hw1_solutions

January 9, 2024

Problem 1 Without taking derivatives, prove that the median minimizes the average absolute loss, i.e.

$$Med(X) \in \arg\min_{a\in\mathbb{R}} \mathbb{E}|X-a|.$$

[This implies that the conditional median $Med(Y|X)$ minimizes $\mathbb{E}|Y-g(X)|$ over functions g.] *Hint.* Show that

$$\mathbb{E}|X-a| - \mathbb{E}|X-Med(X)| = Med(X) - a - 2\int_a^{Med(X)} F(x)\,dx.$$

Solution Denote $m = Med(X)$ and $f(a) = \mathbb{E}|X-a|$. We have

$$f(a) = \mathbb{E}|X-a|(1(X\le a)+1(X>a)) = \mathbb{E}(a-X)1(X\le a) + \mathbb{E}(X-a)1(X>a) \tag{1}$$

$$= aF(a) - a(1-F(a)) - \int_{-\infty}^{a} x\,dF(x) + \int_a^{\infty} x\,dF(x). \tag{2}$$

Therefore,

$$f(a) - f(m) = m - a - 2\int_a^m F(x)\,dx$$

Suppose $a \le m$. Then $F(x) \le F(m) = 1/2$ for all $x \in [a,m]$ and hence

$$f(a) - f(m) \ge m - a - 2\cdot\frac{1}{2}\cdot(m-a) = 0.$$

The case $a > m$ is similar. This proves that $f(a) - f(m) \ge 0$ for all $a \in \mathbb{R}$, and hence m is a global minimum of f.

Problem 2 Suppose the true model is $y = f(x) + \varepsilon$ with $f(x) = \beta x$, where $x \sim N(0, \sigma_X^2)$, x and ε are independent, $\mathbb{E}(\varepsilon) = 0$, and $\mathbb{E}(\varepsilon^2) = \sigma^2$. Calculate the out-of-sample risk (average squared loss) of two predictors $\tilde{f}(x) = \bar{y} = \frac{1}{n}\sum_{i=1}^n y_i$ (regression on a constant) and $\hat{f}(x) = x'\hat{\beta}$ with $\hat{\beta} = \sum_{i=1}^n x_i y_i / \sum_{i=1}^n x_i^2$ (regression on x). When is the risk of regression on a constant lower than the risk of regression on x? Explain. *Hints*: the mean of the inverse χ^2 distribution with n degrees of freedom is $n-2$. The out-of-sample risk of a predictor $\hat{f}$ is

$$R(\hat{f}) = \mathbb{E}(\hat{f}(x_0) - f(x_0))^2,$$

where $\hat{f}$ is calculated from the data $(x_i, y_i)_{i=1}^n$, while x_0 is a draw from x that is independent of the data; the expectation is taken w.r.t. both the data and x_0. Your answers will depend on n, β, σ_x^2, and σ only.

Solution Let x_0 be a draw from x independent of the sample $(x_i, y_i)_{i=1}^n$. The risk of the regression on a constant is

$$\mathbb{E}(\tilde{f}(x_0) - f(x_0))^2 = \mathbb{E}(\bar{y} - \beta x_0)^2 = \mathbb{E}\bar{y}^2 + \mathbb{E}(\beta x_0)^2 - 2\mathbb{E}\bar{y}\mathbb{E}\beta x_0 \tag{3}$$

$$= \frac{\beta^2\sigma_x^2 + \sigma^2}{n} + \beta^2\sigma_x^2 = \left(1 + \frac{1}{n}\right)\beta^2\sigma_x^2 + \frac{\sigma^2}{n}. \tag{4}$$

The risk of the regression on x is

$$\mathbb{E}(\hat{f}(x_0) - f(x_0))^2 = \mathbb{E}(\hat{\beta}x_0 - \beta x_0)^2 = \sigma_x^2\mathbb{E}\left(\frac{\sum_i x_i\varepsilon_i}{\sum_i x_i^2}\right)^2 \tag{5}$$

$$= \sigma_x^2\mathbb{E}\left(\frac{\sum_i x_i^2\varepsilon_i^2 + \sum_{i\neq j} x_i x_j \varepsilon_i\varepsilon_j}{(\sum_i x_i^2)^2}\right) = \sigma_x^2\mathbb{E}\left(\frac{\sigma^2\sum_i x_i^2}{(\sum_i x_i^2)^2}\right) \tag{6}$$

$$= \sigma^2\mathbb{E}\left(\frac{1}{\sum_i (x_i/\sigma_x)^2}\right) = \frac{\sigma^2}{n-2}, \tag{7}$$

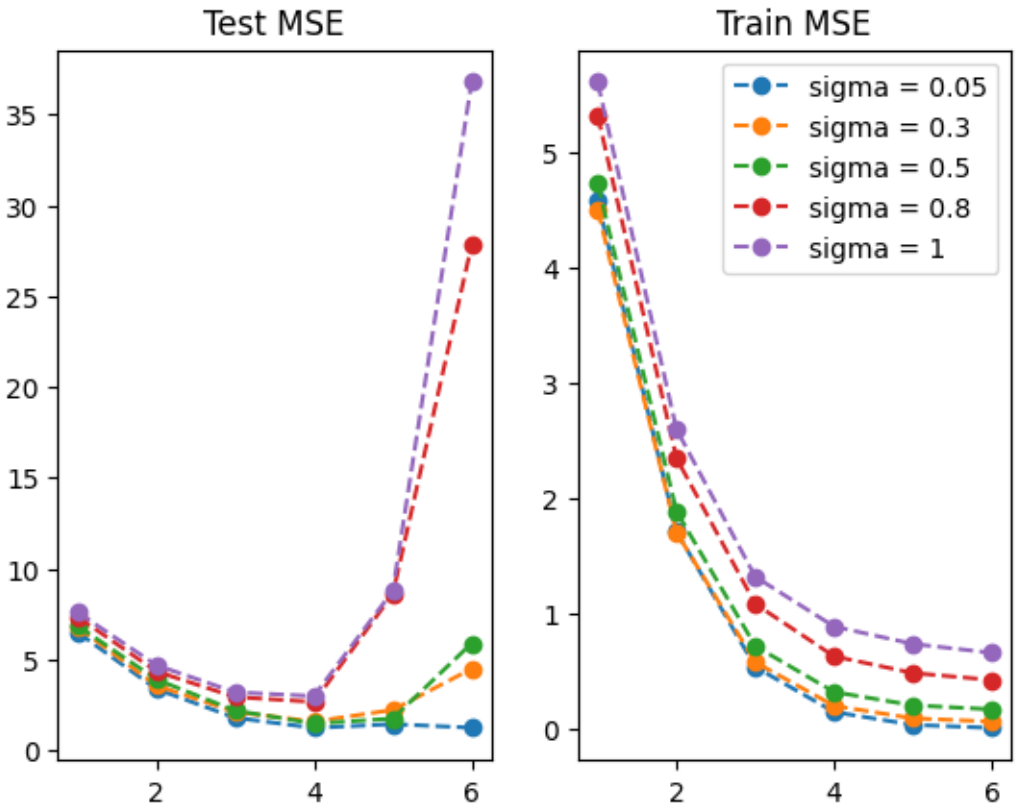
where the last equality uses the fact that the mean of the inverse χ^2 distribution with n degrees of freedom is $n-2$. We see that when β or σ_x^2 are very low for the given sample size n (i.e. x is a not an important determinant of y), the risk of regression on a constant is lower.

Problem 3 Suppose $y = \sum_{j=1}^{25}\beta_j x^j + \varepsilon$, where $x \sim N(0,1)$ is independent of $\varepsilon \sim N(0,\sigma^2)$, and $\beta_j = 0.5 + 0.5j/25$. Run a Monte Carlo simulation to estimate (as accurately as possible) the train and test MSE of a linear regression on the first p features $x, \ldots, x^p$ with the sample size $n = 20$. For each $\sigma \in (0.05, 0.3, 0.5, 0.8, 1)$, plot the test MSE as a function of p (take the maximum p small enough to see a U-shaped curve). Do the same for the train MSE. What is the optimal p for each σ? Explain this relationship.

[1]:
```
#### SOLUTION TO PROBLEM 3 ####

from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import numpy as np
from matplotlib import pyplot as plt

# parameters
n = 20
d = 25
d_range = range(1, 7, 1)
sigma_range = [0.05, 0.3, 0.5, 0.8, 1]
sigma_x = 1
n_test = 100
n_sims = 500
np.random.seed(5)
beta = np.linspace(0.5, 1, d)


# beta = np.random.randn(d)
```

```
# beta /= np.linalg.norm(beta)

def generate_data_polynomial(n, d, sigma, sigma_x, beta):
    x_base = sigma_x * np.random.rand(n)
    X = [x_base ** i for i in range(1, d + 1)]
    X = np.column_stack(X)
    eps = sigma * np.random.randn(n)
    Y = X @ beta + eps
    return X, Y


model = LinearRegression()
mse_test, mse_train = dict(), dict()

for d_fit in d_range:
    for sigma in sigma_range:
        mse_test[(d_fit, sigma)] = np.empty(n_sims)
        mse_train[(d_fit, sigma)] = np.empty(n_sims)

for s in range(n_sims):
    for sigma in sigma_range:
        X_train, Y_train = generate_data_polynomial(n, d, sigma, sigma_x, beta)
        for d_fit in d_range:
            model.fit(X_train[:, :d_fit], Y_train)
            X_test, Y_test = generate_data_polynomial(n_test, d, sigma,␣
 ↪sigma_x, beta)
            Y_pred = model.predict(X_test[:, :d_fit])
            mse_test[(d_fit, sigma)][s] = mean_squared_error(Y_test, Y_pred)
            Y_pred = model.predict(X_train[:, :d_fit])
            mse_train[(d_fit, sigma)][s] = mean_squared_error(Y_train, Y_pred)

fig, (ax_test, ax_train) = plt.subplots(1, 2)
for sigma in sigma_range:
    for d_fit in d_range:
        mse_test[(d_fit, sigma)] = np.mean(mse_test[(d_fit, sigma)])
        mse_train[(d_fit, sigma)] = np.mean(mse_train[(d_fit, sigma)])
        # print('d =', d_fit, ', sigma =', sigma, ', test MSE =',␣
 ↪mse_test[(d_fit, sigma)])
    ax_test.plot(d_range, [mse_test[(d_fit, sigma)] for d_fit in d_range],␣
 ↪linestyle='--', marker='o',
                 label='sigma = ' + str(sigma))
    ax_train.plot(d_range, [mse_train[(d_fit, sigma)] for d_fit in d_range],␣
 ↪linestyle='--', marker='o',
                  label='sigma = ' + str(sigma))
ax_test.title.set_text('Test MSE')
ax_train.title.set_text('Train MSE')
plt.legend()
```

```
plt.show()
```



Problem 4 Let $y = x - 2x^2 + \varepsilon$, where x and ε are independent $N(0, 1)$ random variables.

(a) Set a random seed, and compute the LOOCV errors from fitting the following four models using least squares on $n = 100$ points:

$$y = \beta_0 + \beta_1 x + u, \tag{8}$$
$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + u, \tag{9}$$
$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + u, \tag{10}$$
$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \beta_4 x^4 + u. \tag{11}$$
$$\tag{12}$$

Also create a scatterplot of the data.

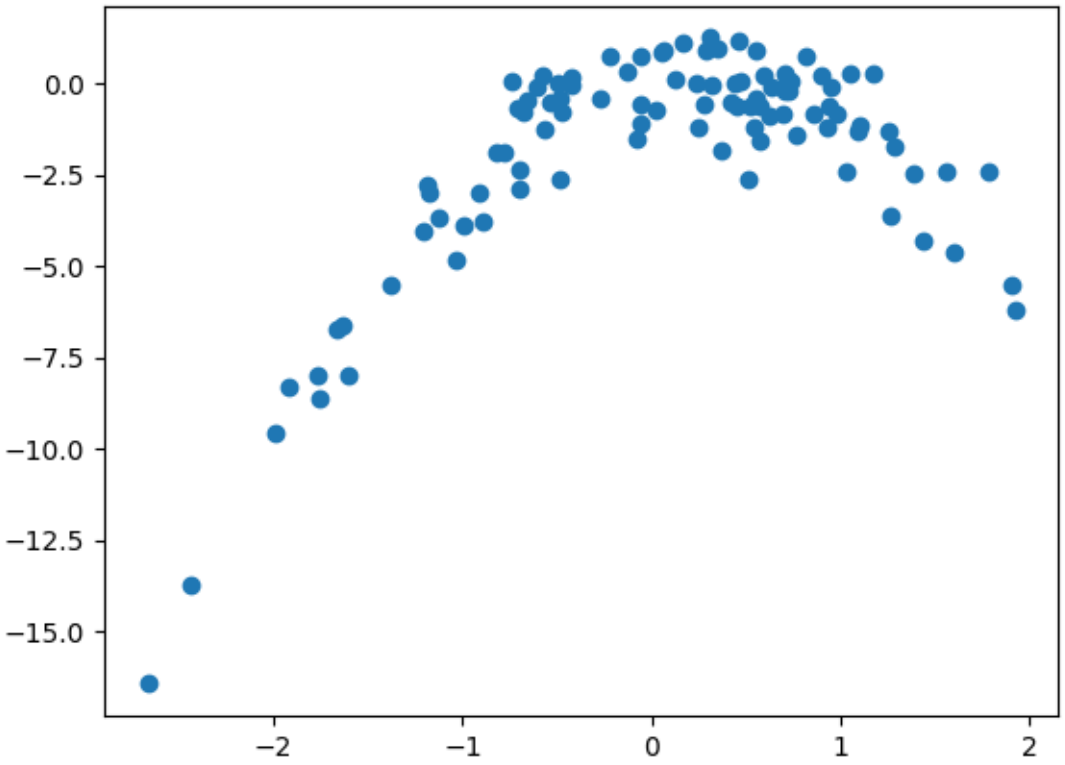
(b) Repeat (a) using another random seed, and report your results. Are your results the same as what you got in (a)?

(c) Which of the models in (a) had the smallest LOOCV error? Is this what you expected? Explain your answer.

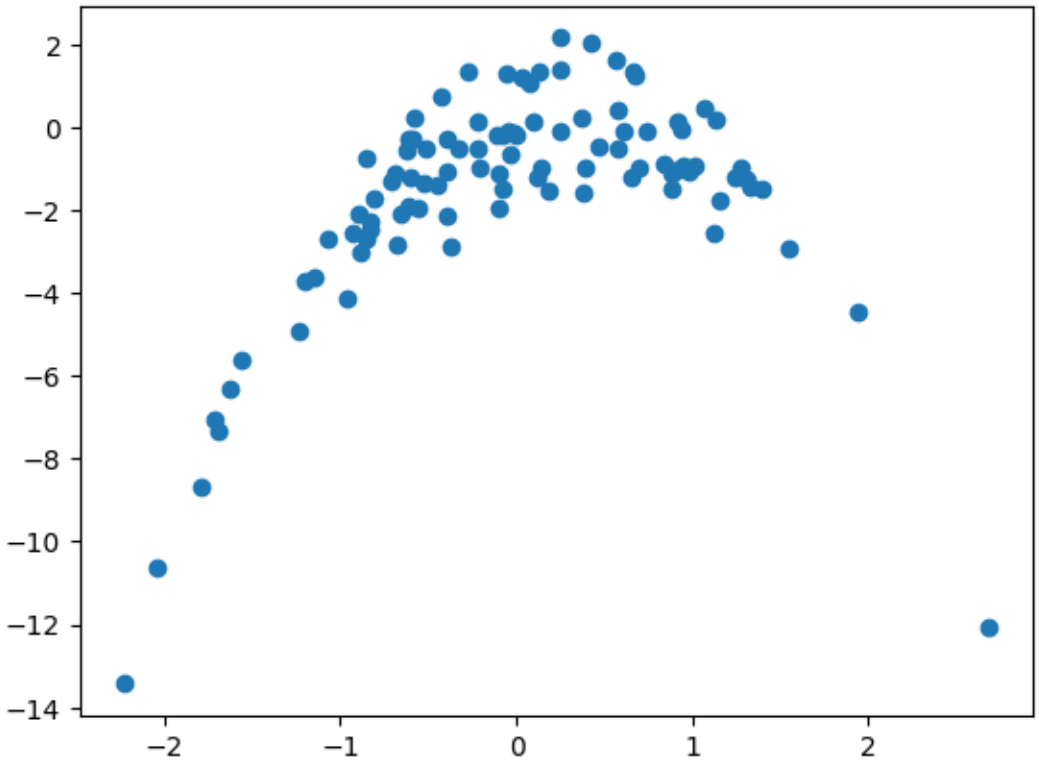
(d) Comment on the statistical significance of the coefficient estimates that results from fitting each of the models in (a) using least squares. Do these results agree with the conclusions drawn based on the cross-validation results?

[2]:
```
# (a), (b), (c), (d) code:

import numpy as np
from matplotlib import pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_validate
from sklearn.metrics import mean_squared_error
from scipy import stats

n = 100
model = LinearRegression()

for seed in [1,2]:
    np.random.default_rng(seed)
    print(">>>>>> Random Seed: " + str(seed))
    x = np.random.normal(size=n)
    y = x - 2 * x**2 + np.random.normal(size=n)
    plt.scatter(x, y)
    plt.show()
    for p in range(1,5):
        xp = np.transpose([x**i for i in range(0,p+1)])
        # compute LOOCV MSE:
        cv =␣
 ↪cross_validate(model,xp,y,cv=n,scoring='neg_mean_squared_error',return_train_score=True)
        # calculate standard errors:
        model.fit(xp,y)
        mse = mean_squared_error(y, model.predict(xp))
        se = np.sqrt(np.diag(np.linalg.inv(np.dot(xp.T, xp))) * mse)
        # calculate p-values:
        t_stats = model.coef_ / se
        p_values = [2 * (1 - stats.t.cdf(np.abs(t), n-p)) for t in t_stats]
        print('p =', p, ', test MSE (CV) =', -np.mean(cv['test_score']))
        print('p-values = ' + str([f"{p_val:.2f}" for p_val in p_values]))


# (b) The results are different across seeds because of randomness in the data,␣
 ↪and therefore LOOCV-estimated MSE.

# (c) The lowest cross-validated MSE is achieved for p=2, which is expected␣
 ↪given that the true model has p=2.

# (d) Calculating the p-values (see codes), we see that only the coefficients␣
 ↪on x and x^2 (whenever included in the specification) are significant at 1%.␣
 ↪This agrees with the fact that the true model only includes x and x^2␣
 ↪(without an intercept).
```

```
>>>>>> Random Seed: 1
```




```
p = 1 , test MSE (CV) = 7.0271527478964355
p-values = ['1.00', '0.00']
p = 2 , test MSE (CV) = 0.9187201723378083
p-values = ['1.00', '0.00', '0.00']
p = 3 , test MSE (CV) = 0.9379653245993989
p-values = ['1.00', '0.00', '0.00', '0.76']
p = 4 , test MSE (CV) = 0.9540037749953731
p-values = ['1.00', '0.00', '0.00', '0.53', '0.55']
>>>>>> Random Seed: 2
```




```
p = 1 , test MSE (CV) = 6.79712986311981
p-values = ['1.00', '0.00']
p = 2 , test MSE (CV) = 0.9138739200730818
p-values = ['1.00', '0.00', '0.00']
p = 3 , test MSE (CV) = 1.0714000706918358
p-values = ['1.00', '0.00', '0.00', '0.81']
p = 4 , test MSE (CV) = 0.8590197042065492
p-values = ['1.00', '0.00', '0.00', '0.25', '0.02']
```