), fit_intercept=True)

# Realizamos el proceso de búsqueda por gridsearch invocando al método
del mismo nombre
gam.gridsearch(X_train, y_train, lam = lams)

# Con el método summary obtenemos una tabla con los resultados del
ajuste del modelo para la mejor combinación de hiper parámetros
gam.summary()
```

```
[9]: gam.summary()

LinearGAM
=============================================== ==========================================================
Distribution:                        NormalDist Effective DoF:                                     50.5084
Link Function:                     IdentityLink Log Likelihood:                                -388710.4607
Number of Samples:                        15129 AIC:                                            777523.9383
                                                AICc:                                           777524.297
                                                GCV:                                       57794407073.9769
                                                Scale:                                     57447204678.7826
                                                Pseudo R-Squared:                                 0.5944
=============================================================================================================
Feature Function                  Lambda               Rank         EDoF         P > x        Sig. Code
================================= ==================== ============ ============ ============ ============
s(0)                              [1000.]              20           4.9          1.11e-16     ***
s(1)                              [0.001]              20           17.5         1.11e-16     ***
s(2)                              [0.001]              20           14.5         1.11e-16     ***
s(3)                              [0.001]              20           13.6         1.11e-16     ***
intercept                         0                    1            0.0          1.11e-16     ***
=============================================================================================================
Significance codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

WARNING: Fitting splines and a linear function to a feature introduces a model identifiability problem
         which can cause p-values to appear significant when they are not.

WARNING: p-values calculated in this manner behave correctly for un-penalized models or models with
         known smoothing parameters, but when smoothing parameters have been estimated, the p-values
         are typically lower than they should be, meaning that the tests reject the null too readily.
```

Imagen 1. Resultados del ajuste del modelo.
Fuente: Desafío Latam.

Cuando generamos la búsqueda de hiper parámetros en la grilla el objeto generado guardará las estimaciones del modelo en consideración al mejor valor de lambda.

El método `summary()` nos entrega una descripción detallada del modelo:

- Podemos ver que para todos los atributos el p-value obtenido es considerablemente pequeño por lo que no tendremos que preocuparnos por descartarlos en este caso.
- Aunque no aparece especificado en la tabla para la versión actual de pyGAM, la documentación de la librería menciona que para todos los casos se utilizan 20 splines de orden 3.
- Podemos observar que la penalización sobre la concavidad encontrada para la primera columna (`'bedrooms'`, representada en el término `s(0)`) es considerablemente mayor a la encontrada para las demás columnas, siendo esta $\lambda_{bedrooms} = 1000$.

Para evaluar el desempeño del modelo, podemos implementar las funciones de métricas de `sklearn`. Para este caso importaremos `r2_score`, `mean_squared_error` y `median_absolute_error`.

Posteriormente los compararemos con otras especificaciones del modelo.

```
from sklearn.metrics import r2_score, mean_squared_error,
median_absolute_error

def report_gam_metrics(model, X_test, y_test):
   print('Test R^2: {0}'.format(r2_score(y_test,
model.predict(X_test)).round(3)))
   print('Test RMSE: {0}'.format(np.sqrt(mean_squared_error(y_test,
model.predict(X_test))).round(3)))
   print('Test Median Absolute Error:
{0}'.format(median_absolute_error(y_test,
model.predict(X_test)).round(3)))


report_gam_metrics(gam, X_test, y_test)
```

```
Test R^2: 0.504
Test RMSE: 59381987543.50868
Test MAE: 120633.493
```

Una pregunta pertinente ahora es si nuestro modelo es razonablemente bueno o no, podemos fijarnos en el MAE calculado para el conjunto de pruebas, los resultados de evaluar el modelo sobre este conjunto muestran que, la mediana de todos los errores (en la diferencia absoluta) en los que incurrió el modelo al hacer las predicciones de los elementos en el conjunto de pruebas es de 120633.393, dicho de otra forma, ese es el error mediano del modelo al momento de predecir el precio de un inmueble. Este valor puede ser muy malo (nuestro modelo puede estar errando demasiado) o muy bueno, dependiendo de la escala y distribución de la variable dependiente, veamos algunos estadísticos de esta variable para contrastar:

```
round(df.price.describe(), 2)
```

```
count      21613.00
mean      540088.14
std       367127.20
min        75000.00
25%       321950.00
50%       450000.00
75%       645000.00
max      7700000.00
Name: price, dtype: float64
```

Imagen 2. Estadísticos.
Fuente: Desafío Latam.

Con una media de 540088.14 y una desviación estándar de 367127 al parecer nuestro modelo predice dentro de rangos razonables, por ejemplo, en el caso de que una vivienda tenga en efecto un valor de precio igual al de la media observada, nuestro modelo podría predecir como valor para ese inmueble 540088.14 - 120633.493 = 419454.647 o 540088.14 + 120633.493 = 660721.633 como valor para esa vivienda, es una diferencia que puede llegar a ser bastante grande dependiendo de los requerimientos que tengamos sobre el precio.

Dependencia Parcial

Hasta el momento sabemos que los modelos aditivos generalizados permiten incorporar formas funcionales flexibles para generar un buen ajuste a los datos. Lamentablemente, las funciones de identidad elicitadas en el modelo no se pueden interpretar de manera directa. Para ello, implementaremos **métodos de dependencia parcial**, que buscan identificar la naturaleza de la dependencia de una función de identidad arbitraria $f(x)$ en los valores conjuntos.

Desde el punto de vista matemático, las funciones de identidad son derivadas parciales linealmente independientes entre los atributos predictores. Podemos estudiar el comportamiento y respuesta de la predicción con respecto a cada atributo por separado si mantenemos los demás atributos fijos como su media. `pyGAM` nos permite hacer esto por medio del método `partial_dependence()` que presenta la evaluación de la función de suavización para cada uno de los atributos. Mediante esta forma generaremos una interpretación visual del modelo.

Lo que evaluamos mediante la fórmula detallada abajo, es la esperanza matemática de la función de suavización para un atributo específico, manteniendo todo lo demás constante (Esta es la condición conocida como **ceteris paribus** en la econometría). Por lo general, cuando implementemos métodos de dependencia parcial, graficamos la esperanza de los valores predichos a lo largo de un atributo específico.

$$\widehat{f}_{x_0}(x_s) = E_{x_c}\left[\widehat{f}(x_s, x_c)\right]$$

```
fig, axs = plt.subplots(1,4, figsize = (20,6));

titles = X_train.columns

for i, ax in enumerate(axs.flatten()):
    XX = gam.generate_X_grid(term=i)
    ax.plot(XX[:, i], gam.partial_dependence(term=i, X=XX))
    ax.plot(XX[:, i], gam.partial_dependence(term=i, X=XX,
width=.95)[1], c='r', ls='--')
    ax.scatter(X_train[titles[i]],
               [0] * len(X_train[titles[i]]),
               marker = '|', alpha = .5)
    ax.set_title(titles[i]);
```

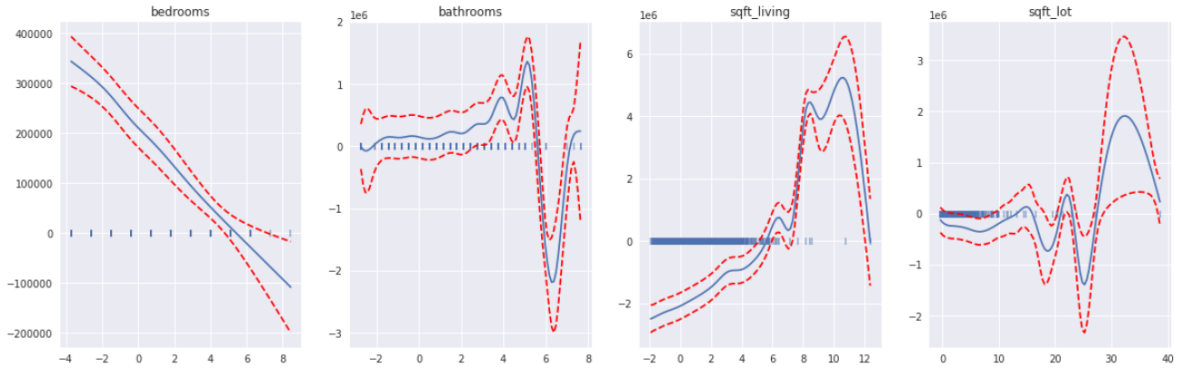

Imagen 3. Gráficos de la esperanza.
Fuente: Desafío Latam.

Hay varias curvas cuyo comportamiento cambia bastante a lo largo de sus dominios, si nos fijamos en las marcas de datos que hay en el eje X veremos que la curva parece tener un ajuste más errático donde hay menor cantidad de puntos, es razonable pensar que en un espacio con poca cantidad de información las splines tendrán una mayor varianza.

El parámetro de penalización (λ) también nos permite modificar qué tanto de la tendencia general y de la tendencia particular de la data se ve reflejada en las curvas. Veamos cómo cambian las curvas de dependencia al modificar el valor de λ.

```
new_gam = LinearGAM(lam = 1e6).fit(X_train, y_train)
report_gam_metrics(new_gam, X_test, y_test)
```

```
Test R^: 0.4820
Test RMSE: 62064324749.4708
Test MAE accuracy: 125551.6367
```

```
fig, axs = plt.subplots(1,4, figsize = (20,6));

titles = X_train.columns

for i, ax in enumerate(axs.flatten()):
    XX = new_gam.generate_X_grid(term=i)
    ax.plot(XX[:, i], new_gam.partial_dependence(term=i, X=XX))
    ax.plot(XX[:, i], new_gam.partial_dependence(term=i, X=XX,
width=.95)[1], c='r', ls='--')
    ax.scatter(X_train[titles[i]],
               [0] * len(X_train[titles[i]]),
               marker = '|', alpha = .5)
    ax.set_title(titles[i]);
```

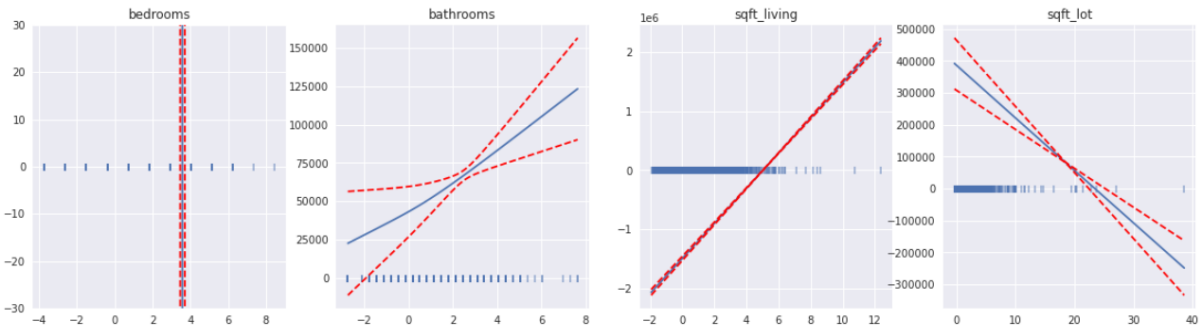

Imagen 4. Curvas de dependencia.
Fuente: Desafío Latam.

Las rectas reflejan muy a groso modo la tendencia general de la curva de dependencia en algunos casos, en otros parece no tener mucha relación probablemente debido a que inicialmente ya habíamos observado que la estimación de la curva de dependencia era bastante mala en zonas con poca cantidad de datos, podemos observar también que usar una penalización de `1e6` es excesivo para la variable `'bedrooms'` resultando en una recta casi vertical que no refleja en absoluto el comportamiento observado anteriormente.
Por lo general, elegiremos un factor de penalización que permita "amortiguar" los comportamientos oscilatorios extremos y repetitivos, pero evitando las rectas, que sabemos que son demasiado ingenuas para modelar casi cualquier fenómeno de la vida real.

Al comparar las métricas de desempeño de los modelos, encontramos evidencia en contra de generar un modelo que penaliza de manera tan fuerte los coeficientes.

Finalmente, con respecto a aquellas curvas en las que vemos problemas en los ajustes aún luego de hacer una búsqueda por grilla, podríamos intentar modelar la dependencia con otro término que no sea una spline y que implementa la librería o, si nos interesa poder evaluar visualmente la interacción mediante los gráficos de dependencia parcial, ajustar manualmente para ese término un factor que entregue un ajuste lo suficientemente bueno como para presentar un margen de error razonable alrededor de la spline junto con una forma funcional interpretable.

Usos

Hastie et al. (2009) sugieren que los modelos aditivos generalizados se pueden implementar para la generación de pronósticos en series de tiempo. Dado que el algoritmo Backfitting permite la reducción del error en las funciones de suavización aditivas. El equipo de Data Science de Facebook creó prophet, una librería para la automatización de pronósticos en series de tiempo mediante GAM. Pueden encontrar más información en el link.

Referencias

- Una introducción exhaustiva respecto a los modelos generalizados aditivos se encuentra en Wood, S. 2017. Generalized Additive Models: An Introduction with R. Texts in Statistical Science. Boca Raton, FL: CRC Press. El capítulo 3: Introducing GAMs presenta los principales elementos analíticos y aspectos formales de los GAM.

- Para profundizar sobre las no linealidades en puntos fijos en la combinación lineal de parámetros se recomienda leer Marsh, L; Cormier, D. 2001. Spline Regression Models. Quantitative applications in the Social Sciences.

- Los vínculos existentes entre el aprendizaje automatizado y los modelos nolineales en X se detallan en Hastie, T; Tibshirani, T; Friedman, J. 2008. The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Springer Series in Statistics. Springer. Ch5: Basis Expansions and Regularization.