Created by:

Ellák Somfai

Eötvös Loránd Tudományegyetem
Informatikai Kar

Linear regression

```
# imports
import numpy as np
from matplotlib.pyplot import *   # plotting data
```

We create synthetic data according to

$$t = y(x; w) + \epsilon = -1 + 2x + 0.5x^2 + \epsilon\,,$$

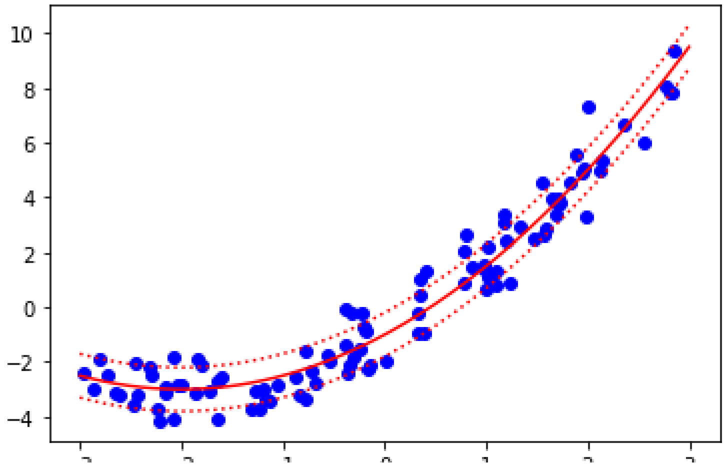
where the noise is normal: $\epsilon \sim \mathcal{N}(0, \sigma^2)$ with $\sigma = 0.8$.

The independent variable x is taken to be uniformly distributed between -3 and 3: $x \sim \mathcal{U}[-3, 3]$

```
N = 100          # number of data points
x0, x1 = -3, 3   # endpoints of the interval for x
sigma = 0.8      # noise amplitude
w_true = [-1, +2, 0.5]  # ground truth coefficients: w0, w1, w2

def model(x, w):
  return w[0] + w[1]*x + w[2]*x*x

rng = np.random.default_rng(42)  # setup random number generator with fixed seed

# independent variable: uniform in [x0, x1]
x = x0 + (x1-x0) * rng.random(N)

# target
t = model(x, w_true) + rng.normal(0, sigma, N)
```

```
# plot data (blue) and ground truth (red)
plot(x, t, 'bo')                  # data (blue)
x_plot = np.linspace(x0, x1, 200)
y_plot = model(x_plot, w_true)
plot(x_plot, y_plot, 'r-')        # ground truth (red line)
plot(x_plot, y_plot+sigma, 'r:')  # ground truth +- 1*sigma (red dotted)
plot(x_plot, y_plot-sigma, 'r:')
```

```
[<matplotlib.lines.Line2D at 0x7efbfc8f6b50>]
```




```
# loss function: mean squared error
def loss(y_true, y_pred):
  return ((y_true - y_pred)**2).mean()

print('loss for ground truth: {:.5f}'.format(loss(model(x, w_true), t)))
```

```
loss for ground truth: 0.61408
```

General linear model: second order polynomial. Parameters: w_0, w_1, w_2; basis functions:

- $\phi_0(x) = 1$,
- $\phi_1(x) = x$,
- $\phi_2(x) = x^2$

```
Phi = np.stack(([1]*N, x, x*x), axis=1)
print('Phi.shape:', Phi.shape)
```

```
Phi.shape: (100, 3)
```

```
# solution of general linear model
print('shape of matrix to be inverted:', (Phi.T @ Phi).shape)
w = np.linalg.inv(Phi.T @ Phi) @ Phi.T @ t
print(f'w={w}')
plot(x_plot, model(x_plot, w), 'g-')
plot(x_plot, model(x_plot, w_true), 'r-')
print(f'loss for predicted w: {loss(model(x, w), t):.5f}')
```

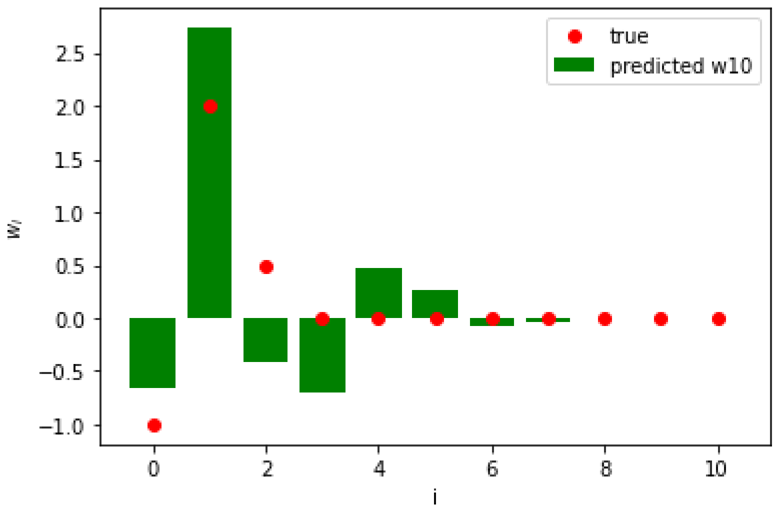
```
shape of matrix to be inverted: (3, 3)
w=[-0.94831804  2.00499009  0.47663723]
loss for predicted w: 0.61049
```

Overfitting

Fitting 10th order polynomial to noisy quadratic data.

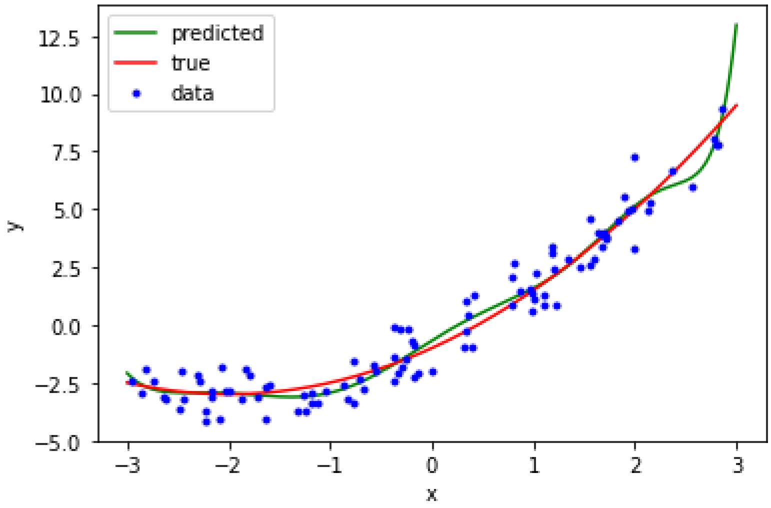
```
# new model: 10th order polynomial
def model10(x, w):
  return w[0] + w[1]*x + w[2]*x*x + w[3]*x**3 + w[4]*x**4 + w[5]*x**5 + w[6]*x**6 +

# solution of general linear model
Phi10 = np.stack(([1]*N, x, x*x, x**3, x**4, x**5, x**6, x**7, x**8, x**9, x**10),
w10 = np.linalg.inv(Phi10.T @ Phi10) @ Phi10.T @ t
print(f'w10={w10}')
print(f'loss for predicted w: {loss(model10(x, w10), t):.5f}')

# plot coefficients
bar(range(11), height=w10, bottom=0, color='green', label='predicted w10')
plot(w_true + [0]*8, 'ro', label='true')
xlabel('i')
ylabel('$w_i$')
legend();
```

```
w10=[-6.65680198e-01  2.73429426e+00 -4.07129776e-01 -6.92907759e-01
  4.76989510e-01  2.67913070e-01 -7.53244705e-02 -4.54093789e-02
  8.78241382e-04  2.65429227e-03  3.43640545e-04]
loss for predicted w: 0.55086
```




```
# plot prediction and ground truth (noisless) function
plot(x_plot, model10(x_plot, w10), 'g-', label='predicted')
plot(x_plot, model(x_plot, w_true), 'r-', label='true')
xlabel('x')
ylabel('y')
plot(x, t, 'bo', markersize=3, label='data')                    # data (blue)
legend()
```

```
<matplotlib.legend.Legend at 0x7efbf4221dd0>
```



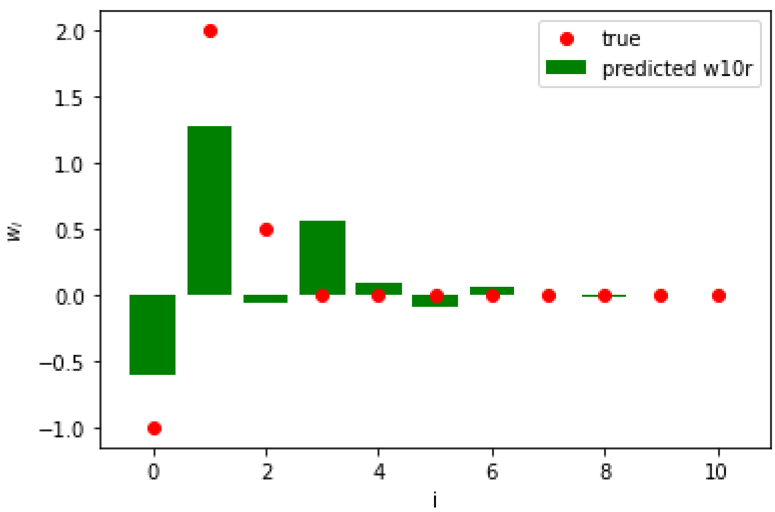
Regularization

Minimizing

$$\frac{1}{2}\sum_{n=1}^{N}(t - y(x; w))^2 + \frac{\lambda}{2}\sum_{d}|w_d|^2$$

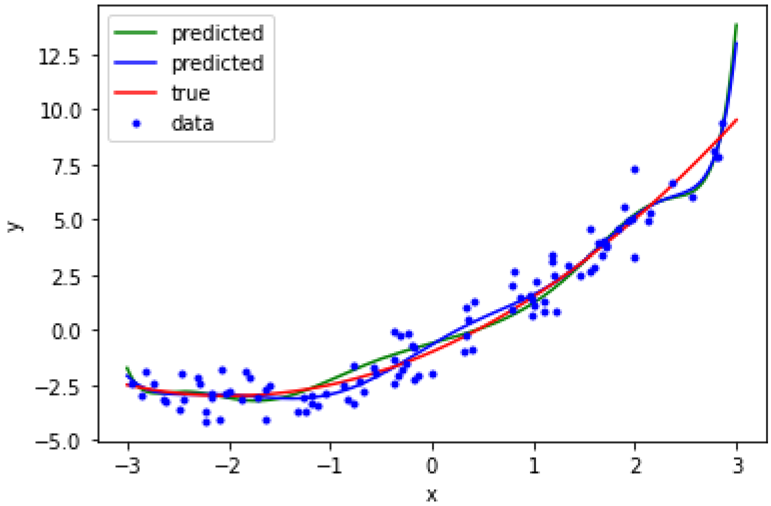
```
lamb = 10    # lambda is keyword in python
w10r = np.linalg.inv(lamb*np.identity(11) + Phi10.T @ Phi10) @ Phi10.T @ t
print(w10r)

# plot coefficients
bar(range(11), height=w10r, bottom=0, color='green', label='predicted w10r')
plot(w_true + [0]*8, 'ro', label='true')
xlabel('i')
ylabel('$w_i$')
legend();
```

```
[-6.08413465e-01  1.27848914e+00 -5.74148358e-02  5.68223294e-01
  8.88312325e-02 -9.23622942e-02  5.76712021e-02 -3.93652343e-03
 -1.74603274e-02  9.98325782e-04  1.22751010e-03]
```



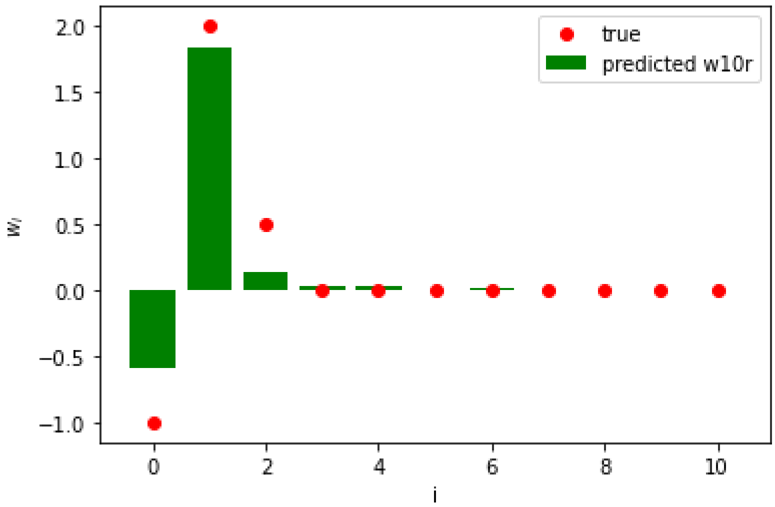
```
# plot prediction and ground truth (noiseless) function
plot(x_plot, model10(x_plot, w10r), 'g-', label='predicted')
plot(x_plot, model10(x_plot, w10), 'b-', label='predicted')
plot(x_plot, model(x_plot, w_true), 'r-', label='true')
xlabel('x')
ylabel('y')
plot(x, t, 'bo', markersize=3, label='data')                 # data (blue)
legend()
```

```
<matplotlib.legend.Legend at 0x7efbf411ecd0>
```



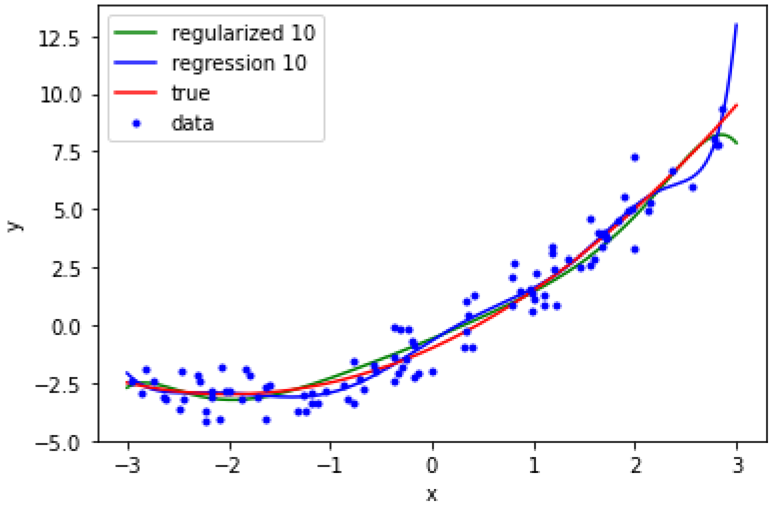

```
regularizer = np.diagflat([i**4 for i in range(0, 11)])
lamb = 10    # lambda is keyword in python
w10r = np.linalg.inv(lamb*regularizer + Phi10.T @ Phi10) @ Phi10.T @ t
print(w10r)

# plot coefficients
bar(range(11), height=w10r, bottom=0, color='green', label='predicted w10r')
plot(w_true + [0]*8, 'ro', label='true')
xlabel('i')
ylabel('$w_i$')
legend();
```

```
[-5.96205151e-01  1.83943821e+00  1.30617762e-01  2.91208727e-02
  2.37383012e-02  4.25116517e-03  8.68214544e-03 -4.06533595e-04
 -1.90275090e-04 -5.88132164e-05 -8.46652416e-05]
```



```
# plot prediction and ground truth (noiseless) function
plot(x_plot, model10(x_plot, w10r), 'g-', label='regularized 10')
plot(x_plot, model10(x_plot, w10), 'b-', label='regression 10')
plot(x_plot, model(x_plot, w_true), 'r-', label='true')
xlabel('x')
ylabel('y')
plot(x, t, 'bo', markersize=3, label='data')                    # data (blue)
legend()
```

```
<matplotlib.legend.Legend at 0x7efbf3fcfad0>
```



Same in sklearn:

```
from sklearn.linear_model import LinearRegression, Lasso
```

```
reg = LinearRegression(fit_intercept=False)
reg.fit(Phi10, t)
print(reg.coef_ - w10)
```

```
[ 3.25928173e-11 -4.83870721e-11 -3.41351392e-11  5.36118927e-11
  3.53156393e-11 -2.40147902e-11 -1.38012241e-11  4.10199652e-12
  2.10126014e-12 -2.30538678e-13 -1.08100161e-13]
```

Lasso (L1) regularization

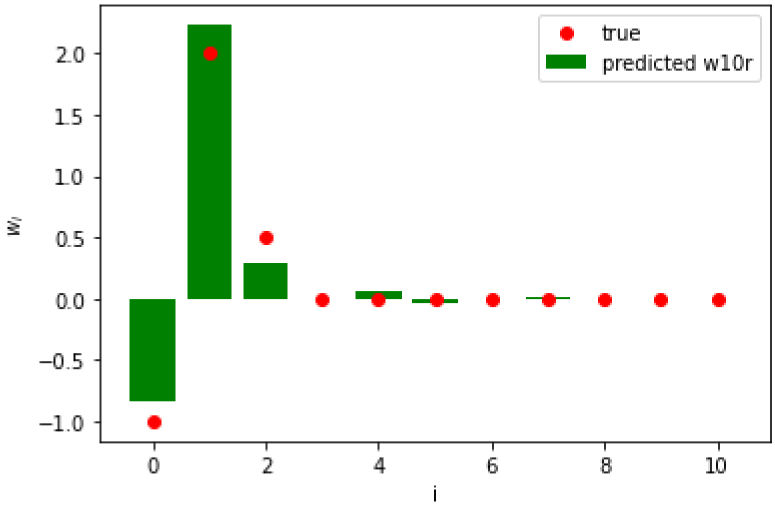
Minimizing

$$\frac{1}{2N}\sum_{n=1}^{N}(t-y(x;w))^2+\alpha\sum_{d}|w_d|$$

```
reg = Lasso(alpha=.01, fit_intercept=False, selection='random')
reg.fit(Phi10, t)
print(reg.coef_)
# plot coefficients
bar(range(len(reg.coef_)), height=reg.coef_, bottom=0, color='green', label='predic
```

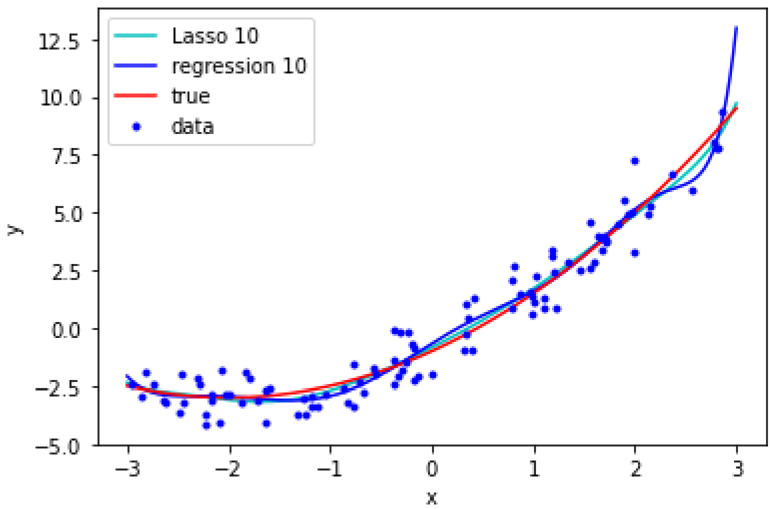
```
plot(w_true + [0]*8, 'ro', label='true')
xlabel('i')
ylabel('$w_i$')
legend();
```

```
[-8.42179410e-01  2.23048101e+00  2.82248574e-01  2.54077618e-03
  6.80078351e-02 -2.89951734e-02 -5.88934091e-03  3.94471991e-03
 -7.24052666e-04 -1.17242068e-04  9.33805160e-05]
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_coordinate_des
  positive)
```




```
# plot prediction and ground truth (noiseless) function
plot(x_plot, model10(x_plot, reg.coef_), 'c-', label='Lasso 10')
plot(x_plot, model10(x_plot, w10), 'b-', label='regression 10')
plot(x_plot, model(x_plot, w_true), 'r-', label='true')
xlabel('x')
ylabel('y')
plot(x, t, 'bo', markersize=3, label='data')                    # data (blue)
legend()
```

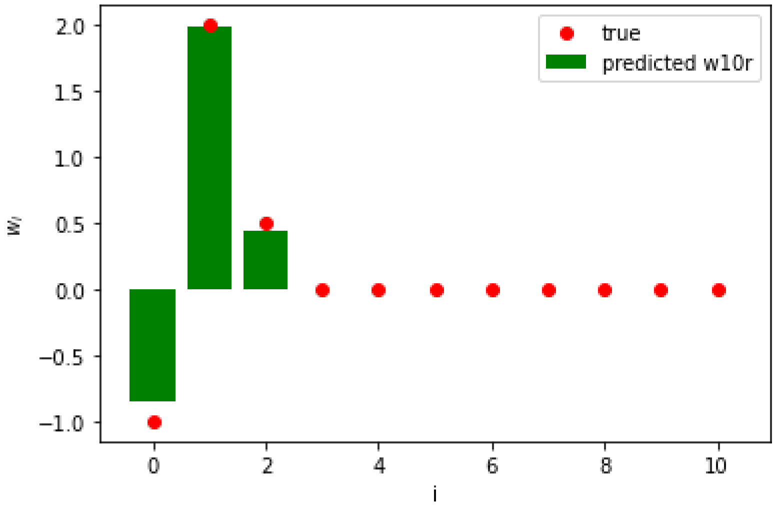
```
<matplotlib.legend.Legend at 0x7efbe15f3850>
```



Penalize high order terms (still Lasso)

```
reg_base = 4
Phi10x = np.stack([x**i_ / reg_base**i_ for i_ in range(11)], axis=1)
model10x = lambda x, w: sum([w[i_] * x**i_ / reg_base**i_ for i_ in range(11)])

reg = Lasso(alpha=0.01, fit_intercept=False, selection='random')
reg.fit(Phi10x, t)
orig_coeffs = [reg.coef_[i_] / reg_base**i_ for i_ in range(11)]
print(orig_coeffs)
# plot coefficients
bar(range(len(reg.coef_)), height=orig_coeffs, bottom=0, color='green', label='pred
plot(w_true + [0]*8, 'ro', label='true')
xlabel('i')
ylabel('$w_i$')
legend();
```

```
[-0.8497892670998304, 1.9827585697294576, 0.4428754742497901, -0.0, 0.0, -0
```




```
# plot prediction and ground truth (noiseless) function
plot(x_plot, model10x(x_plot, reg.coef_), 'c-', label='Lasso 10')
plot(x_plot, model10(x_plot, w10), 'b-', label='regression 10')
plot(x_plot, model(x_plot, w_true), 'r-', label='true')
xlabel('x')
ylabel('y')
plot(x, t, 'bo', markersize=3, label='data')                    # data (blue)
legend()
```

```
<matplotlib.legend.Legend at 0x7efbe1493890>
```