Practica 6: Aprendizaje automático en la práctica

Alberto Muñoz Fernández

Óscar García Castro

```
from pickle import NONE
import numpy as np
import matplotlib.pyplot as plt
import sklearn.model_selection as sms
import sklearn.preprocessing as sp
import sklearn.linear_model as slm

def draw_data(x, y, x_i, y_i, x_tr, y_tr):      #Dibujado para apartados
1-4
    plt.figure()
    plt.plot(x, y, 'bo', label = "train")
    plt.plot(x_i, y_i, c = 'red', label = "y_ideal", linestyle='dashed')
    plt.plot(x_tr, y_tr, c='green', label='predict', linestyle='dashed',
linewidth=2, markersize=12)

    plt.legend()
    plt.show()

def draw_data_2(x, y, x_i, y_i):                #Dibujado apartado 5
    plt.figure()
    plt.plot(x_i, y_i, c = 'orange', label = "validation", linewidth=2,
markersize=12)
    plt.plot(x, y, c='blue', label='train', linewidth=2, markersize=12)
    plt.ylim(0, 175000)
    plt.legend()
    plt.show()

def gen_data(m, seed=1, scale=0.7):
    """ generate a data set based on a x^2 with added noise """
    c = 0
    x_train = np.linspace(0, 49, m)
    np.random.seed(seed)
    y_ideal = x_train**2 + c
    y_train = y_ideal + scale * y_ideal*(np.random.sample((m,))-0.5)
    x_ideal = x_train
    return x_train, y_train, x_ideal, y_ideal

def compute_cost(y, y_predict):
    sum = 0
    m = len(y)
```

```
    sum = np.sum((y_predict - y) ** 2)

    return sum/(2*m)

def train(degr, x_train, y_train):
    pol = sp.PolynomialFeatures(degree= degr, include_bias=False)
    x_train = pol.fit_transform(x_train)

    scal = sp.StandardScaler()
    x_train = scal.fit_transform(x_train)

    lin = slm.LinearRegression()
    lin.fit(x_train, y_train)

    return pol, scal, lin, x_train

def trainReg(degr, x_train, y_train, lambda_):
    pol = sp.PolynomialFeatures(degree= degr, include_bias=False)
    x_train = pol.fit_transform(x_train)

    scal = sp.StandardScaler()
    x_train = scal.fit_transform(x_train)

    lin = slm.Ridge(lambda_)
    lin.fit(x_train, y_train)

    return pol, scal, lin, x_train

def test(x_test, y_test, x_train, y_train, pol, scal, lin):
    x_test = pol.transform(x_test)
    x_test = scal.transform(x_test)

    y_testpre = lin.predict(x_test)
    test_cst = compute_cost(y_test, y_testpre)
    y_trainpre = lin.predict(x_train)
    train_cst = compute_cost(y_train, y_trainpre)

    return test_cst, train_cst, y_testpre

def sobreAjuste(x, y):
    x = x[:, None]
    x_train, x_test, y_train, y_test = sms.train_test_split(x, y,
test_size = 0.33, random_state = 1)

    pol, scal, lin, x_train = train(15, x_train, y_train)

    x_draw = np.linspace(0, 49, 1000)
    x_draw = x_draw[:, None]
```

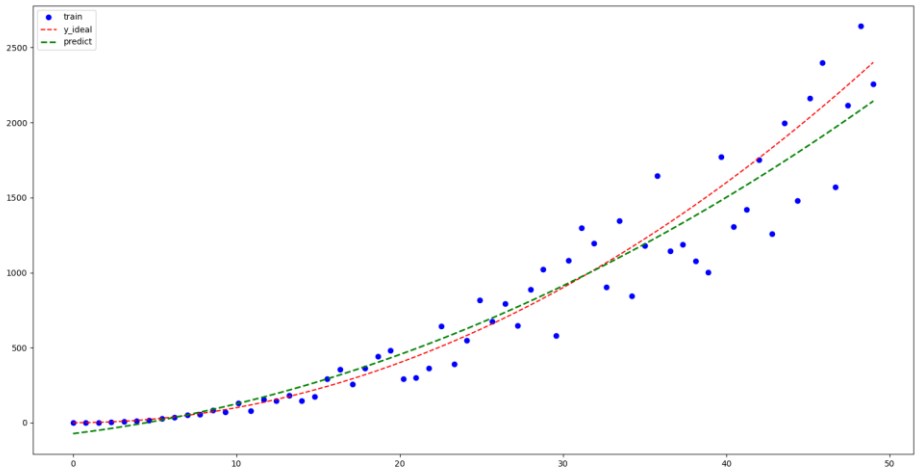
```
    x_d = pol.transform(x_draw)
    x_d = scal.transform(x_d)
    y_testpre1 = lin.predict(x_d)
    testc, trc, ytestp = test(x_test, y_test, x_train, y_train, pol,
scal, lin)
    print("Test cost: " + str(testc))
    print("Train cost: " + str(trc))

    return x_draw, y_testpre1

def eleccGrado(x, y):
    x = x[:, None]
    x_train, x_test, y_train, y_test = sms.train_test_split(x, y,
test_size = 0.2, random_state = 1)
    x_train, x_val, y_train, y_val = sms.train_test_split(x_train,
y_train, test_size = 0.25, random_state = 1)

    minCos = 0
    degree = 0

    for i in range(10):
        pol, scal, lin, x_train_t = train(i + 1, x_train, y_train)
        test_cst, train_cst, y_tspr = test(x_val, y_val, x_train_t,
y_train, pol, scal, lin)
        if(minCos == 0 or test_cst < minCos):
            minCos = test_cst
            degree = i+1

    pol, scal, lin, x_train = train(degree, x_train, y_train)

    x_draw = np.linspace(0, 49, 1000)
    x_draw = x_draw[:, None]
    x_d = pol.transform(x_draw)
    x_d = scal.transform(x_d)
    y_testpre1 = lin.predict(x_d)

    print("Mejor grado: ", degree)
    print("Coste test: ", minCos)

    return x_draw, y_testpre1

def searchLambda(degree, x_train, y_train, x_val, y_val):
    lambda_ = [1e-6, 1e-5, 1e-4, 1e-3, 1e-2, 1, 10, 100, 300, 600, 900]
    alpha = 0
    minCos = -1
    for i in range(11):
        pol, scal, lin, x_train_t = trainReg(degree, x_train, y_train,
lambda_[i])
```

```
        test_cst, train_cst, y_tspr = test(x_val, y_val, x_train_t,
y_train, pol, scal, lin)
        if(minCos == -1 or test_cst < minCos):
            minCos = test_cst
            alpha = lambda_[i]
    return alpha, minCos

def eleccLambda(x, y):
    x = x[:, None]
    x_train, x_test, y_train, y_test = sms.train_test_split(x, y,
test_size = 0.2, random_state = 1)
    x_train, x_val, y_train, y_val = sms.train_test_split(x_train,
y_train, test_size = 0.25, random_state = 1)

    alpha, minCos = searchLambda(15, x_train, y_train, x_val, y_val)

    pol, scal, lin, x_train = trainReg(15, x_train, y_train, alpha)

    x_draw = np.linspace(0, 49, 1000)
    x_draw = x_draw[:, None]
    x_d = pol.transform(x_draw)
    x_d = scal.transform(x_d)
    y_testpre1 = lin.predict(x_d)

    print("Mejor lambda: ", alpha)

    return x_draw, y_testpre1

def eleccHiperParams(x, y):
    x = x[:, None]
    x_train, x_test, y_train, y_test = sms.train_test_split(x, y,
test_size = 0.2, random_state = 1)
    x_train, x_val, y_train, y_val = sms.train_test_split(x_train,
y_train, test_size = 0.25, random_state = 1)

    minCos = -1
    defAlpha = 0
    defDegree = 0
    for i in range(14):     #For para los grados
        alpha, cost = searchLambda(i + 1, x_train, y_train, x_val, y_val)
        if(minCos == -1 or minCos > cost):
            minCos = cost
            defAlpha = alpha
            defDegree = i + 1

    pol, scal, lin, x_train = trainReg(defDegree, x_train, y_train,
defAlpha)
    test_cst, train_cst, y_tspr = test(x_test, y_test, x_train, y_train,
pol, scal, lin)
```

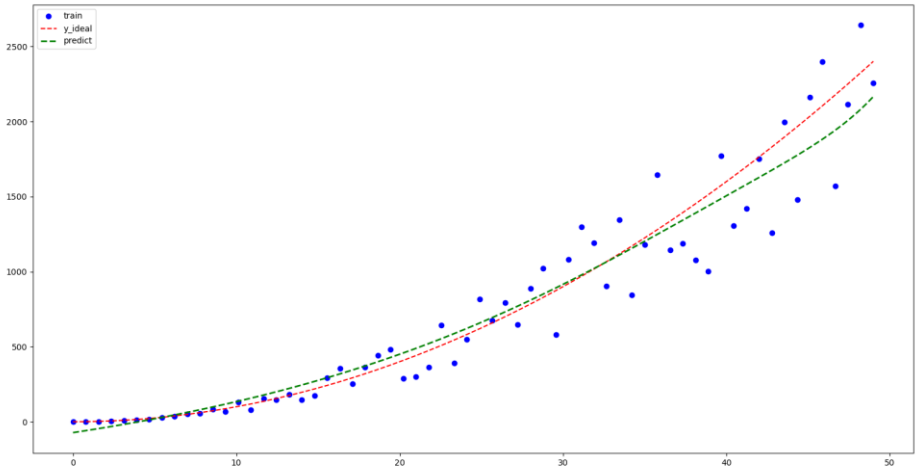
```
    print("Mejor lambda: ", defAlpha, "Mejor grado: ", defDegree)
    print("Coste test: " + str(test_cst))

    x_draw = np.linspace(0, 49, 1000)
    x_draw = x_draw[:, None]
    x_d = pol.transform(x_draw)
    x_d = scal.transform(x_d)
    y_testpre1 = lin.predict(x_d)

    return x_draw, y_testpre1

def learningCurve():
    J_costTrain = []
    J_costVal = []
    params = []

    for i in range(20):
        params += [50 *(i+1)]
        x, y, x_i, y_i = gen_data(params[i])
        x = x[:, None]
        x_train, x_test, y_train, y_test = sms.train_test_split(x, y,
test_size = 0.2, random_state = 1)
        x_train, x_val, y_train, y_val = sms.train_test_split(x_train,
y_train, test_size = 0.25, random_state = 1)
        pol, scal, lin, x_train_t = train(16, x_train, y_train)
        test_cst, train_cst, y_tspr = test(x_val, y_val, x_train_t,
y_train, pol, scal, lin)
        J_costTrain += [train_cst]
        J_costVal += [test_cst]


    draw_data_2(params, J_costTrain, params, J_costVal)

def main():
    x, y, x_i, y_i = gen_data(64)
    #Apartado 1: Sobreajuste ejemplos entrenamiento
    x_sorted, y_sorted = sobreAjuste(x, y)

    #Apartado 2: Elección grado polinomio
    x_sorted, y_sorted = eleccGrado(x, y)

    #Apartado 3: Elección lambda
    x_sorted, y_sorted = eleccLambda(x, y)

    #Apartado 4: Eleccion hiper-parámetros
    x_sorted, y_sorted = eleccHiperParams(x, y)

    #Dibujado de los resultados
```

```
    draw_data(x, y, x_i, y_i, x_sorted, y_sorted)

    #Apartado 5: Curvas aprendizaje
    learningCurve()

main()
```

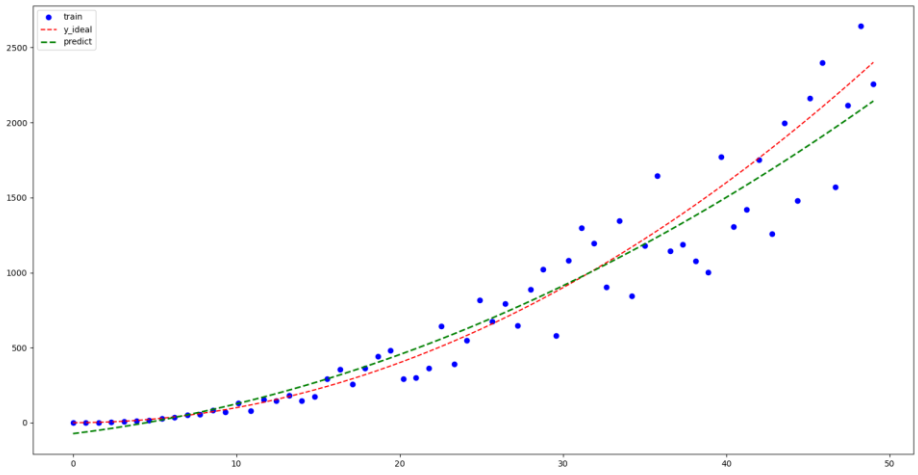
Apartado 1



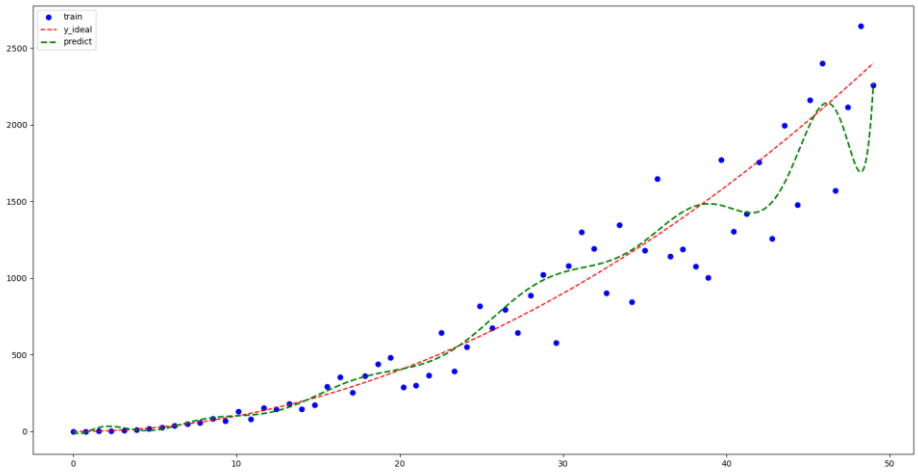
Apartado 2



Apartado 3



Apartado 4



Apartado 5