Memoria P6. Aprendizaje Automático en la Práctica

En esta práctica se ha probado las herramientas aportadas por la libreria **sklearn** para el desarrollo de sistemas de entrenamiento.

Método de entrenamiento

Este es el método principal utilizado para la mayor parte de pruebas. Recibe los parámetros:

- **X_train**: datos de entrenamiento.
- **y_train**: resultados de entrenamiento.
- **degree**: grado del sistema polinomico.
- **lambda_**: constante de regularizacion.

Cabe destacar que, para **lambda_**=0, **skl.Ridge** aplica igual que **skl.LinearRegression**

```
def train(x_train, y_train, degree=15, lambda_=0):
    poly = skp.PolynomialFeatures(degree=degree, include_bias=False)
    scalar = skp.StandardScaler()
    model = skl.Ridge(alpha=lambda_) # model = skl.LinearRegression()

    x_train = poly.fit_transform(x_train)
    x_train = scalar.fit_transform(x_train)

    model.fit(x_train, y_train)

    return poly, scalar, model
```

Cálculo de error

Esta función se encarga de calcular el error del entrenamiento con unos datos de prueba. Recibe los parámetros:

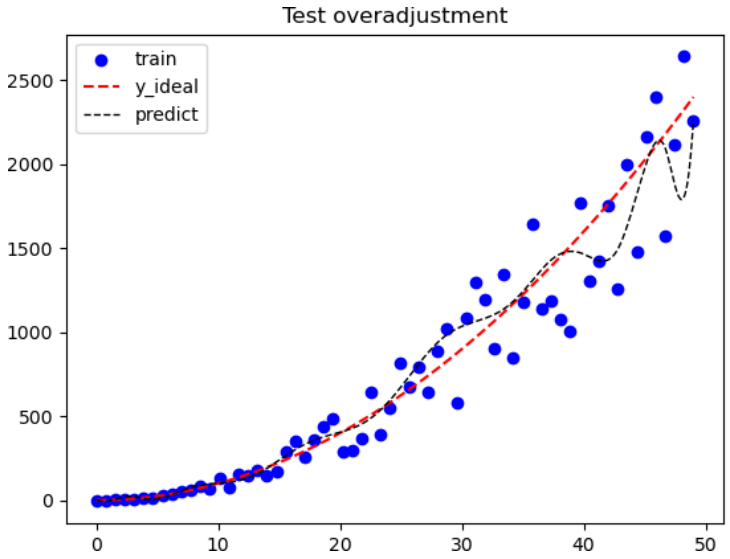
- **poly, scalar, model**: modelo de entrenamiento.
- **x, y:** datos de prueba.

```
def calc_error(poly, scalar, model, x, y):
    x_test = poly.transform(x)
    x_test = scalar.transform(x_test)

    y_predict = model.predict(x_test)

    return np.sum((y_predict - y)**2) / (2 * y.shape[0]), x_test, y_predict[:,
None]
```

Resultado de pruebas

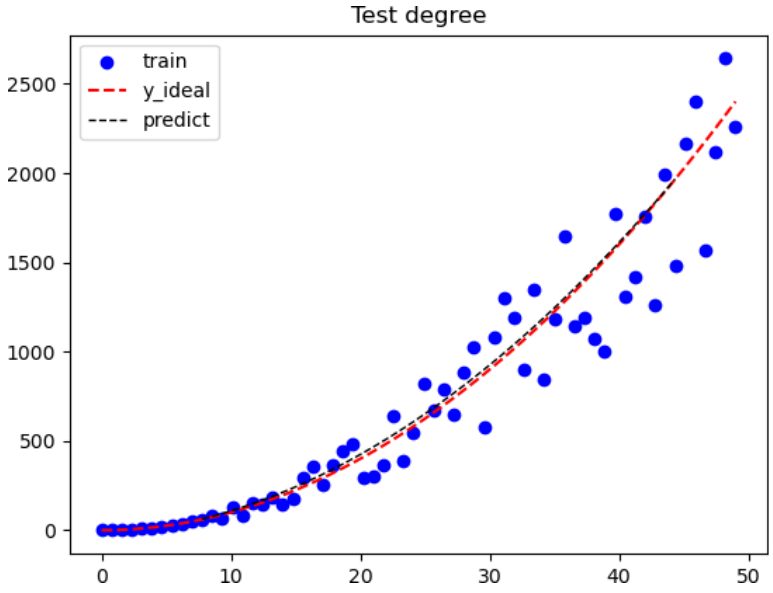
Para cada función presentada a continuación, se obtienen los siguientes resultados:

```
def test_overadjustment(X, y, x_ideal, y_ideal):
    x_train, x_test, y_train, y_test = skms.train_test_split(X, y,
test_size=0.33, random_state=1)

    poly, scalar, model = train(x_train, y_train)

    error_train, _, y_train_predict = calc_error(poly, scalar, model ,
x_train, y_train)
    error_test, _, _ = calc_error(poly, scalar, model, x_test, y_test)
```



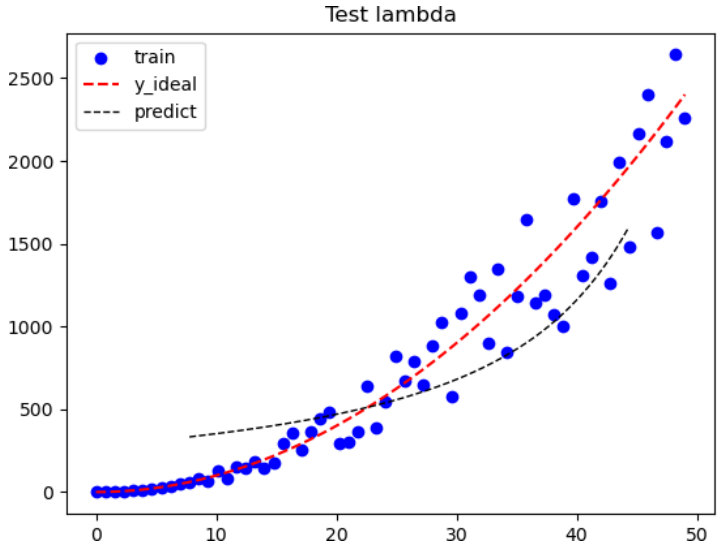

```
--OVERADJUSTMENT TEST--
Error train: 11855.047962965764
Error test: 48579.55496636422
```

```python
def test_degree(X, y, x_ideal, y_ideal):
    x_train, x_, y_train, y_ = skms.train_test_split(X, y, test_size=0.4,
random_state=1)
    x_cv, x_test, y_cv, y_test = skms.train_test_split(x_, y_, test_size=0.5,
random_state=1)

    error_data = np.zeros(10) # degree from 1 to 10 [(0,9) + 1]
    for temp_degree in range(len(error_data)):
        poly, scalar, model = train(x_train, y_train, temp_degree + 1)

        error_data[temp_degree], _, _ = calc_error(poly, scalar, model, x_cv,
y_cv)

    proper_degree = np.argmin(error_data) + 1

    poly, scalar, model = train(x_train, y_train, proper_degree)

    _, _, y_predict = calc_error(poly, scalar, model, x_test, y_test)
```



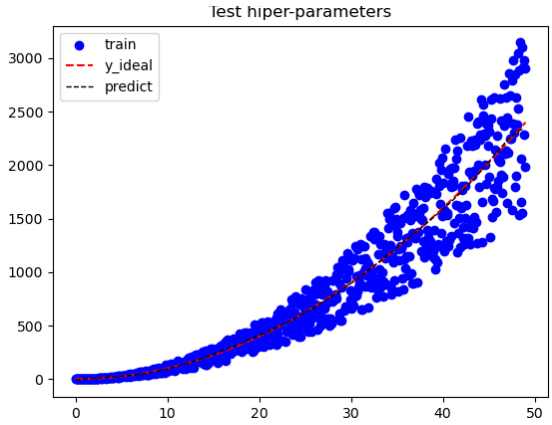

```
--DEGREE TEST--
Proper degree: 2
Error degree: 28693.275740594694
```

```python
def test_lambda(X, y, x_ideal, y_ideal):
    x_train, x_, y_train, y_ = skms.train_test_split(X, y, test_size=0.4,
random_state=1)
    x_cv, x_test, y_cv, y_test = skms.train_test_split(x_, y_, test_size=0.5,
random_state=1)

    degree = 15
    lambdas = np.array([1e-6, 1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 1, 10, 100, 300,
600, 900])
    error_data = np.zeros(len(lambdas))
    for temp_lambda in range(len(lambdas)):
        poly, scalar, model = train(x_train, y_train, degree,
lambdas[temp_lambda])

        error_data[temp_lambda], _, _ = calc_error(poly, scalar, model, x_cv,
y_cv)

    proper_lambda = lambdas[np.argmin(error_data)]

    poly, scalar, model = train(x_train, y_train, degree, proper_lambda)

    error_test, _, y_predict = calc_error(poly, scalar, model, x_test, y_test)
```




```
--LAMBDA TEST--
Proper lambda: 100.0
Error lambda: 35759.737760578726
```

```python
def test_hiper_parameters(X, y, x_ideal, y_ideal):
    x_train, x_, y_train, y_ = skms.train_test_split(X, y, test_size=0.4,
random_state=1)
    x_test, x_cv, y_test, y_cv = skms.train_test_split(x_, y_, test_size=0.5,
random_state=1)

    degrees = 15
    lambdas = np.array([1e-6, 1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 1, 10, 100, 300,
600, 900])
    error_data = np.zeros((len(lambdas), degrees))
    for temp_lambda in range(len(lambdas)):
        for temp_degree in range(degrees):
            poly, scalar, model = train(x_train, y_train, temp_degree + 1,
lambdas[temp_lambda])

            error_data[temp_lambda, temp_degree], _, _ = calc_error(poly,
scalar, model, x_cv, y_cv)

    min = np.argmin(error_data)
    lambda_id = int(min/len(lambdas))
    proper_lambda = lambdas[lambda_id]
    proper_degree = min - (len(lambdas) * lambda_id) + 1

    poly, scalar, model = train(x_train, y_train, proper_degree,
proper_lambda)

    error, _, y_predict = calc_error(poly, scalar, model, x_test, y_test)
```




```
--HIPER PARAMETERS TEST--
Proper degree: 12
Proper lambda: 1e-06
Error test: 28652.337001767624
```

```
def test_error():
    num_data = np.linspace(50, 1000, 20)
    error_data_train = np.zeros(len(num_data))
    error_data_cv = np.zeros(len(num_data))

    for value in range(len(num_data)):
        X, y, x_ideal, y_ideal = ut.gen_data(int(num_data[value]))
        X = X[:, None]

        x_train, x_test, y_train, y_test = skms.train_test_split(X, y,
test_size=0.4, random_state=1)
        x_test, x_cv, y_test, y_cv = skms.train_test_split(x_test, y_test,
test_size=0.5, random_state=1)

        degree = 16
        poly, scalar, model = train(x_train, y_train, degree)

        error_data_train[value], _, _ = calc_error(poly, scalar, model,
x_train, y_train)
        error_data_cv[value], _, _ = calc_error(poly, scalar, model, x_cv,
y_cv)

    ut.plot_error(num_data[1:], error_data_train[1:], error_data_cv[1:],
'./results/test_error.png')
```