SU-2019-LAB1-0036501052

October 25, 2019

Sveuilite u Zagrebu
Fakultet elektrotehnike i raunarstva

0.1 Strojno uenje 2019/2020

http://www.fer.unizg.hr/predmet/su

0.1.1 Laboratorijska vjeba 1: Regresija

Verzija: 1.2
Zadnji put aurirano: 27. rujna 2019.

(c) 2015-2019 Jan najder, Domagoj Alagi

Objavljeno: **30. rujna 2019.**
Rok za predaju: **21. listopada 2019. u 07:00h**

0.1.2 Upute

Prva laboratorijska vjeba sastoji se od deset zadataka. U nastavku slijedite upute navedene u elijama s tekstom. Rjeavanje vjebe svodi se na **dopunjavanje ove biljenice**: umetanja elije ili vie njih **ispod** teksta zadatka, pisanja odgovarajueg kôda te evaluiranja elija.

Osigurajte da u potpunosti **razumijete** kôd koji ste napisali. Kod predaje vjebe, morate biti u stanju na zahtjev asistenta (ili demonstratora) preinaiti i ponovno evaluirati Va kôd. Nadalje, morate razumjeti teorijske osnove onoga to radite, u okvirima onoga to smo obradili na predavanju. Ispod nekih zadataka moete nai i pitanja koja slue kao smjernice za bolje razumijevanje gradiva (**nemojte pisati** odgovore na pitanja u biljenicu). Stoga se nemojte ograniiti samo na to da rijeite zadatak, nego slobodno eksperimentirajte. To upravo i jest svrha ovih vjebi.

Vjebe trebate raditi **samostalno**. Moete se konzultirati s drugima o naelnom nainu rjeavanja, ali u konanici morate sami odraditi vjebu. U protivnome vjeba nema smisla.

[1]:
```
# Uitaj osnovne biblioteke...
import numpy as np
import sklearn
import matplotlib.pyplot as plt
%pylab inline
```

```
Populating the interactive namespace from numpy and matplotlib
```

0.2 Zadatci

0.2.1 1. Jednostavna regresija

Zadan je skup primjera $\mathcal{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^{4} = \{(0,4),(1,1),(2,2),(4,5)\}$. Primjere predstavite matrixom $\mathbf{X}$ dimenzija $N \times n$ (u ovom sluaju 4×1) i vektorom oznaka $\mathbf{y}$, dimenzija $N \times 1$ (u ovom sluaju 4×1), na sljedei nain:

[2]:
```
X = np.array([[0],[1],[2],[4]])
y = np.array([4,1,2,5])
```

0.2.2 (a)

Prouite funkciju `PolynomialFeatures` iz biblioteke `sklearn` i upotrijebite je za generiranje matrice dizajna $\mathbf{\Phi}$ koja ne koristi preslikavanje u prostor vie dimenzije (samo e svakom primjeru biti dodane *dummy* jedinice; $m = n + 1$).

[3]:
```
from sklearn.preprocessing import PolynomialFeatures
poly = PolynomialFeatures(1)
Fi = poly.fit_transform(X)
print(Fi)
```

```
[[1. 0.]
 [1. 1.]
 [1. 2.]
 [1. 4.]]
```

0.2.3 (b)

Upoznajte se s modulom `linalg`. Izraunajte teine $\mathbf{w}$ modela linearne regresije kao $\mathbf{w} = (\mathbf{\Phi}^\intercal\mathbf{\Phi})^{-1}\mathbf{\Phi}^\intercal\mathbf{y}$. Zatim se uvjerite da isti rezultat moete dobiti izraunom pseudoinverza $\mathbf{\Phi}^+$ matrice dizajna, tj. $\mathbf{w} = \mathbf{\Phi}^+\mathbf{y}$, koritenjem funkcije `pinv`.

[4]:
```
from numpy import linalg
```

[5]:
```
def calc_weight(Fi,y):
    Fi_plus  = linalg.pinv(Fi)
    return Fi_plus.dot(y)

Fi_other = linalg.inv((Fi.transpose().dot(Fi)))
Fi_other = Fi_other.dot(Fi.transpose())

print(calc_weight(Fi,y))
print("---------")
print(Fi_other.dot(y))
```

```
[2.2        0.45714286]
---------
[2.2        0.45714286]
```

Radi jasnoe, u nastavku je vektor $\mathbf{x}$ s dodanom *dummy* jedinicom $x_0 = 1$ oznaen kao $\tilde{\mathbf{x}}$.

0.2.4 (c)

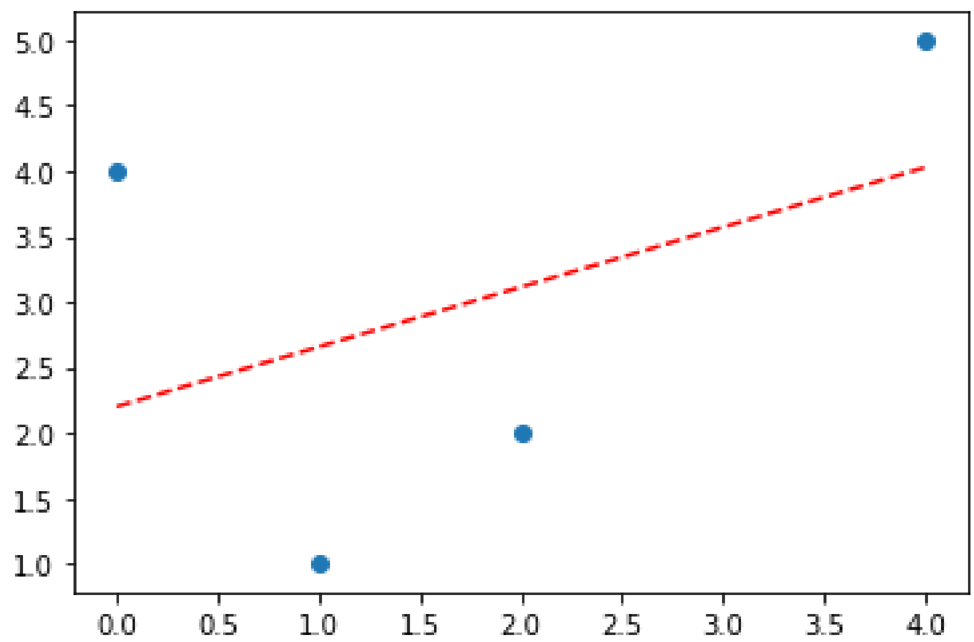
Prikaite primjere iz $\mathcal{D}$ i funkciju $h(\tilde{\mathbf{x}}) = \mathbf{w}^\intercal \tilde{\mathbf{x}}$. Izraunajte pogreku uenja prema izrazu $E(h|\mathcal{D}) = \frac{1}{2}\sum_{i=1}^{N}(\tilde{\mathbf{y}}^{(i)} - h(\tilde{\mathbf{x}}))^2$. Moete koristiti funkciju srednje kvadratne pogreke `mean_squared_error` iz modula `sklearn.metrics`.

Q: Gore definirana funkcija pogreke $E(h|\mathcal{D})$ i funkcija srednje kvadratne pogreke nisu posve identine. U emu je razlika? Koja je "realnija"?

[6]:
```
from sklearn.metrics import mean_squared_error

def h(x_tilda,w):
    return x_tilda.dot(w)

def Error(y, y_true):
    return sum((y-y_true)**2)/2

D = list(zip(Fi,y))
w = calc_weight(Fi,y)

y_pred = list(map(lambda x_tilda : h(x_tilda, w), Fi))
e1 = Error(y, y_pred)
# constant is len(y)
e2 = mean_squared_error(y, y_pred)

print("we : {}".format(e1))
print("sklearn : {}".format(e2))

plt.scatter(X,y)
plt.plot(X,y_pred,"r--")
```

```
we : 4.085714285714286
sklearn : 2.042857142857143
```

[6]:
```
[<matplotlib.lines.Line2D at 0x7fe07ea21a58>]
```



0.2.5 (d)

Uvjerite se da za primjere iz $\mathcal{D}$ teine $\mathbf{w}$ ne moemo nai rjeavanjem sustava $\mathbf{w} = \mathbf{\Phi}^{-1}\mathbf{y}$, ve da nam doista treba pseudoinverz.

Q: Zato je to sluaj? Bi li se problem mogao rijeiti preslikavanjem primjera u viu dimenziju? Ako da, bi li to uvijek funkcioniralo, neovisno o skupu primjera $\mathcal{D}$? Pokaite na primjeru.

[7]:
```
X_ = np.array([[0, 10],[1,2],[2,3],[4,9],[5,10],[7,3]])
print(X_)
poly = PolynomialFeatures(2)
Fi = poly.fit_transform(X_)
print(Fi)
print(linalg.matrix_rank(Fi))
linalg.inv(Fi)
```

```
[[ 0 10]
 [ 1  2]
 [ 2  3]
 [ 4  9]
 [ 5 10]
 [ 7  3]]
[[  1.   0.  10.   0.   0. 100.]
 [  1.   1.   2.   1.   2.   4.]
 [  1.   2.   3.   4.   6.   9.]
 [  1.   4.   9.  16.  36.  81.]
 [  1.   5.  10.  25.  50. 100.]
```

```
 [ 1.   7.   3.  49.  21.   9.]]
6
```

[7]:
```
array([[ 0.52380952, -0.83333333,  3.66666667, -6.11111111,  4.16666667,
        -0.41269841],
       [ 0.8       , -6.25      ,  9.        , -8.33333333,  5.45      ,
        -0.66666667],
       [-0.71428571,  4.25      , -6.7       ,  7.83333333, -5.25      ,
         0.58095238],
       [-0.06666667,  0.58333333, -0.86666667,  0.77777778, -0.51666667,
         0.08888889],
       [-0.06666667,  0.33333333, -0.46666667,  0.44444444, -0.26666667,
         0.02222222],
       [ 0.07619048, -0.41666667,  0.63333333, -0.72222222,  0.48333333,
        -0.05396825]])
```

0.2.6 (e)

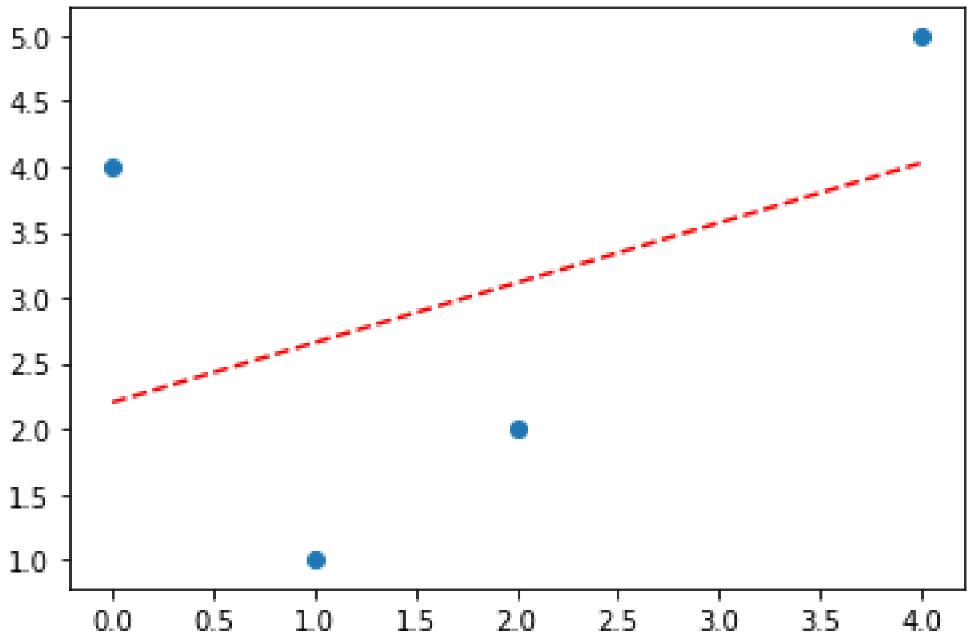
Prouite klasu `LinearRegression` iz modula `sklearn.linear_model`. Uvjerite se da su teine koje izraunava ta funkcija (dostupne pomou atributa `coef_` i `intercept_`) jednake onima koje ste izraunali gore. Izraunajte predikcije modela (metoda `predict`) i uvjerite se da je pogreka uenja identina onoj koju ste ranije izraunali.

[8]:
```
from sklearn.linear_model import LinearRegression
```

[9]:
```
reg = LinearRegression()
reg = reg.fit(X,y)
w0 = reg.intercept_
w1 = reg.coef_[0]
print("sklearn : {}".format([w0,w1]))
print("we : {}".format(w))

y_pred = reg.predict(X)
print("erorr : {}".format(mean_squared_error(y_pred,y)))

plt.scatter(X,y)
plt.plot(X,w0 + w1*X,"r--")
```

```
sklearn : [2.2, 0.45714285714285713]
we : [2.2        0.45714286]
erorr : 2.042857142857143
```

[9]:
```
[<matplotlib.lines.Line2D at 0x7fe07e6413c8>]
```



0.2.7 2. Polinomijalna regresija i utjecaj uma

0.2.8 (a)

Razmotrimo sada regresiju na veem broju primjera. Definirajte funkciju `make_labels(X, f, noise=0)` koja uzima matricu neoznaenih primjera $\mathbf{X}_{N\times n}$ te generira vektor njihovih oznaka $\mathbf{y}_{N\times 1}$. Oznake se generiraju kao $y^{(i)} = f(x^{(i)}) + \mathcal{N}(0, \sigma^2)$, gdje je $f : \mathbb{R}^n \to \mathbb{R}$ stvarna funkcija koja je generirala podatke (koja nam je u stvarnosti nepoznata), a σ je standardna devijacija Gaussovog uma, definirana parametrom `noise`. Za generiranje uma moete koristiti funkciju `numpy.random.normal`.

Generirajte skup za uenje od $N = 50$ primjera uniformno distribuiranih u intervalu $[-5, 5]$ pomou funkcije $f(x) = 5 + x - 2x^2 - 5x^3$ uz um $\sigma = 200$:

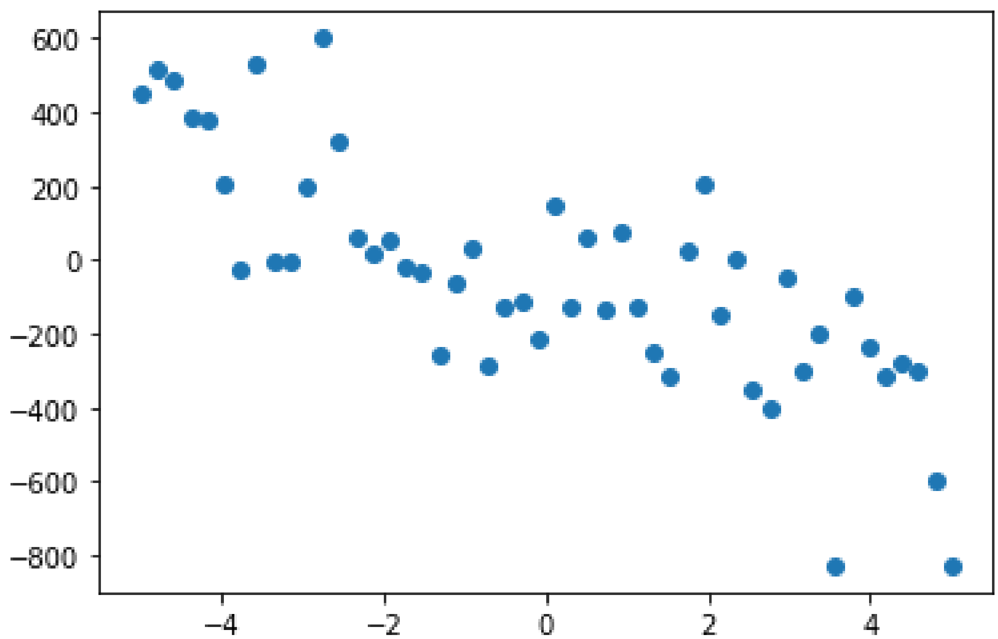
[10]:
```
from numpy.random import normal

def f(x):
    return 5 + x - 2*x**2 - 5*x**3

def make_labels(X, f, noise=0) :
    RV = np.random.normal(loc = 0, scale = noise, size = len(X))
    RV = RV.reshape(len(X),1)
    return f(X) + RV
```

[11]:
```
def make_instances(x1, x2, N) :
    return np.array([np.array([x]) for x in np.linspace(x1,x2,N)])
```

Prikaite taj skup funkcijom `scatter`.

[12]:
```python
X = make_instances(-5,5,50)
y = make_labels(X, f, 200)

plt.scatter(X,y)
```

[12]:
```
<matplotlib.collections.PathCollection at 0x7fe07e4cb160>
```

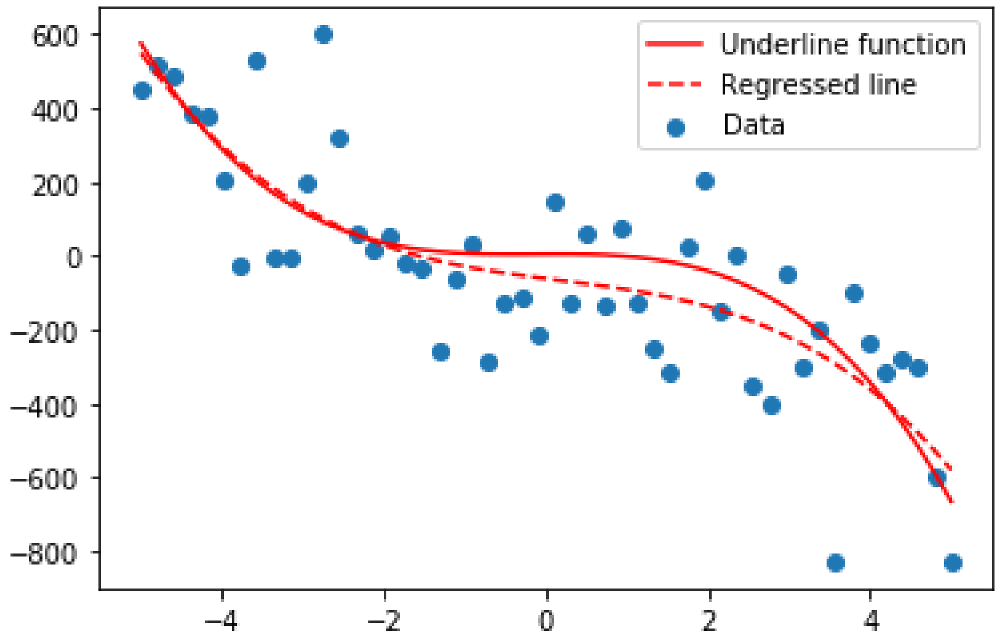


0.2.9 (b)

Trenirajte model polinomijalne regresije stupnja $d = 3$. Na istom grafikonu prikaite naueni model $h(\mathbf{x}) = \mathbf{w}^\intercal \tilde{\mathbf{x}}$ i primjere za uenje. Izraunajte pogreku uenja modela.

[13]:
```python
poly = PolynomialFeatures(degree = 3)
Fi   = poly.fit_transform(X)
reg = LinearRegression().fit(Fi, y)

y_pred = reg.predict(Fi)
plt.scatter(X,y,label = "Data")
plt.plot(X,f(X), "r", label = "Underline function")
plt.plot(X, y_pred,"r--", label = "Regressed line")
plt.legend(loc = "best")

e = mean_squared_error(y, y_pred)
print("MSE: {}".format(e))
```

```
MSE: 33001.32929201811
```



0.2.10 3. Odabir modela

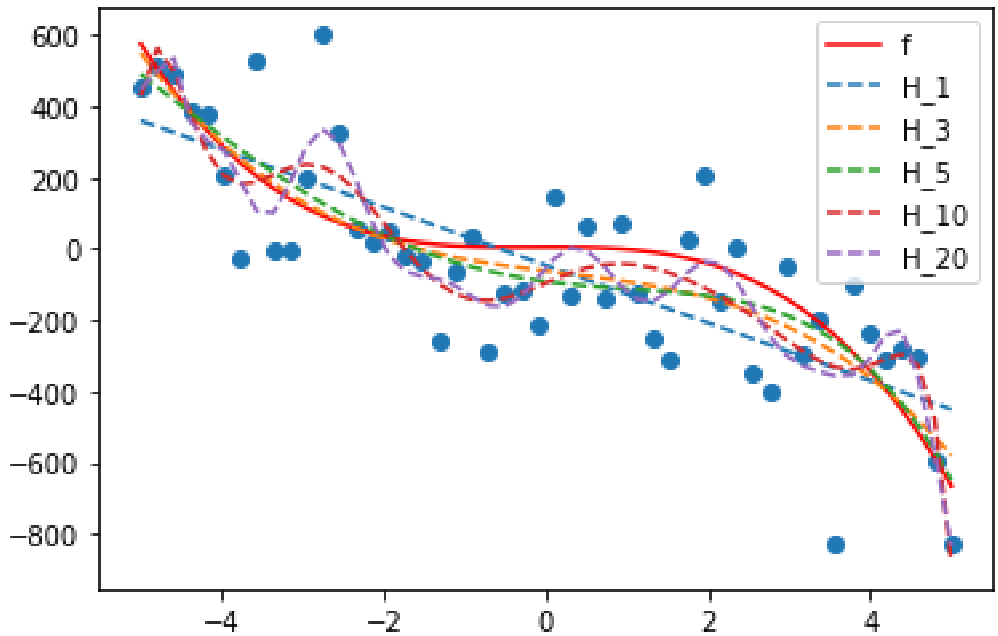
0.2.11 (a)

Na skupu podataka iz zadatka 2 trenirajte pet modela linearne regresije $\mathcal{H}_d$ razliite sloenosti, gdje je d stupanj polinoma, $d \in \{1, 3, 5, 10, 20\}$. Prikaite na istome grafikonu skup za uenje i funkcije $h_d(\mathbf{x})$ za svih pet modela (preporuujemo koristiti `plot` unutar `for` petlje). Izraunajte pogreku uenja svakog od modela.

Q: Koji model ima najmanju pogreku uenja i zato?

[14]:
```
d = [1,3,5,10,20]
plt.scatter(X,y)
plt.plot(X,f(X), "r", label = "f")

for i in d:
    poly = PolynomialFeatures(degree = i)
    Fi   = poly.fit_transform(X)
    reg = LinearRegression().fit(Fi, y)
    y_pred = reg.predict(Fi)

    plt.plot(X, y_pred,"--", label = "H_"+str(i))
    plt.legend(loc = "best")

    e = mean_squared_error(y, y_pred)
    print(f"MSE(H_{i}): {e}")
```

```
MSE(H_1): 37815.72686308033
MSE(H_3): 33001.32929201811
MSE(H_5): 32327.52418536688
MSE(H_10): 26297.99999771041
MSE(H_20): 23317.0508221788
```



0.2.12 (b)

Razdvojite skup primjera iz zadatka 2 pomou funkcije `model_selection.train_test_split` na skup za uenja i skup za ispitivanje u omjeru 1:1. Prikaite na jednom grafikonu pogreku uenja i ispitnu pogreku za modele polinomijalne regresije $\mathcal{H}_d$, sa stupnjem polinoma d u rasponu $d \in [1, 2, \ldots, 20]$. Budui da kvadratna pogreka brzo raste za vee stupnjeve polinoma, umjesto da iscrtate izravno iznose pogreaka, iscrtajte njihove logaritme.

NB: Podjela na skupa za uenje i skup za ispitivanje mora za svih pet modela biti identina.

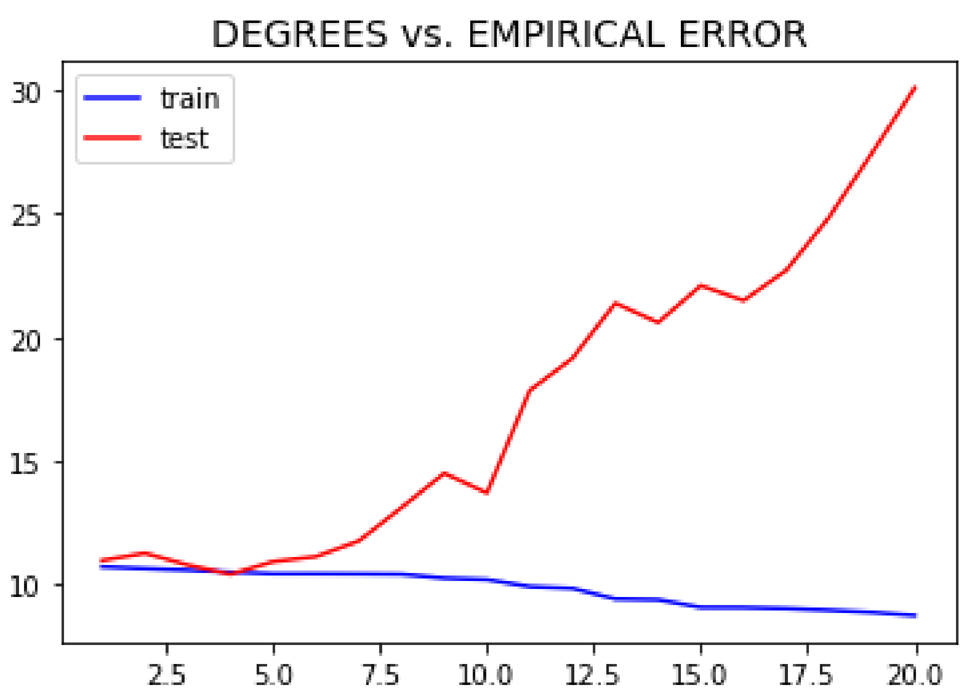
Q: Je li rezultat u skladu s oekivanjima? Koji biste model odabrali i zato?

Q: Pokrenite iscrtavanje vie puta. U emu je problem? Bi li problem bio jednako izraen kad bismo imali vie primjera? Zato?

[15]:
```python
from sklearn.model_selection import train_test_split
```

[45]:
```python
d = [i for i in range(1,21)]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.5)

train_errors = np.array([])
test_errors  = np.array([])
for i in d:
    poly = PolynomialFeatures(degree=i)
```

```
    Fi = poly.fit_transform(X_train)
    Fi_test = poly.fit_transform(X_test)

    reg = LinearRegression().fit(Fi, y_train)
    y_pred_train = reg.predict(Fi)
    y_pred_test = reg.predict(Fi_test)

    train_error = mean_squared_error(y_pred_train, y_train)
    test_error  = mean_squared_error(y_pred_test, y_test)

    train_errors = np.append(train_errors, train_error)
    test_errors  = np.append(test_errors, test_error)

train_errors = np.log(train_errors)
test_errors  = np.log(test_errors)

plt.plot(d,train_errors, "b", label = "train")
plt.plot(d, test_errors, "r", label = "test")
plt.legend(loc = "best")
_ = plt.title("DEGREES vs. EMPIRICAL ERROR", fontsize = 14)
```



0.2.13 (c)

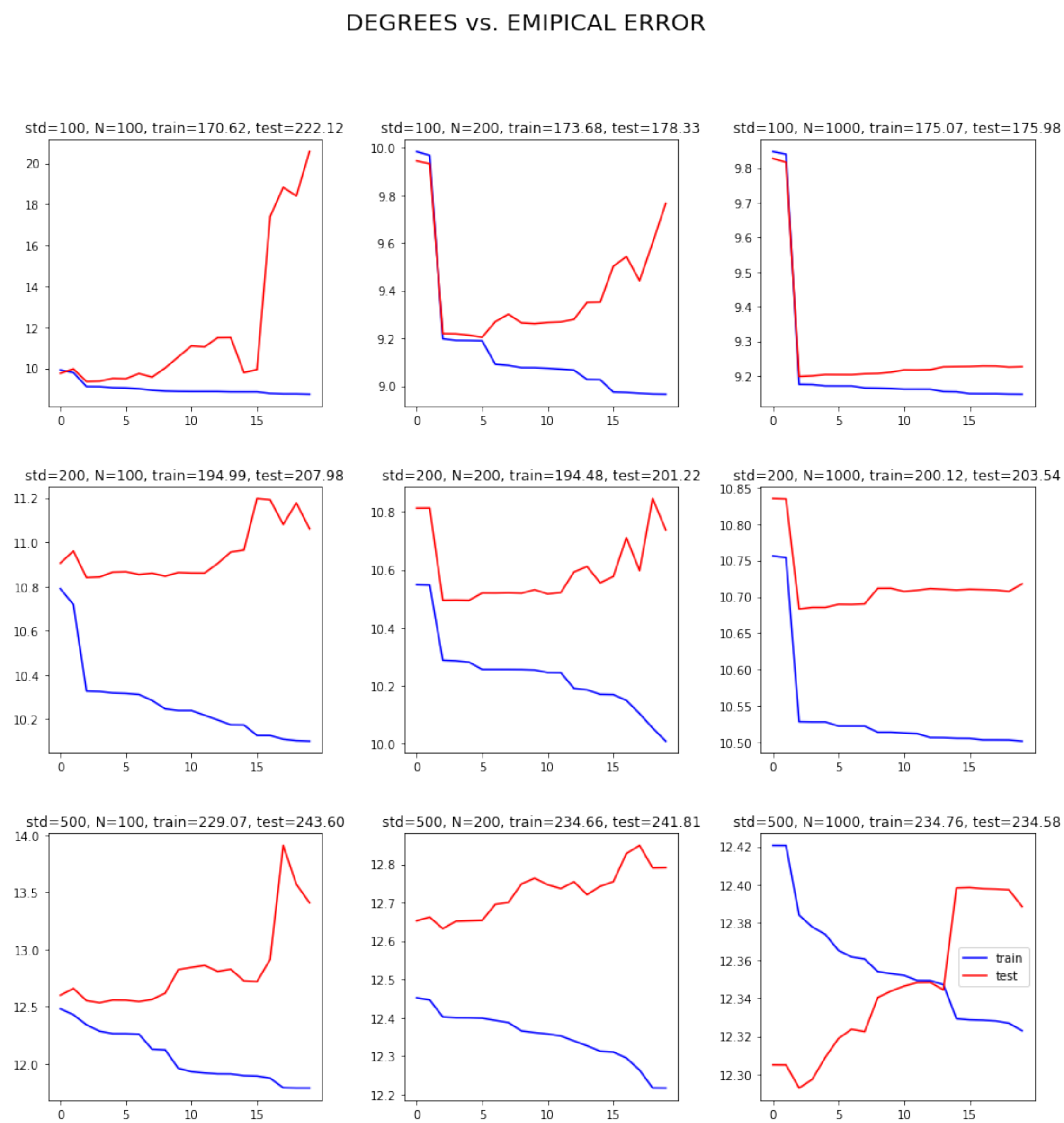
Tonost modela ovisi o (1) njegovoj sloenosti (stupanj d polinoma), (2) broju primjera N, i (3) koliini uma. Kako biste to analizirali, nacrtajte grafikone pogreaka kao u 3b, ali za sve kombinacija broja primjera $N \in \{100, 200, 1000\}$ i koliine uma $\sigma \in \{100, 200, 500\}$ (ukupno 9 grafikona). Upotrijebite funkciju `subplots` kako biste pregledno posloili grafikone u tablicu 3×3. Podatci se generiraju na isti nain kao u zadatku 2.

NB: Pobrinite se da svi grafikoni budu generirani nad usporedivim skupovima podataka, na sljedei nain. Generirajte najprije svih 1000 primjera, podijelite ih na skupove za uenje i skupove za ispitivanje (dva skupa od po 500 primjera). Zatim i od skupa za uenje i od skupa za ispitivanje nainite tri razliite verzije, svaka s drugaijom koliinom uma (ukupno 2x3=6 verzija podataka). Kako bi simulirali veliinu skupa podataka, od tih dobivenih 6 skupova podataka uzorkujte treinu, dvije treine i sve podatke. Time ste dobili 18 skupova podataka – skup za uenje i za testiranje za svaki od devet grafova.

[17]:
```
from functools import reduce
X = make_instances(x1 = -5, x2 = 5, N = 1000)
#y = make_labels(X, f, noise = 200)

#X_train, X_test, _, _ = train_test_split(X, y , test_size = 0.5)
sigma = [100, 200, 500]
Ns    = [100, 200]

D = [X]
for n in reversed(Ns):
    ratio = float(n/len(X))
    X_train, _ = train_test_split(X, train_size = ratio)
    D.insert(0, X_train)

datasets = []
for s in sigma:
    for i, d in enumerate(D):
        y = make_labels(d, f, noise = s)
        data = train_test_split(d, y , test_size = 0.5)
        data = (data, s, len(d))
        datasets.append(data)
```

[18]:
```
from scipy import integrate

degrees = [i for i in range(1,21)]
train_errors = np.array([])
test_errors  = np.array([])
errors       = {}
for i, D in enumerate(datasets):
    train_errors = np.array([])
    test_errors  = np.array([])
    (X_train, X_test, y_train, y_test), sigma, N = D
    for d_i in degrees:
        poly = PolynomialFeatures(degree=d_i)
        Fi_train = poly.fit_transform(X_train)
        Fi_test = poly.fit_transform(X_test)
```

```
        reg = LinearRegression().fit(Fi_train, y_train)
        y_pred_train = reg.predict(Fi_train)
        y_pred_test = reg.predict(Fi_test)

        train_error = mean_squared_error(y_pred_train, y_train)
        test_error  = mean_squared_error(y_pred_test, y_test)

        train_errors = np.append(train_errors, train_error)
        test_errors  = np.append(test_errors, test_error)
    errors[i] = (np.log(train_errors), np.log(test_errors), sigma, N)


fig, axs = plt.subplots(3, 3, figsize = (15,15))
axs = axs.flatten()
for i, a in enumerate(axs):
    train_error, test_error, sigma, N = errors[i]
    a.plot(train_error, "b", label = "train")
    a.plot(test_error, "r", label = "test")
    train_int = integrate.simps(train_error)
    test_int = integrate.simps(test_error)
    a.set_title("std={}, N={}, train={:0.2f}, test={:0.2f}".format(sigma, N,␣
 ↪train_int, test_int))
plt.legend(loc = "right")
fig.subplots_adjust(wspace = 0.3,hspace = 0.3)

_ = fig.suptitle('DEGREES vs. EMIPICAL ERROR',fontsize = 20)
```



Q: Jesu li rezultati oekivani? Obrazloite.

0.2.14 4. Regularizirana regresija

0.2.15 (a)

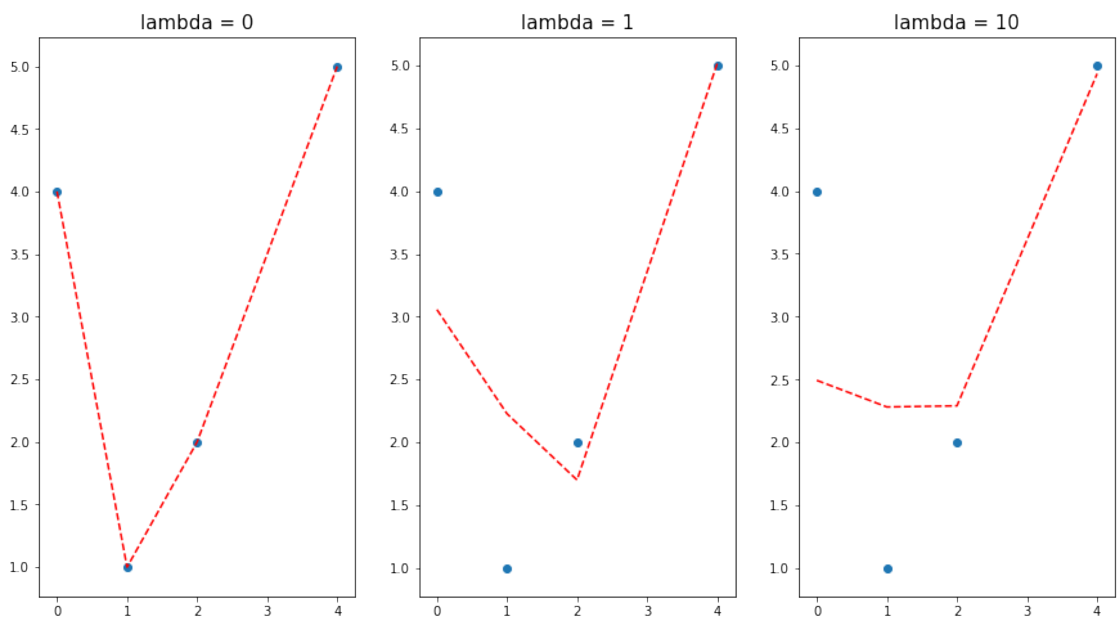
U gornjim eksperimentima nismo koristili **regularizaciju**. Vratimo se najprije na primjer iz zadatka 1. Na primjerima iz tog zadatka izraunajte teine $\mathbf{w}$ za polinomijalni regresijski model stupnja $d = 3$ uz L2-regularizaciju (tzv. *ridge regression*), prema izrazu $\mathbf{w} = (\mathbf{\Phi}^\intercal\mathbf{\Phi} + \lambda\mathbf{I})^{-1}\mathbf{\Phi}^\intercal\mathbf{y}$. Napravite izraun teina za regularizacijske faktore $\lambda = 0$, $\lambda = 1$ i $\lambda = 10$ te usporedite dobivene teine.

Q: Kojih je dimenzija matrica koju treba invertirati?

Q: Po emu se razlikuju dobivene teine i je li ta razlika oekivana? Obrazloite.

[19]:
```python
from numpy import linalg

def ridge_reg(Fi, y, scalar):
    E = np.eye(len(Fi)) * scalar
    E[0,0] = 0
    G = Fi.transpose().dot(Fi) + E
    G_i = linalg.inv(G)
    G_plus = G_i.dot(Fi.transpose())
    return G_plus.dot(y)

X = np.array([[0],[1],[2],[4]])
y = np.array([4,1,2,5])
Fi  = PolynomialFeatures(3).fit_transform(X)

fig, axs = plt.subplots(1, 3, figsize = (15,8))
lambdas = [0, 1, 10]
ws = []
for i, lam in enumerate(lambdas):
    w = ridge_reg(Fi, y, lam)
    ws.append(w)
    y_pred = Fi.dot(w)
    axs[i].plot(X, y_pred, "r--")
    axs[i].scatter(X, y)
    axs[i].set_title("lambda = {}".format(lam), fontsize = 15)

for lamb,w in zip(lambdas,ws):
    print(str(lamb) + "->" + str(w))
```

```
0->[ 4.         -5.91666667  3.375      -0.45833333]
1->[ 3.05696145 -0.69079365 -0.2831746   0.1445805 ]
10->[ 2.49444184 -0.15897295 -0.13423067  0.0815601 ]
```



0.2.16 (b)

Prouite klasu `Ridge` iz modula `sklearn.linear_model`, koja implementira L2-regularizirani regresijski model. Parametar α odgovara parametru λ. Primijenite model na istim primjerima kao u prethodnom zadatku i ispiite teine **w** (atributi `coef_` i `intercept_`).

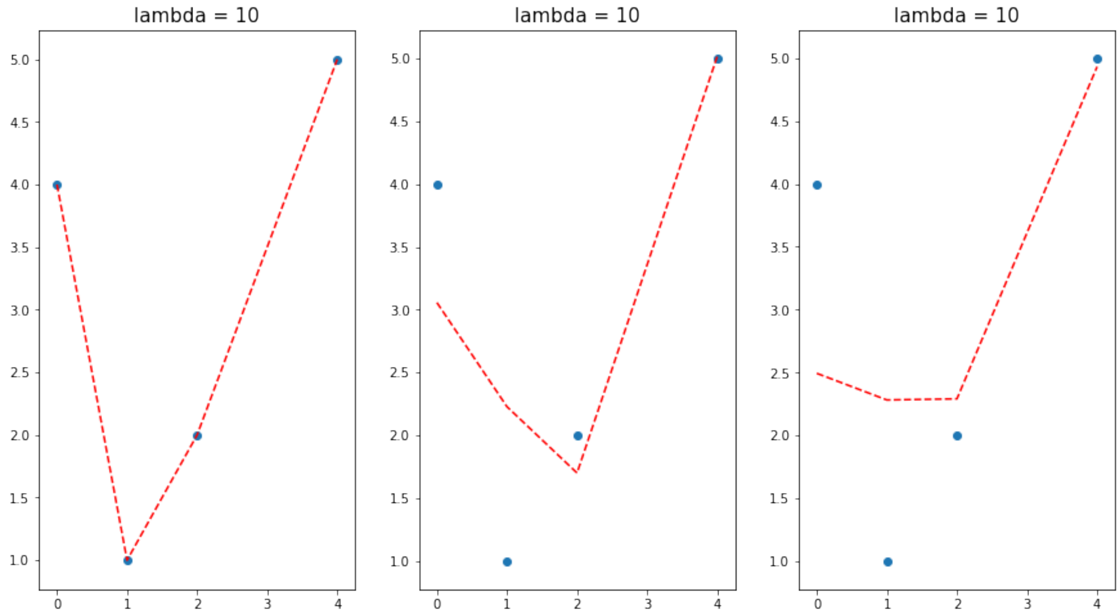
Q: Jesu li teine identine onima iz zadatka 4a? Ako nisu, objasnite zato je to tako i kako biste to popravili.

[20]:
```
from sklearn.linear_model import Ridge
```

[21]:
```
fig, axs = plt.subplots(1, 3, figsize = (15,8))
lambdas = [0, 1, 10]
ws = []
for i, alpha in enumerate(lambdas):
    reg = Ridge(alpha = alpha).fit(Fi, y)
    y_pred = reg.predict(Fi)
    axs[i].plot(X, y_pred, "r--")
    axs[i].scatter(X, y)
    axs[i].set_title("lambda = {}".format(lam), fontsize = 15)

    w  = np.concatenate(([reg.intercept_], reg.coef_[1:] ))
    ws.append(w)

for w in ws:
    print(w)
```

```
[ 4.         -5.91666667  3.375      -0.45833333]
[ 3.05696145 -0.69079365 -0.2831746   0.1445805 ]
```

```
[ 2.49444184 -0.15897295 -0.13423067  0.0815601 ]
```



0.2.17 5. Regularizirana polinomijalna regresija

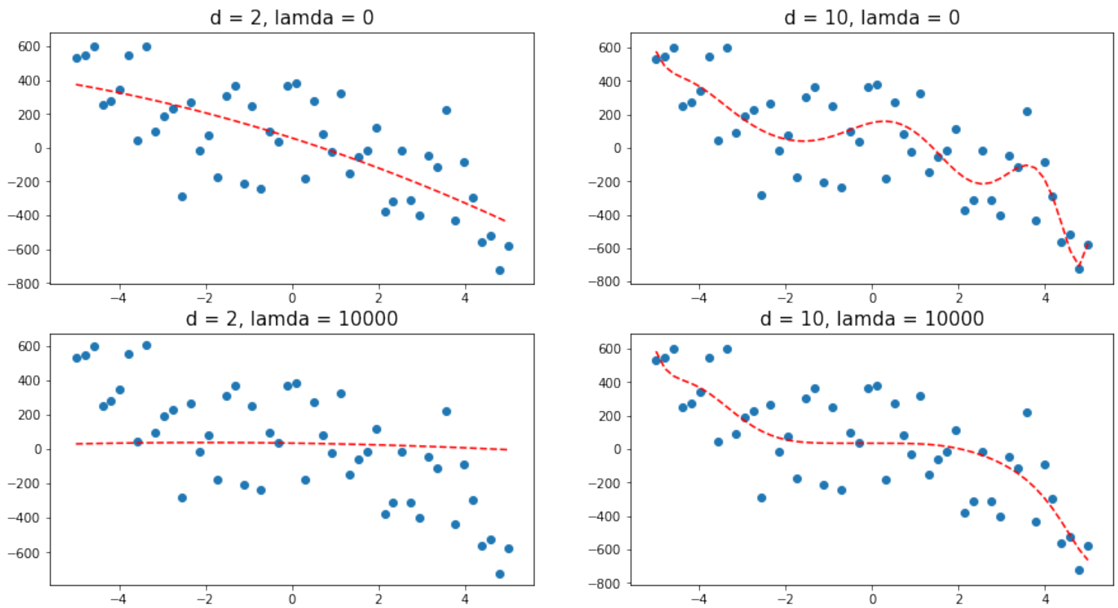
0.2.18 (a)

Vratimo se na sluaj $N = 50$ sluajno generiranih primjera iz zadatka 2. Trenirajte modele polinomijalne regresije $\mathcal{H}_{\lambda,d}$ za $\lambda \in \{0, 100\}$ i $d \in \{2, 10\}$ (ukupno etiri modela). Skicirajte pripadne funkcije $h(\mathbf{x})$ i primjere (na jednom grafikonu; preporuujemo koristiti `plot` unutar `for` petlje).

Q: Jesu li rezultati oekivani? Obrazloite.

[22]:
```
import warnings; warnings.simplefilter('ignore')

X = make_instances(-5,5,50)
y = make_labels(X,f,200)
lambdas = [0, 10000]
ds      = [2, 10]

fig, axs = plt.subplots(2, 2, figsize = (15,8))
axs = axs.flatten()

k = 0
for i, alpha in enumerate(lambdas):
    reg  = Ridge(alpha)
    for j, d in enumerate(ds):
        poly = PolynomialFeatures(d)
        Fi   = poly.fit_transform(X)
        reg  = reg.fit(Fi, y)
```

```
        y_pred = reg.predict(Fi)
        axs[k].plot(X, y_pred, "r--")
        axs[k].scatter(X, y)

        string = "d = {}, lamda = {}".format(d, alpha)
        axs[k].set_title(string, fontsize = 15)
        k += 1
```



0.2.19 (b)

Kao u zadataku 3b, razdvojite primjere na skup za uenje i skup za ispitivanje u omjeru 1:1. Prikaite krivulje logaritama pogreke uenja i ispitne pogreke u ovisnosti za model $\mathcal{H}_{d=10,\lambda}$, podeavajui faktor regularizacije λ u rasponu $\lambda \in \{0, 1, \ldots, 50\}$.

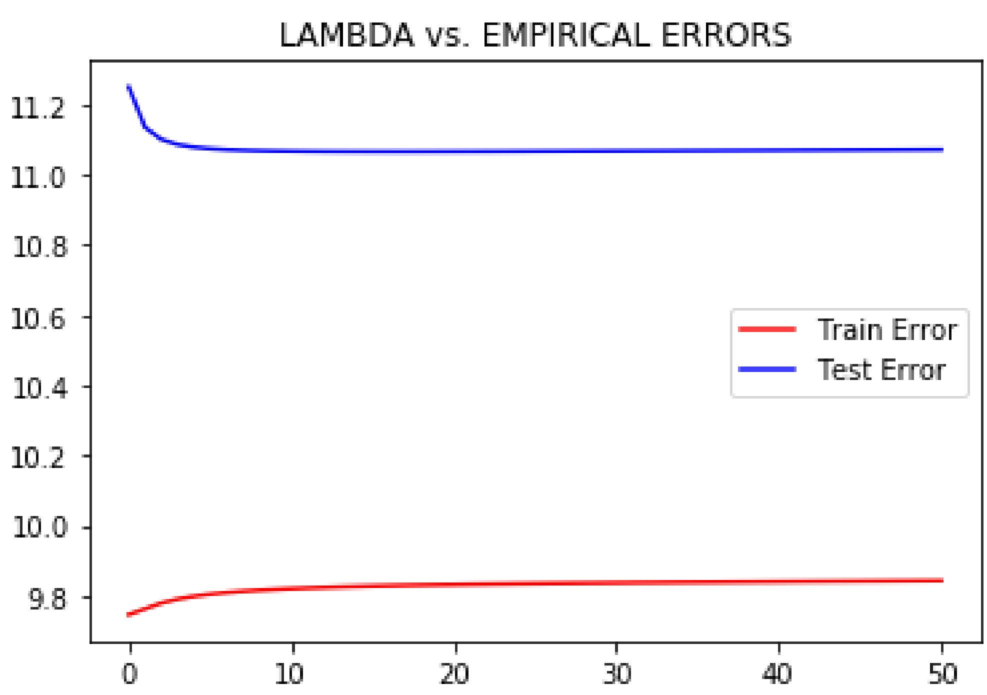
Q: Kojoj strani na grafikonu odgovara podruje prenauenosti, a kojoj podnauenosti? Zato?

Q: Koju biste vrijednosti za λ izabrali na temelju ovih grafikona i zato?

[23]:
```
X = make_instances(-5,5,50)
y = make_labels(X, f, 200)

X_train, X_test, y_train, y_test = train_test_split(X, y , test_size = 0.5)
Fi_train = PolynomialFeatures(degree = 10).fit_transform(X_train)
Fi_test = PolynomialFeatures(degree = 10).fit_transform(X_test)

train_errors = []
```

```
test_errors = []
for alpha in range(51):
    reg = Ridge(alpha).fit(Fi_train, y_train)
    y_pred_train = reg.predict(Fi_train)
    y_pred_test  = reg.predict(Fi_test)

    e_train = mean_squared_error(y_pred_train, y_train)
    e_test  = mean_squared_error(y_pred_test, y_test)

    train_errors.append(e_train)
    test_errors.append(e_test)

train_errors = np.log(train_errors)
test_errors = np.log(test_errors)

plt.plot(train_errors, "r", label = "Train Error")
plt.plot(test_errors, "b", label  = "Test Error")
plt.legend(loc = "best")
plt.title("LAMBDA vs. EMPIRICAL ERRORS")
```

[23]:
```
Text(0.5, 1.0, 'LAMBDA vs. EMPIRICAL ERRORS')
```



0.2.20 6. L1-regularizacija i L2-regularizacija

Svrha regularizacije jest potiskivanje teina modela **w** prema nuli, kako bi model bio to jednostavniji. Sloenost modela moe se okarakterizirati normom pripadnog vektora teina **w**, i to tipino L2-normom ili L1-normom. Za jednom trenirani model moemo izraunati i broj ne-nul znaajki, ili L0-normu, pomou sljedee funkcije koja prima vektor teina **w**:

[24]:
```
def nonzeroes(coef, tol=1e-6):
    return len(coef) - len(coef[np.isclose(0, coef, atol=tol)])

def L2_norm(w):
    from math import sqrt
    return sqrt(w.dot(w.transpose()))

def L1_norm(w):
    w = abs(np.array(w))
    return sum(w)

def L0_norm(w):
    return nonzeroes(w)
```

0.2.21 (a)

Za ovaj zadatak upotrijebite skup za uenje i skup za testiranje iz zadatka 3b. Trenirajte modele **L2-regularizirane** polinomijalne regresije stupnja $d = 10$, mijenjajui hiperparametar λ u rasponu $\{1, 2, \ldots, 100\}$. Za svaki od treniranih modela izraunajte L{0,1,2}-norme vektora teina **w** te ih prikaite kao funkciju od λ. Pripazite to tono aljete u funkciju za izraun normi.

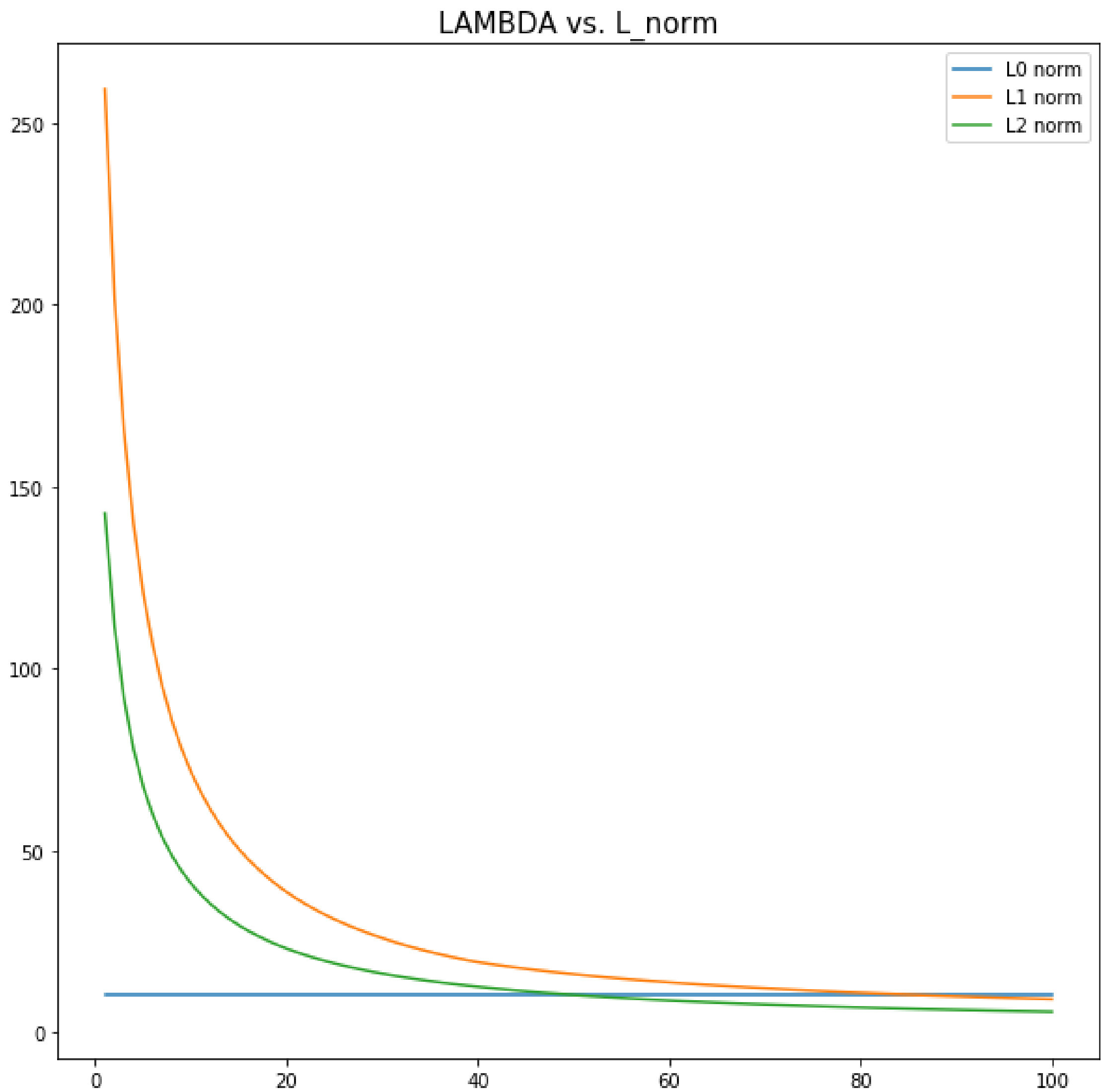
Q: Objasnite oblik obiju krivulja. Hoe li krivulja za $\|\mathbf{w}\|_2$ dosei nulu? Zato? Je li to problem? Zato?

Q: Za $\lambda = 100$, koliki je postotak teina modela jednak nuli, odnosno koliko je model rijedak?

[25]:
```
X = make_instances(x1 = -5, x2 = 5, N = 50)
y = make_labels(X, f, noise = 200)
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size = 0.7)
Fi_train = PolynomialFeatures(degree = 10).fit_transform(X_train)
Fi_test = PolynomialFeatures(degree = 10).fit_transform(X_test)

lambdas = [i for i in range(1, 101)]
l0 = []
l1 = []
l2 = []
for alpha in lambdas:
    reg = Ridge(alpha).fit(Fi_train, y_train)
    w  = reg.coef_[0][1:]
    l0.append(L0_norm(w))
    l1.append(L1_norm(w))
    l2.append(L2_norm(w))

plt.figure(figsize = (10,10))
```

```python
plt.plot(lambdas, l0, label = "L0 norm")
plt.plot(lambdas, l1, label = "L1 norm")
plt.plot(lambdas, l2, label = "L2 norm")

plt.legend(loc = "best")
plt.title("LAMBDA vs. L_norm", fontsize = 15)

print("sum(L0) = {}".format(sum(l0)))
print("sum(L1) = {}".format(sum(l1)))
print("sum(L2) = {}".format(sum(l2)))
```

```
sum(L0) = 1000
sum(L1) = 3105.5365526560527
sum(L2) = 1834.5612747481669
```

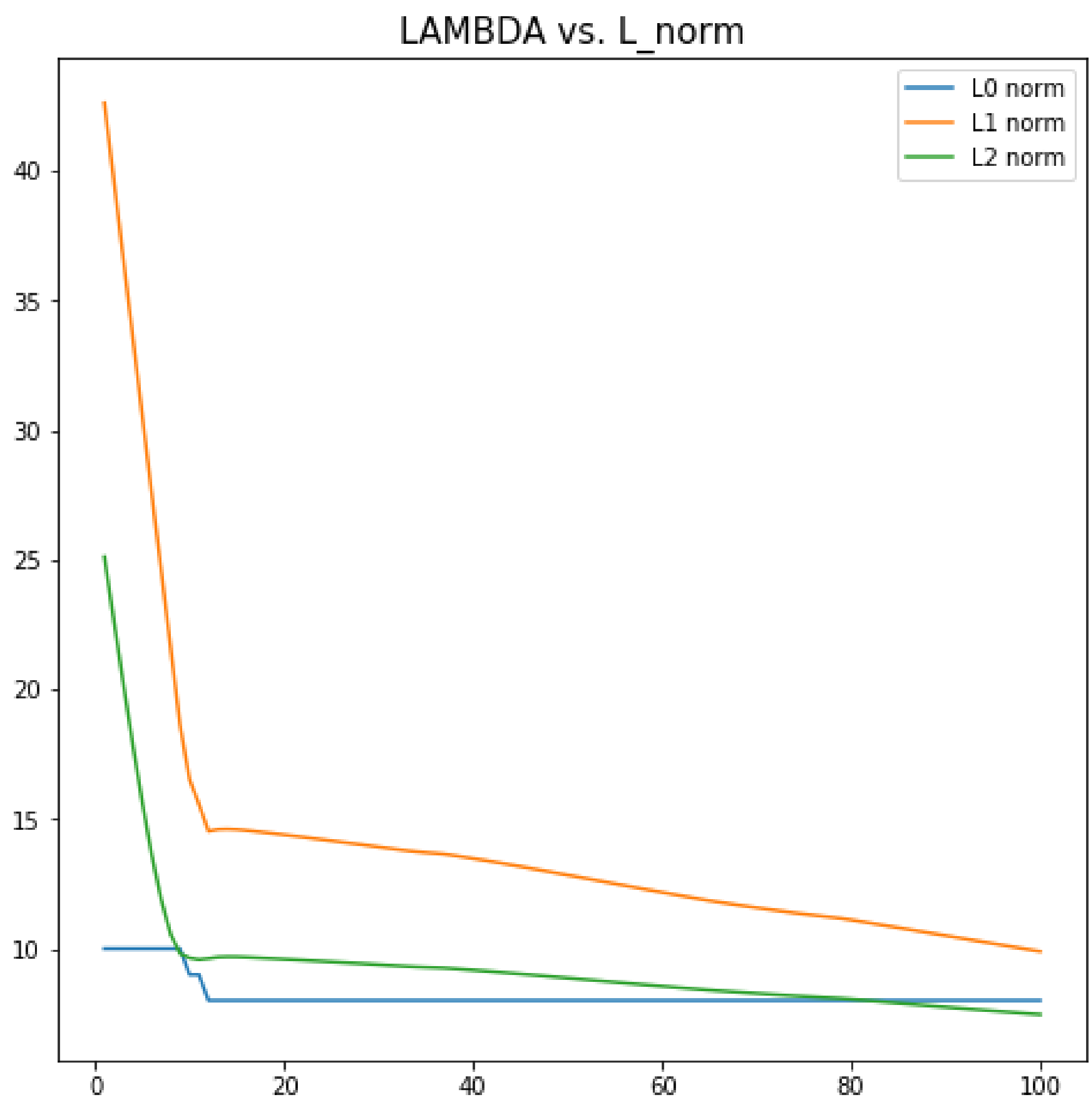


0.2.22 (b)

Glavna prednost L1-regularizirane regresije (ili *LASSO regression*) nad L2-regulariziranom regresijom jest u tome to L1-regularizirana regresija rezultira **rijetkim modelima** (engl. *sparse models*), odnosno modelima kod kojih su mnoge teine pritegnute na nulu. Pokaite da je to doista tako, ponovivi gornji eksperiment s **L1-regulariziranom** regresijom, implementiranom u klasi `Lasso` u modulu `sklearn.linear_model`. Zanemarite upozorenja.

[26]:
```
from sklearn.linear_model import Lasso
import warnings; warnings.simplefilter('ignore')

X = make_instances(x1 = -5, x2 = 5, N = 50)
y = make_labels(X, f, noise = 200)

X_train, X_test, y_train, y_test = train_test_split(X, y, train_size = 0.7)
Fi_train = PolynomialFeatures(degree = 10).fit_transform(X_train)
Fi_test = PolynomialFeatures(degree = 10).fit_transform(X_test)

lambdas = [i for i in range(1, 101)]
l0 = []
l1 = []
l2 = []
for alpha in lambdas:
    reg = Lasso(alpha).fit(Fi_train, y_train)
    w  = reg.coef_[1:]
    l0.append(L0_norm(w))
    l1.append(L1_norm(w))
    l2.append(L2_norm(w))

_, ax = plt.subplots(nrows=1, ncols=1,figsize=(8,8))
ax.plot(lambdas, l0, label = "L0 norm")
ax.plot(lambdas, l1, label = "L1 norm")
ax.plot(lambdas, l2, label = "L2 norm")

ax.legend(loc = "best")
plt.title("LAMBDA vs. L_norm", fontsize = 15)

print("sum(L0) = {}".format(sum(l0)))
print("sum(L1) = {}".format(sum(l1)))
print("sum(L2) = {}".format(sum(l2)))
```

```
sum(L0) = 820
sum(L1) = 1413.5431660432012
sum(L2) = 938.1076125682704
```



0.2.23 7. Znaajke razliitih skala

esto se u praksi moemo susreti sa podatcima u kojima sve znaajke nisu jednakih magnituda. Primjer jednog takvog skupa je regresijski skup podataka `grades` u kojem se predvia prosjek ocjena studenta na studiju (1–5) na temelju dvije znaajke: bodova na prijamnom ispitu (1–3000) i prosjeka ocjena u srednjoj koli. Prosjek ocjena na studiju izraunat je kao teinska suma ove dvije znaajke uz dodani um.

Koristite sljedei kôd kako biste generirali ovaj skup podataka.

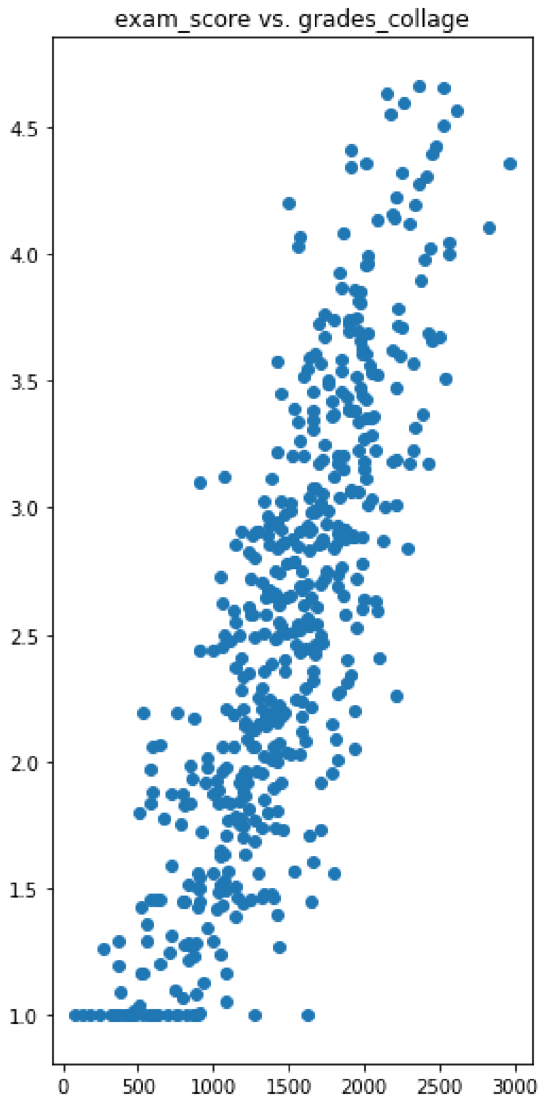
[27]:
```
n_data_points = 500
np.random.seed(69)
```

```
# Generiraj podatke o bodovima na prijamnom ispitu koristei normalnu razdiobu i␣
↪ogranii ih na interval [1, 3000].
exam_score = np.random.normal(loc=1500.0, scale = 500.0, size = n_data_points)
exam_score = np.round(exam_score)
exam_score[exam_score > 3000] = 3000
exam_score[exam_score < 0] = 0

# Generiraj podatke o ocjenama iz srednje kole koristei normalnu razdiobu i␣
↪ogranii ih na interval [1, 5].
grade_in_highschool = np.random.normal(loc=3, scale = 2.0, size = n_data_points)
grade_in_highschool[grade_in_highschool > 5] = 5
grade_in_highschool[grade_in_highschool < 1] = 1

# Matrica dizajna.
grades_X = np.array([exam_score,grade_in_highschool]).T

# Zavrno, generiraj izlazne vrijednosti.
rand_noise = np.random.normal(loc=0.0, scale = 0.5, size = n_data_points)
exam_influence = 0.9
grades_y = ((exam_score / 3000.0) * (exam_influence) + (grade_in_highschool / 5.
↪0) \
            * (1.0 - exam_influence)) * 5.0 + rand_noise
grades_y[grades_y < 1] = 1
grades_y[grades_y > 5] = 5
```

a) Iscrtajte ovisnost ciljne vrijednosti (y-os) o prvoj i o drugoj znaajki (x-os). Iscrtajte dva odvojena grafa.

[28]:
```
X0 = grades_X[:, 0]
X1 = grades_X[:, 1]

_, axs = plt.subplots(1,2, figsize = (10,10))
axs[0].scatter(X0, grades_y)
axs[1].scatter(X1, grades_y, color = "r")

axs[0].set_title("exam_score vs. grades_collage")
axs[1].set_title("grades_high_school vs. grades_collage")
```

[28]:
```
Text(0.5, 1.0, 'grades_high_school vs. grades_collage')
```



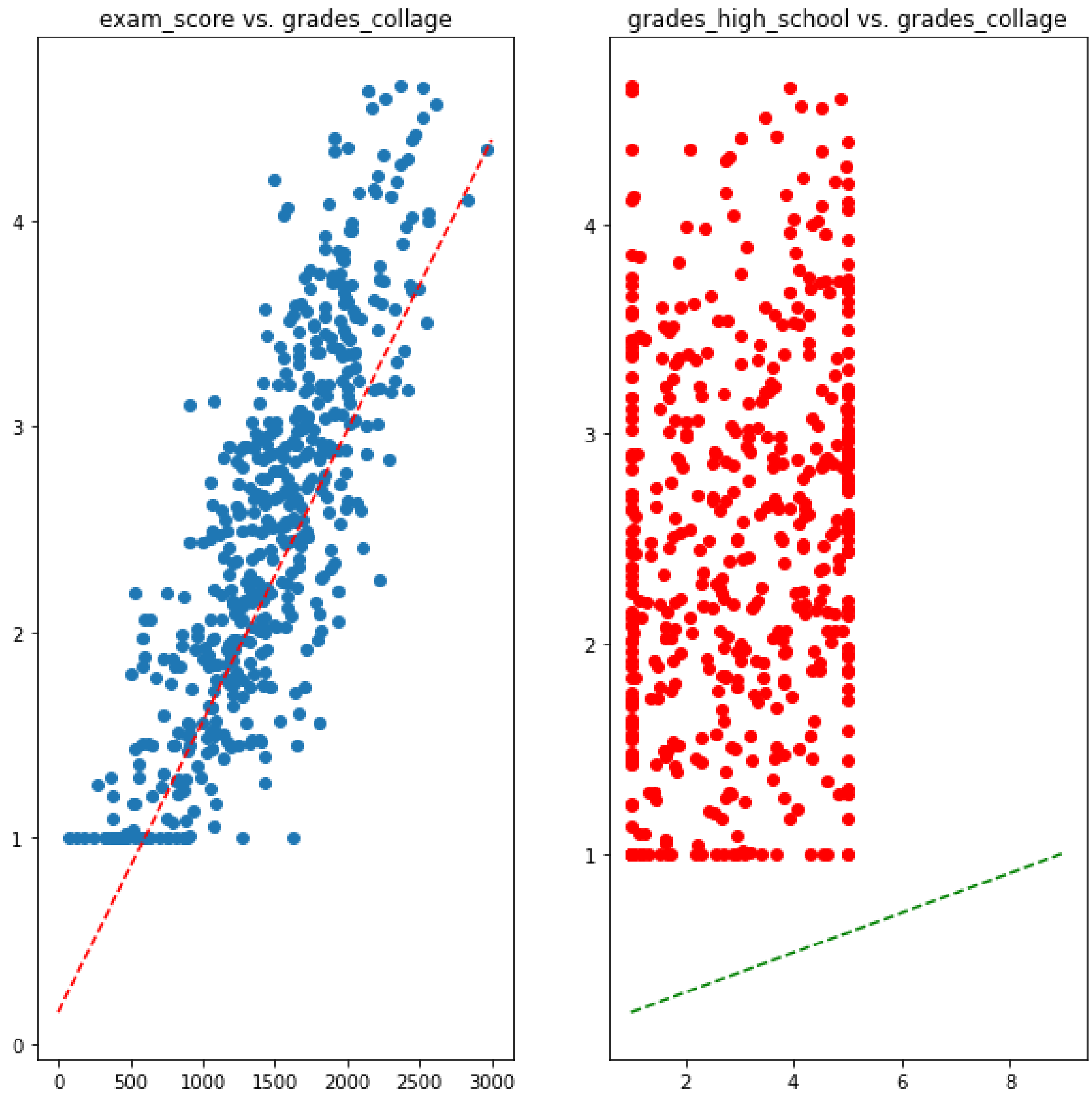
b) Nauite model L2-regularizirane regresije ($\lambda = 0.01$), na podacima grades_X i grades_y:

[29]:
```
alpha = 0.01
reg = Ridge(alpha).fit(grades_X, grades_y)

w0 = reg.intercept_
w  = reg.coef_
print(f"w0 = {w0}, w1 = {w[0]}, w2 = {w[1]}")

_, axs = plt.subplots(1,2, figsize = (10,10))

axs[0].scatter(X0, grades_y)
axs[1].scatter(X1, grades_y, color = "r")
axs[0].plot(np.arange(0,3000,1), w0 + w[0]*np.arange(0,3000,1), "r--")
```

```
axs[1].plot(np.arange(1,10,1), w0 + w[1]*np.arange(1,10,1), "g--")

axs[0].set_title("exam_score vs. grades_collage")
axs[1].set_title("grades_high_school vs. grades_collage")
```

```
w0 = 0.15061179575776018, w1 = 0.0014149686631874092, w2 = 0.09477275879174112
```

[29]:
```
Text(0.5, 1.0, 'grades_high_school vs. grades_collage')
```



Sada ponovite gornji eksperiment, ali prvo skalirajte podatke `grades_X` i `grades_y` i spremite ih u varijable `grades_X_fixed` i `grades_y_fixed`. Za tu svrhu, koristite `StandardScaler`.

[30]:
```
from sklearn.preprocessing import StandardScaler
```

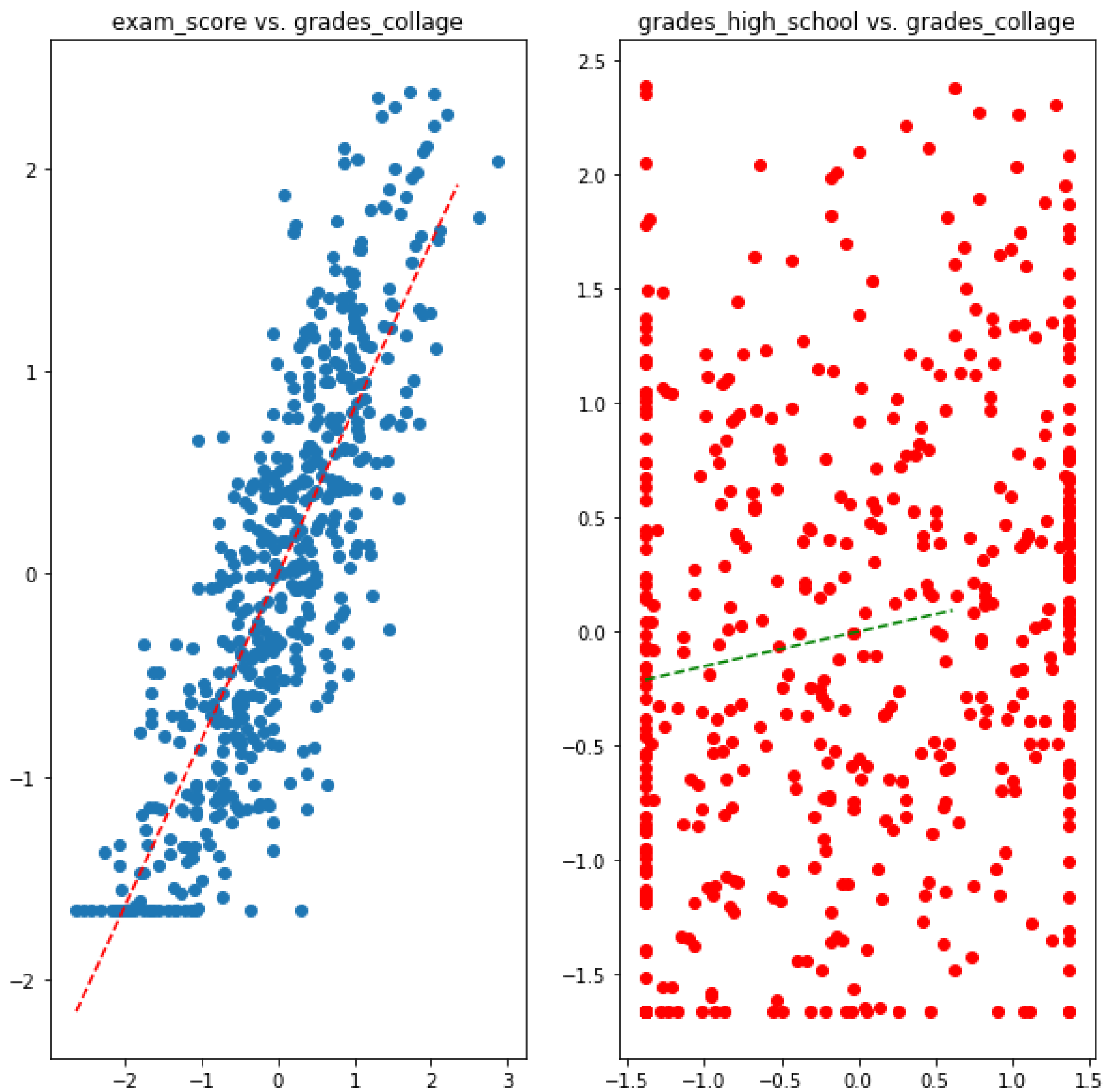
[31]:
```
scaler = StandardScaler()
scaler.fit(grades_X)
grades_x_fixed = scaler.transform(grades_X)
print("grades_X:")
print("  *mean:",scaler.mean_)
print("  *std:",scaler.var_**2)

scaler = StandardScaler()
ys = grades_y.reshape(-1,1)
scaler.fit(ys)
grades_y_fixed = scaler.transform(ys).flatten()
print("grades_y:")
print("  *mean:",scaler.mean_)
print("  *std:",scaler.var_**2)
```

```
grades_X:
  *mean: [1461.13          3.00684164]
  *std: [7.46294285e+10 4.41989341e+00]
grades_y:
  *mean: [2.50303164]
  *std: [0.67368295]
```

[32]:
```
alpha = 0.01
reg = Ridge(alpha).fit(grades_x_fixed, grades_y_fixed)

w0 = reg.intercept_
w  = reg.coef_
print("w0 = {}, w1 = {}, w2 = {}".format(w0, w[0], w[1]))

X0 = grades_x_fixed[:, 0]
X1 = grades_x_fixed[:, 1]

_, axs = plt.subplots(1,2, figsize = (10,10))

arr = np.arange(X0.flatten().min(), X0.flatten().max(), 1)
axs[0].plot(arr, w0 + w[0]*arr, "r--")

arr = np.arange(X1.flatten().min(), X1.flatten().max(), 1)
axs[1].plot(arr, w0 + w[1]*arr, "g--")


axs[0].scatter(X0, grades_y_fixed)
axs[1].scatter(X1, grades_y_fixed, color = "r")

axs[0].set_title("exam_score vs. grades_collage")
axs[1].set_title("grades_high_school vs. grades_collage")
```

```
w0 = 8.625513311328875e-17, w1 = 0.8163037502836595, w2 = 0.15167761205975838
```

[32]: `Text(0.5, 1.0, 'grades_high_school vs. grades_collage')`



Q: Gledajui grafikone iz podzadatka (a), koja znaajka bi trebala imati veu magnitudu, odnosno vanost pri predikciji prosjeka na studiju? Odgovaraju li teine Vaoj intuiciji? Objasnite.

0.2.24 8. Multikolinearnost i kondicija matrice

a) Izradite skup podataka `grades_X_fixed_colinear` tako to ete u skupu `grades_X_fixed` iz zadatka *7b* duplicirati zadnji stupac (ocjenu iz srednje kole). Time smo efektivno uveli savrenu multikolinearnost.

[33]:
```
grades_X_fixed_colinear = np.hstack((grades_x_fixed, grades_x_fixed[:,1].
↪reshape(-1,1)))
print(grades_X_fixed_colinear)
```

```
[[ 0.95063817 -0.78607869 -0.78607869]
 [-0.50343434 -0.50193004 -0.50193004]
 [ 1.18596832 -0.52213172 -0.52213172]
 ...
 [-0.31593552  0.80074234  0.80074234]
 [ 0.81288393  1.24783236  1.24783236]
 [-0.01938126  1.21729714  1.21729714]]
```

Ponovno, nauite na ovom skupu L2-regularizirani model regresije ($\lambda = 0.01$).

[34]:
```python
alpha = 0.01
reg = Ridge(alpha).fit(grades_X_fixed_colinear, grades_y_fixed)
w  = np.append(reg.intercept_,reg.coef_)
print(w)
```

```
[8.62559710e-17 8.16303637e-01 7.58395686e-02 7.58395686e-02]
```

Q: Usporedite iznose teina s onima koje ste dobili u zadatku *7b*. to se dogodilo?

b) Sluajno uzorkujte 50% elemenata iz skupa `grades_X_fixed_colinear` i nauite dva modela L2-regularizirane regresije, jedan s $\lambda = 0.01$ i jedan s $\lambda = 1000$). Ponovite ovaj pokus 10 puta (svaki put s drugim podskupom od 50% elemenata). Za svaki model, ispiite dobiveni vektor teina u svih 10 ponavljanja te ispiite standardnu devijaciju vrijednosti svake od teina (ukupno est standardnih devijacija, svaka dobivena nad 10 vrijednosti).

[35]:
```python
def sample(X,y,ratio):
    n = len(X)
    X_ = []
    y_ = []
    for i in range(int(ratio*n)):
        rand = np.random.randint(0,n)
        X_.append(X[rand])
        y_.append(y[rand])
    return np.array(X_), np.array(y_)
```

[36]:
```python
w0 = []
w1 = []
for i in range(10):
    X_train, y_train = sample(grades_X_fixed_colinear, grades_y, ratio = 0.5)
    reg1 = Ridge(alpha = 0.01).fit(X_train, y_train)
    reg2 = Ridge(alpha = 1000).fit(X_train, y_train)

    w_0 = np.append(reg1.intercept_ , reg1.coef_)
    w_1 = np.append(reg2.intercept_, reg2.coef_)

    w0.append(w_0)
    w1.append(w_1)

    print("iter = " + str(i))
    print("w0 = {}".format(w_0))
    print("w1 = {}".format(w_1))
```

```python
    print("--------------------")

w0 = np.array(w0)
w1 = np.array(w1)

print("model 1 : s0 = {}, s1 = {}, s2 = {}".format(w0[:,0].std(), w0[:,1].
 ↪std(), w0[:,2].std()))
print("model 2 : s0 = {}, s1 = {}, s2 = {}".format(w1[:,0].std(), w1[:,1].
 ↪std(), w1[:,2].std()))
```

```
iter = 0
w0 = [2.46050416 0.71066667 0.05253577 0.05253577]
w1 = [2.40893172 0.1544004  0.04780907 0.04780907]
--------------------
iter = 1
w0 = [2.51500787 0.76269546 0.09832572 0.09832572]
w1 = [2.46195654 0.15044662 0.033364   0.033364  ]
--------------------
iter = 2
w0 = [2.45948927 0.72503766 0.08387738 0.08387738]
w1 = [2.47831134 0.1451034  0.0271132  0.0271132 ]
--------------------
iter = 3
w0 = [2.52285503 0.76469684 0.04678085 0.04678085]
w1 = [2.54801874 0.15883391 0.02272297 0.02272297]
--------------------
iter = 4
w0 = [2.48076604 0.71545857 0.07144547 0.07144547]
w1 = [2.50808238 0.14636448 0.03105748 0.03105748]
--------------------
iter = 5
w0 = [2.4522034  0.72693194 0.05939941 0.05939941]
w1 = [2.44815027 0.14577985 0.02156075 0.02156075]
--------------------
iter = 6
w0 = [2.506411   0.76340264 0.05616862 0.05616862]
w1 = [2.4165627  0.1530837  0.02217751 0.02217751]
--------------------
iter = 7
w0 = [2.49867477 0.76868603 0.07205262 0.07205262]
w1 = [2.50198805 0.14105459 0.03844858 0.03844858]
--------------------
iter = 8
w0 = [2.48834108 0.73120579 0.07253901 0.07253901]
w1 = [2.43612868 0.15479163 0.03607719 0.03607719]
--------------------
iter = 9
```

```
w0 = [2.5390552  0.81979479 0.05863447 0.05863447]
w1 = [2.55512566 0.13947502 0.01671629 0.01671629]
-------------------
model 1 : s0 = 0.027783118264638584, s1 = 0.031643214435648014, s2 =
0.014848928541979414
model 2 : s0 = 0.04873397660031115, s1 = 0.006037314511284559, s2 =
0.00898440114525995
```

Q: Kako regularizacija utjee na stabilnost teina?
Q: Jesu li koeficijenti jednakih magnituda kao u prethodnom pokusu? Objasnite zato.

c) Koristei `numpy.linalg.cond` izraunajte kondicijski broj matrice $\mathbf{\Phi}^\intercal\mathbf{\Phi} + \lambda\mathbf{I}$, gdje je $\mathbf{\Phi}$ matrica dizajna (`grades_X_fixed_colinear`). Ponovite i za $\lambda = 0.01$ i za $\lambda = 10$.

[37]:
```python
lam = [0.01, 10]

Fi = grades_X_fixed_colinear
E = np.eye(3)
E[0,0] = 0

Gram = Fi.T.dot(Fi)
A = Gram + lam[0] * E
B = Gram + lam[1] * E

print("cond(A) = {}".format(linalg.cond(A)))
print("cond(B) = {}".format(linalg.cond(B)))
```

```
cond(A) = 100542.85592735428
cond(B) = 101.53146069838179
```

Q: Kako regularizacija utjee na kondicijski broj matrice $\mathbf{\Phi}^\intercal\mathbf{\Phi} + \lambda\mathbf{I}$?