{desafío}
latam_

Regularización Paramétrica _

Sesión Presencial 1

Itinerario

Activación de conceptos

Desarrollo Desafío

Panel de discusión

{desafío}
latam_

Activación de conceptos

- **Función objetivo**

Función a optimizar para que aprenda los parámetros correspondientes.

- **Entrenamiento**

Proceso mediante el cual la máquina optimiza su función objetivo.

- **Espacio de atributos**

Espacio donde se encuentran todos los atributos posibles para nuestra máquina.

- **Espacio de parámetros**

Dónde se encuentran los posibles parámetros para la máquina que estamos implementando.

- **Atributos y variable objetivo**

Una columna en nuestro dataset.

- **Overfitting (sobre-ajuste)**

Fenómeno en el cual nuestro modelo aprendió demasiado bien el conjunto de datos con el que lo entrenamos.

- **Underfitting (sub-ajuste)**

Fenómeno en el cual nuestro modelo aprendió muy poco sobre el fenómeno subyacente en los datos.

- **Regularización**

Es una técnica ad-hoc para cada máquina/función objetivo, que nos permitirá dos cosas:

➔ Evitar overfitting
➔ Incluir características deseables en la máquina

Como la regularización es un proceso que depende de la máquina con la que se esté trabajando, veremos estas técnicas caso a caso a medida que se vayan presentando los modelos.

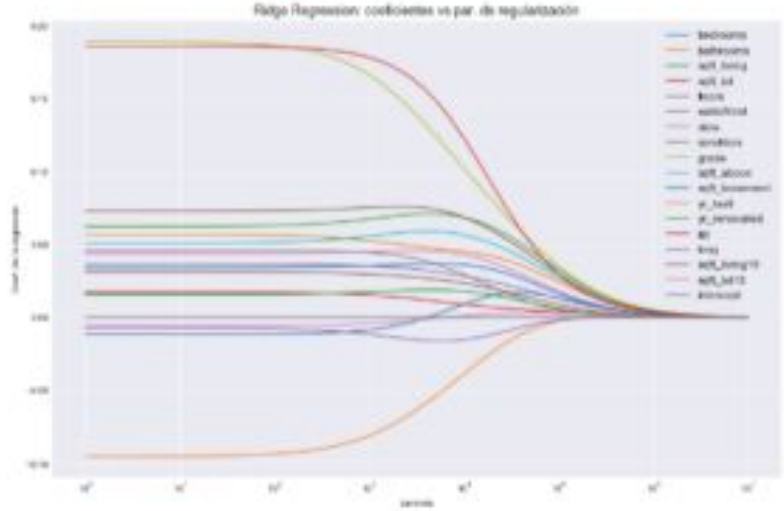
Objetivo de la Regularización

- Evitar el overfitting e incluir características deseables en la máquina.
- Mecanismo: Penalizar la función objetivo.
- Forma canónica de la regularización:

$$\Theta \Rightarrow \underset{\text{Parametros}}{\text{Optimizacion}} \quad \text{Funcion objetivo} \pm \text{Regularizador}$$

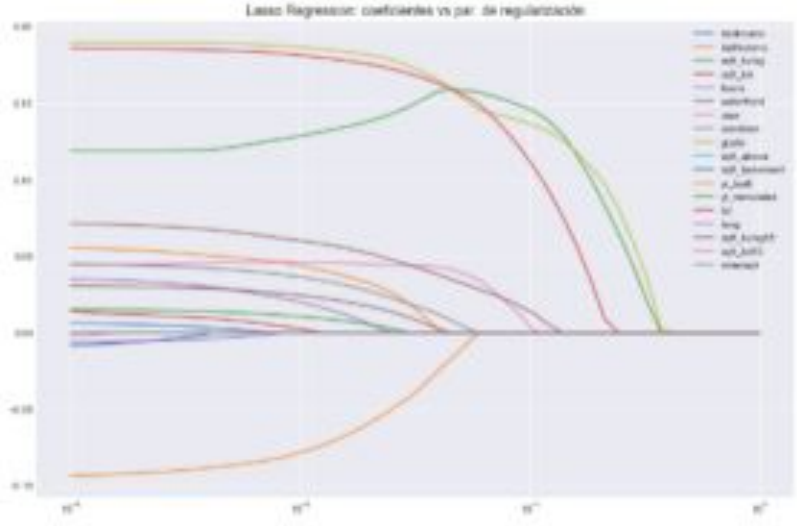
Ridge

$$\beta_{\text{Ridge}} = \underset{\beta}{\text{argmin}} \sum_{i}^{n} (y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$



Lasso

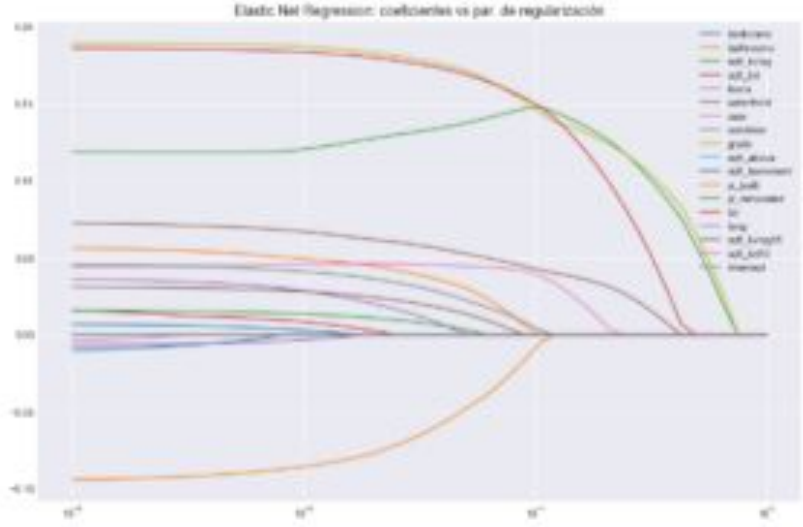
$$\beta_{\text{Lasso}} = \underset{\beta}{\text{argmin}} \sum_{i}^{n} (y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$



{desafío}
latam_

ElasticNet

$$\beta_{\text{ElasticNet}} = \underset{\beta}{\text{argmin}} \sum_{i}^{n} (y_i - \hat{y}_i)^2 + \lambda_1 \sum_{j=1}^{p} |\beta_j| + \lambda_2 \sum_{j=1}^{p} \beta_j^2$$



Ejemplo de Optimización y hiperparámetros

Generaremos un dataset artificial para regresión:

```
In [2]: from sklearn.datasets.samples_generator import make_regression
        X, y = make_regression(n_samples=1400, n_features=7, random_state=21)
        df = pd.DataFrame(X)
```

Probemos el rendimiento de los métodos para el dataset:

```
In [3]: df.head()
```

Out[3]:

	0	1	2	3	4	5	
0	1.671540	2.282226	-0.909524	-0.433356	-0.936663	1.311349	-0.232
1	0.797755	0.920072	-0.489390	-0.727929	0.517007	1.701459	0.6394
2	0.470842	-0.879841	1.073794	-1.369692	-1.247534	-0.819175	-1.895
3	-0.239566	0.096981	0.191484	0.108997	-1.728113	-0.460841	0.1408
4	0.152605	0.969800	0.122987	0.558151	0.960098	1.727859	0.1964

Preparemos nuestro ambiente de trabajo

In [4]:

```
from sklearn.linear_model import Lasso, Ridge, ElasticNet; from sklearn.preprocessing imp
ort StandardScaler; from sklearn.model_selection import train_test_split

# Preprocesamiento
scaler = StandardScaler();df_scaled = scaler.fit_transform(df)
X_train, X_test, y_train, y_test = train_test_split(df_scaled, y, test_size = 0.3, random
_state = 21)

# Lasso
lasso_model = Lasso().fit(X_train, y_train)
# Ridge
ridge_model = Ridge().fit(X_train, y_train)
#Elastic Net
enet_model = ElasticNet().fit(X_train, y_train)
```

Métricas

Los ejemplos de regularización vistos se basan en la regresión lineal → la métrica de optimización es (R)MSE.

```
In [5]: from sklearn.metrics import mean_squared_error
        print('Ridge MSE: {0}'.format(mean_squared_error(ridge_model.predict(X_test), y_test)))
        print('Lasso MSE: {0}'.format(mean_squared_error(lasso_model.predict(X_test), y_test)))
        print('Elastic Net MSE: {0}'.format(mean_squared_error(enet_model.predict(X_test), y_test
        )))

        Ridge MSE: 0.032367749978319066
        Lasso MSE: 6.905771887596452
        Elastic Net MSE: 3569.3027415529227
```

MSE en norma -l_1

Importante: Seleccionamos hiperparámetros en la muestra de entrenamiento.

In [6]:
```
alphas = [0.01, 0.1, 1.0, 10.0, 100.0]
for a in alphas:
    lasso_model = Lasso(alpha=a).fit(X_train, y_train)
    print('Train MSE on Lasso for alpha = {0}: {1}'.format(a, mean_squared_error(lasso_mo
del.predict(X_train), y_train)))
```

```
Train MSE on Lasso for alpha = 0.01: 0.0006754788760091857
Train MSE on Lasso for alpha = 0.1: 0.06750696518946286
Train MSE on Lasso for alpha = 1.0: 6.750288513363294
Train MSE on Lasso for alpha = 10.0: 675.0247724990405
Train MSE on Lasso for alpha = 100.0: 30747.33037964957
```

In [7]:
```
print('-' *60); lasso_model = Lasso(alpha=0.01).fit(X_train, y_train)
print('Test MSE on Lasso for alpha = {0}: {1}'.format(lasso_model.alpha, mean_squared_err
or(lasso_model.predict(X_test), y_test)))
```

```
------------------------------------------------------------
Test MSE on Lasso for alpha = 0.01: 0.0006909428932056883
```

MSE en ElasticNet

In [8]:
```
alphas = [0.01, 0.1, 1.0, 10.0, 100.0]
for a in alphas:
    enet_model = ElasticNet(alpha=a).fit(X_train, y_train)
    print('Train MSE on Enet for alpha = {0}: {1}'.format(a, mean_squared_error(enet_mode
l.predict(X_train), y_train)))
```

```
Train MSE on Enet for alpha = 0.01: 0.7647210013241578
Train MSE on Enet for alpha = 0.1: 70.095198033748
Train MSE on Enet for alpha = 1.0: 3451.0457057216963
Train MSE on Enet for alpha = 10.0: 21848.66950935735
Train MSE on Enet for alpha = 100.0: 30363.99601423732
```

In [9]:
```
print('-' * 60); enet_model = ElasticNet(alpha=0.01).fit(X_train, y_train)
print('Test MSE on Lasso for alpha = {0}: {1}'.format(enet_model.alpha, mean_squared_erro
r(enet_model.predict(X_test), y_test)))
```

```
------------------------------------------------------------
Test MSE on Lasso for alpha = 0.01: 0.792382120149239
```

Comportamiento de la regularización en matrices dispersas

Matriz densa → Matrices donde la cantidad de no ocurrencias es infima.
Matriz dispersa → Matrices donde la mayor cantidad de observaciones son no ocurrentes.

Generaremos dos datasets artificiales para regresión, uno denso y otro disperso (sparse).

Veamos cómo se comportan los modelos con matrices de datos densas vs dispersas

In [13]:
```
from scipy import sparse
from scipy import linalg
import numpy as np
from sklearn.datasets.samples_generator import make_regression
from time import time

X, y = make_regression(n_samples=400000, n_features=500, random_state=21)
X_dense = sparse.coo_matrix(X)

X_sparse = X.copy()
X_sparse[X_sparse < 2.5] = 0.0
X_sparse = sparse.coo_matrix(X_sparse)
X_sparse = X_sparse.tocsc()
```

In [14]:
```
print("Matrix density X_sparse: %s %%" % (X_sparse.nnz / float(X.size) * 100))
```

```
Matrix density X_sparse: 0.6207294999999999 %
```

Ejemplo: Lasso $\lambda = 0.01$ en Dispersas vs Densa

In [15]:

```
sparse_lasso = Lasso(alpha=0.01, fit_intercept=False, max_iter=1000)
dense_lasso = Lasso(alpha=0.01, fit_intercept=False, max_iter=1000)

# sparse
t0 = time(); sparse_lasso.fit(X_sparse, y); print("Sparse Lasso done in %fs" % (time() -
t0))

# dense
t0 = time(); sparse_lasso.fit(X, y); print("Dense Lasso done in %fs" % (time() - t0))
```

```
Sparse Lasso done in 0.079642s
Dense Lasso done in 6.583675s
```

/* Desafío */

Panel de discusión

{desafío} latam_

Academia de talentos digitales

www.desafiolatam.com