{desafío}
latam_

Modelos aditivos generalizados _

Sesión Presencial 2

Itinerario

Activación de conceptos

Desarrollo Desafío

Panel de discusión

{desafío}
latam_

Activación de conceptos

Fenómenos naturales → tienden a no ser lineales

- **En Física, la mayoría de las interacciones vienen dadas por leyes de potencia**
- **El comportamiento del mercado bursátil es extremadamente volátil**

¿Qué es un hiperparámetro?

- Es un parámetro de la distribución a priori de nuestro modelo.
- Es un parámetro que se infiere indirectamente en el modelo.
- Es un parámetro que se declara por el investigador y que afecta el desempeño del modelo.

¿Qué problema busca resolver la regularización paramétrica?

- El trueque entre sesgo y varianza existente en un modelo.
- Extraer un mínimo de parámetros con significancia estadística.
- Maximizar la varianza explicada de un modelo.

¿Cómo se implementa la regularización?

- Ponderando un parámetro $\hat{\beta}$ con su p-value asociado.
- Ponderando $\hat{\beta}$ por el error cuadrático residual.
- Ponderando $\hat{\beta}$ por una norma reflejada en λ.

¿Cuál es la diferencia entre normal ℓ_1 y ℓ_2?

- El rango de valores que toma λ.
- La regularización ℓ_2 entrega soluciones únicas, mientras que ℓ_1 no.
- La regularización ℓ_1 genera ángulos rectos en la distancia de dos vectores, mientras que ℓ_2 sintetiza mediante $\sqrt{x^2 + y^2}$.

Alternativas al problema de la nolinealidad

Modelo	Debilidad
Linear	Forma funcional $y = \mathbf{X}\vec{\beta}$ presenta alto sesgo.
Polinomiales	Reduce capacidad de generalización del modelo. Redefinición constante del modelo.
GLM	Permite flexibilizar el proceso de generación de datos. También hace uso de una forma funcional lineal.

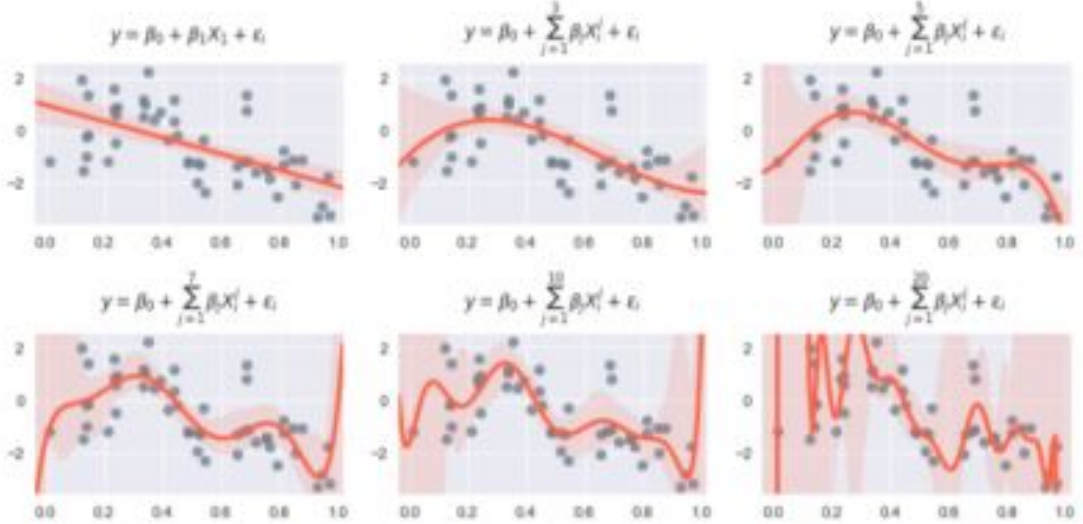
Solución que propone GAM

- Utilizar una superposición de curvas para replicar la tendencia principal del fenómeno, a este tipo de métodos se les conoce como modelos aditivos.

- Las curvas se conocen como spline, estas curvas pueden ser diferentes órdenes.

Uso y abuso de polinomiales

```
In [2]: plt.figure(figsize=(10,5)); polynomial_degrees()
```



Modelos aditivos generalizados

Setup básico:

La forma general de los GAMs es:

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

- Podemos aplicar nolinealidades a la interacción de atributos con $f_i(\cdot)$.
- En un modelo con PGD normal, $f_i(\cdot)$ son simplemente la función de vínculo por identidad.

Entrenamiento de GAMs

Las GAMs clásicas minimizan la siguiente función objetivo:

$$\underset{\hat{f}_i}{\text{argmin}} \sum_{i=1}^{n}(y_i - \sum_{j=1}^{p} f_j(x_{i,j}))^2 + \lambda \sum_{j}^{p} \int f_j''(t)^2 dt$$

- La forma de la función objetivo es muy similar a la vista en regularizadores, de hecho, el término de la derecha actúa como un regularizador sobre la curvatura de la spline, evitando splines que sean muy excéntricas.
- Lo interesante del término de regularización es que también viene acompañado de un parámetro de regularización λ que nos dice como se penalizará la curvatura de los aplines agregadas.

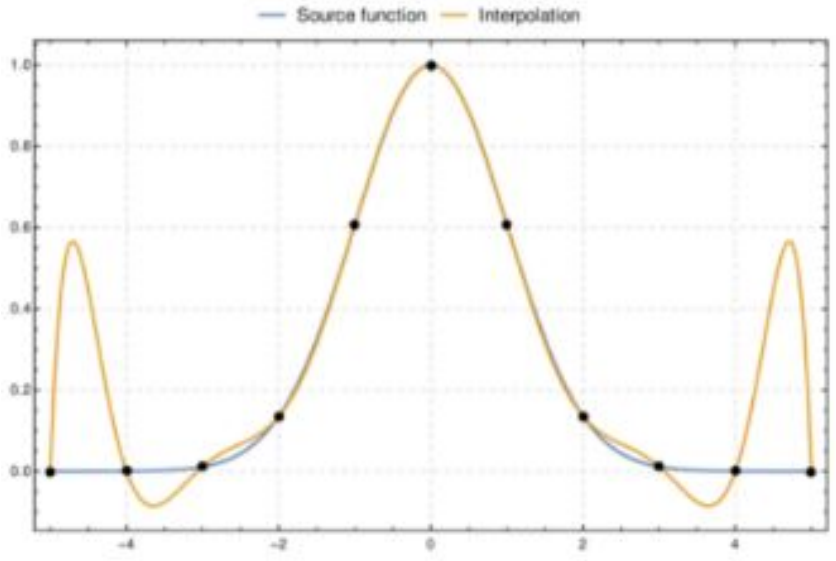
Polinomios y regularización

Fenómeno de Runge: Cuando ocupamos polinomios con un orden superior al cúbico, los extremos de la curva se escapan de la tendencia.

El componente regularizador

$$(\lambda \sum_j^p \int f_j''(t)^2 dt)$$

de los GAM impide que se escapen las curvas en los extremos de la curva.



Implementación

- Librerías clásicas: `numpy, pandas, matplotlib`
- Preprocesadores de sklearn: `StandardScaler, train_test_split`.
- Modelo GAM: `pygam`

```
conda install -c conda-forge pygam
```

pygam funciona de manera similar a sklearn y statmodels:

- Importamos una clase de la librería: `from pygam import LinearGAM`
- Instanciamos la clase en un objeto y asignamos hiperparámetros.
- Ejecutamos .fit

In [3]:

```
# Imports omitidos
from pygam import LogisticGAM #implementamos una variante logística
from sklearn.datasets import load_breast_cancer
# Breast cancer data
data = load_breast_cancer()
df = pd.DataFrame(data.data, columns=data.feature_names).dropna()
df = pd.DataFrame(data.data, columns=data.feature_names)[['mean radius', 'mean texture',
'mean perimeter', 'mean area','mean smoothness', 'mean compactness']]
target_df = pd.Series(data.target).dropna()
```

Vector objetivo → Tiene o no tiene cáncer mamario.

In [4]:

```
target_df.value_counts()
```

Out[4]:

```
1    357
0    212
dtype: int64
```

Preprocesamos

In [5]:

```
scaler = StandardScaler()
df_scaled = pd.DataFrame(scaler.fit_transform(df), columns = df.columns)
X = df_scaled.iloc[:, 1:] # Tomamos todas las columnas menos la primera (price)
N = X.shape[0] # guardamos el número de filas (datos de entrenamiento)
X.insert(X.shape[1], 'intercept', np.ones(N)) #Se crea una columna nueva dentro de las va
riables predictoras llamada 'intecept'
```

Generamos muestras y entrenamos

In [6]:
```
Xtrain, Xtest, ytrain, ytest = train_test_split(df, target_df, test_size = 0.3, random_st
ate = 101)
logistic_gam = LogisticGAM().fit(Xtrain, ytrain)
```

pygam implementa un hiperparámetro de regularización por defecto $\lambda = .6$

```
In [7]: print('Lambda: {0}'.format(logistic_gam.lam))

Lambda: 0.6
```

Exactitud de las predicciones de un 95%

```
In [8]: print('Clasification accuracy: {0}'.format(logistic_gam.accuracy(Xtest,ytest)))

Clasification accuracy: 0.9532163742690059
```

Resumen del modelo

- Al implementar `logistic_gam.summary()` obtendremos una tabla sobre el desempeño del modelo en términos generales, así como qué atributos fueron significativos (al nivel nominal del 95%).

- Para inspeccionar el efecto de cada atributo, es mejor implementar gráficos de dependencia parcial.

Selección de hiperparámetros

- GridSearch → Selección de mejor hiperparámetro en consideración al desempeño.
- Por ahora trabajamos con sólo un hiperparámetro.

- Incorporamos el diccionario que contiene los valores como **kwargs.

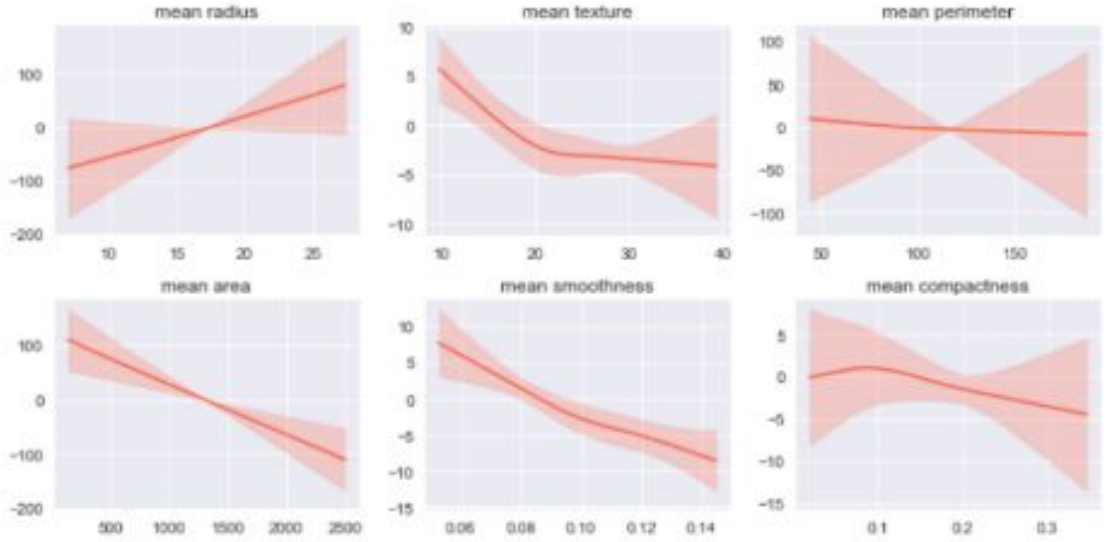
```
In [9]: search_params = {'lam': np.logspace(0, 3, 11)}
        logistic_gam = LogisticGAM().gridsearch(Xtrain, ytrain, **search_params)

        100% (11 of 11) |########################| Elapsed Time: 0:00:00 Time:
          0:00:00
```

- Métricas

```
In [10]: print("Mejor valor de lambda: {}".format(logistic_gam.lam))

         Mejor valor de lambda: 7.943282347242813

In [11]: print('Clasification accuracy: {0}'.format(logistic_gam.accuracy(Xtest,ytest)))

         Clasification accuracy: 0.9181286549707602
```

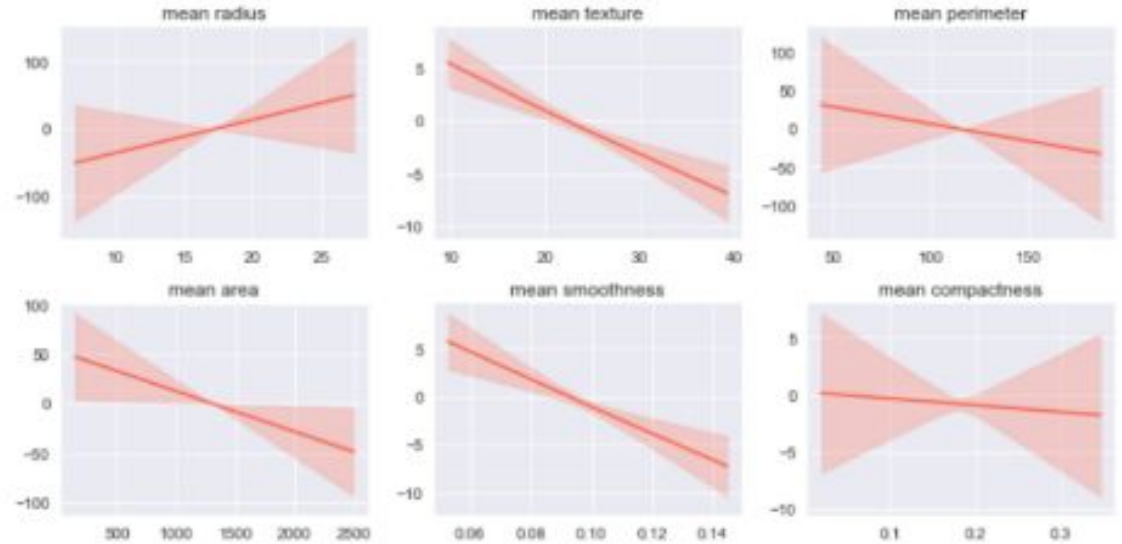
Dependencia parcial con $\lambda = 0.6$

```
In [12]: plt.figure(figsize=(10,5));logistic_gam_default()
```



Dependencia parcial con $\lambda = 1000$

```
In [13]: plt.figure(figsize=(10,5));logistic_gam_1000()
```



/* Desafío */

Panel de discusión

{desafío} latam_

Academia de talentos digitales

www.desafiolatam.com