Benjamin Ye
CS/CNS/EE 156a: Learning Systems (Fall 2023)
November 13, 2023

Homework 7

Problem	Answer
1	[d]
2	[e]
3	[d]
4	[d]
5	[b]
6	[d]
7	[c]
8	
9	
10	

Validation

In the following problems, use the data provided in the files `in.dta` and `out.dta` for Homework 6. We are going to apply linear regression with a nonlinear transformation for classification (without regularization). The nonlinear transformation is given by ϕ_0 through ϕ_7 which transform (x_1, x_2) into

$$1 \quad x_1 \quad x_2 \quad x_1^2 \quad x_2^2 \quad x_1x_2 \quad |x_1 - x_2| \quad |x_1 + x_2|$$

To illustrate how taking out points for validation affects the performance, we will consider the hypotheses trained on $\mathcal{D}_{\text{train}}$ (without restoring the full $\mathcal{D}$ for training after validation is done).

1. Split `in.dta` into training (first 25 examples) and validation (last 10 examples). Train on the 25 examples only, using the validation set of 10 examples to select between five models that apply linear regression to ϕ_0 through ϕ_k, with $k = 3, 4, 5, 6, 7$. For which model is the classification error on the validation set smallest?

   Answer: [d] $k = 6$

2. Evaluate the out-of-sample classification error using `out.dta` on the 5 models to see how well the validation set predicted the best of the 5 models. For which model is the out-of-sample classification error smallest?

   Answer: [e] $k = 7$

3. Reverse the role of training and validation sets; now training with the last 10 examples and validating with the first 25 examples. For which model is the classification error on the validation set smallest?

   Answer: [d] $k = 6$

4. Once again, evaluate the out-of-sample classification error using `out.dta` on the 5 models to see how well the validation set predicted the best of the 5 models. For which model is the out-of-sample classification error smallest?

   Answer: [d] $k = 6$

5. What values are closest in the Euclidean distance to the out-of-sample classification error obtained for the model chosen in Problems 1 and 3, respectively?

   Answer: [b] 0.1, 0.2

See the following pages for the linear regression results and the Python 3 source code.

The output used to answer Problems 1–5 is

```
[HW7 P1-5]
Linear regression statistics for 25:10 split:
  k=3, E_in_test=0.440, E_in_validate=0.300, E_out=0.420
  k=4, E_in_test=0.320, E_in_validate=0.500, E_out=0.416
  k=5, E_in_test=0.080, E_in_validate=0.200, E_out=0.188
  k=6, E_in_test=0.040, E_in_validate=0.000, E_out=0.084
  k=7, E_in_test=0.040, E_in_validate=0.100, E_out=0.072
Linear regression statistics for 10:25 split:
  k=3, E_in_test=0.400, E_in_validate=0.280, E_out=0.396
  k=4, E_in_test=0.300, E_in_validate=0.360, E_out=0.388
  k=5, E_in_test=0.200, E_in_validate=0.200, E_out=0.284
  k=6, E_in_test=0.000, E_in_validate=0.080, E_out=0.192
  k=7, E_in_test=0.000, E_in_validate=0.120, E_out=0.196
```

The Python 3 source code is below and continues on the following pages:

```python
import pathlib

import numpy as np
import requests

CWD = pathlib.Path(__file__).resolve().parent
DATA_DIR = (CWD / "../data").resolve()

def generate_data(
        N, f, d=2, lb=-1.0, ub=1.0, *, bias=False, rng=None, seed=None):
    if rng is None:
        rng = np.random.default_rng(seed)
    x = rng.uniform(lb, ub, (N, d))
    if bias:
        x = np.hstack((np.ones((N, 1)), x))
    return x, f(x)

def validate_binary(w, x, y):
    return np.count_nonzero(np.sign(x @ w) != y, axis=0) / x.shape[0]
```

```python
def linear_regression(
        N=None, f=None, vf=None, *, x=None, y=None, transform=None, noise=None,
        regularization=None, N_test=1_000, x_test=None, y_test=None,
        x_validate=None, y_validate=None, rng=None, seed=None, hyp=False,
        **kwargs):

    if rng is None:
        rng = np.random.default_rng(seed)
    if x is None or y is None:
        if x is None:
            x, y = generate_data(N, f, bias=True, rng=rng)
        else:
            N = x.shape[0]
            y = f(x)
    else:
        N = x.shape[0]
    if transform:
        x = transform(x)
    if noise:
        i = rng.choice(N, round(noise[0] * N), False)
        y[i] = noise[1](y[i])

    if regularization is None:
        w = np.linalg.pinv(x) @ y
    elif regularization == "weight_decay":
        w = np.linalg.inv(
            x.T @ x + kwargs["wd_lambda"] * np.eye(x.shape[1], dtype=float)
        ) @ x.T @ y

    if x_test is None or y_test is None:
        if x_test is None:
            x_test, y_test = generate_data(N_test, f, bias=True, rng=rng)
        else:
            N_test = x_test.shape[0]
            y_test = f(x_test)
    else:
        N_test = x_test.shape[0]
    if transform:
        x_test = transform(x_test)
    if noise:
        i = rng.choice(N_test, round(noise[0] * N_test), False)
        y_test[i] = noise[1](y_test[i])

    if x_validate is None or y_validate is None:
        return (w, vf(w, x, y), vf(w, x_test, y_test))[1 - hyp:]
    else:
        N_validate = len(y_validate)
        if transform:
            x_validate = transform(x_validate)
        if noise:
            i = rng.choice(N_validate, round(noise[0] * N_validate), False)
            y_validate[i] = noise[1](y_validate[i])
        return (w, (vf(w, x, y), vf(w, x_validate, y_validate)),
                vf(w, x_test, y_test))[1 - hyp:]

if __name__ == "__main__":
    DATA_DIR.mkdir(exist_ok=True)
    raw_data = {}
    for prefix in ["in", "out"]:
        if not (DATA_DIR / f"{prefix}.dta").exists():
            r = requests.get(f"http://work.caltech.edu/data/{prefix}.dta")
            with open(DATA_DIR / f"{prefix}.dta", "wb") as f:
                f.write(r.content)
        raw_data[prefix] = np.loadtxt(DATA_DIR / f"{prefix}.dta")

    print("\n[HW7 P1-5]")
    ns = (25, len(raw_data["in"]) - 25)
    data = np.array_split(raw_data["in"], (ns[0],))
    transform_funcs = (
        lambda x: np.ones((len(x), 1), dtype=float),
        lambda x: x,
        lambda x: x[:, :1] ** 2,
        lambda x: x[:, 1:] ** 2,
        lambda x: np.prod(x, axis=1, keepdims=True),
        lambda x: np.abs(x[:, :1] - x[:, 1:]),
        lambda x: np.abs(x[:, :1] + x[:, 1:])
    )
    for i in range(2):
        print(f"Linear regression statistics for {ns[i]}:{ns[1 - i]} split:")
        for k in np.arange(3, 8):
            w, E_in, E_out = linear_regression(
                vf=validate_binary,
                x=data[i][:, :-1],
                y=data[i][:, -1],
                transform=lambda x: np.hstack(
                    tuple(f(x) for f in transform_funcs[:k])
                ),
                x_test=raw_data["out"][:, :-1],
                y_test=raw_data["out"][:, -1],
                x_validate=data[1 - i][:, :-1],
                y_validate=data[1 - i][:, -1],
                hyp=True
            )
            print(f"  {k=}, E_in_test={E_in[0]:.3f}, "
                  f"E_in_validate={E_in[1]:.3f}, {E_out=:.3f}")
```

Validation Bias

6. Let e_1 and e_2 be independent random variables, distributed uniformly over the interval $[0,\ 1]$. Let $\mathrm{e} = \min(\mathrm{e}_1,\ \mathrm{e}_2)$. The expected values of e_1, e_2, and e are closest to

Answer: [d] 0.5, 0.5, 0.4

For a continuous random variable X over the interval $[a,\ b]$, the expected value is given by

$$\mathrm{E}[X] = \int_a^b x f(x)\,dx$$

For e_1 and e_2, the expected values are the same since they share the same distribution:

$$\mathrm{e}_1 = \mathrm{e}_2 = E[X] = \frac{1}{b-a}\int_a^b x\,dx = \frac{1}{1-0}\int_0^1 x\,dx = \frac{1}{2}$$

For $\mathrm{e} = \min(\mathrm{e}_1,\ \mathrm{e}_2)$,

$$\begin{aligned}\mathrm{e} = E[\min(X_i)] &= \int_a^b P(x < \min(X_i))\,dx = \int_{a_1}^{b_1} P(x_1 < X_1)\,dx_1 \int_{a_2}^{b_2} P(x_2 < X_2)\,dx_2 \\ &= \int_0^1 (1 - x_1)\,dx_1 \int_0^1 (1 - x_2)\,dx_2 = \int_0^1 (1-x)^2\,dx = \frac{1}{3}\end{aligned}$$

Therefore, the expected values of e_1, e_2, and e are 0.5, 0.5, and $0.\overline{3}$, respectively.

This is supported by a simulation with 10 million randomly generated points for both X_1 and X_2:

```
[HW7 P6]
Expected values for continuous uniform distribution:
  e_1=0.500, e_2=0.500, e=0.333
```

The Python 3 source code is below.

```python
import numpy as np

if __name__ == "__main__":
    rng = np.random.default_rng()
    x = rng.uniform(size=(10_000_000, 2))
    e_1, e_2 = x.mean(axis=0)
    e = x.min(axis=1).mean()
    print("\n[HW7 P6]\nExpected values for continuous uniform distribution:",
          f"  {e_1=:.3f}, {e_2=:.3f}, {e=:.3f}", sep="\n")
```

Cross Validation

7. You are given the data points (x, y): $(-1, 0)$, $(\rho, 1)$, $(1, 0)$, $\rho \geq 0$, and a choice between two models: constant $\{h_0(x) = b\}$ and linear $\{h_1(x) = ax + b\}$. For which value of ρ would the two models be tied using leave-one-out cross-validation with the squared error measure?

Answer: [c] $\sqrt{9 + 4\sqrt{6}}$

With the leave-one-out cross-validation approach, one of the three data points will be discarded. Let us take generally $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$ to be the two data points used in the model.

For the constant model, the best fit is the constant line between p_1 and p_2, or

$$b_{\text{const}} = \frac{y_1 + y_2}{2}$$

For the linear model, the line of best fit gives

$$a = \frac{y_1 - y_2}{x_1 - x_2}, \quad b_{\text{lin}} = \frac{x_1 y_2 - x_2 y_1}{x_1 - x_2}$$

(as derived in Homework 4 Problem 7).

With the three possible unique combinations of points, we have

$$p_1 = (-1, 0),\ p_2 = (\rho, 1) \longrightarrow b_{\text{const}} = \frac{1}{2},\ a = \frac{1}{1+\rho},\ b_{\text{lin}} = \frac{1}{1+\rho}$$

$$p_1 = (\rho, 1),\ p_2 = (1, 0) \longrightarrow b_{\text{const}} = \frac{1}{2},\ a = \frac{1}{\rho - 1},\ b_{\text{lin}} = \frac{1}{1-\rho}$$

$$p_1 = (-1, 0),\ p_2 = (1, 0) \longrightarrow b_{\text{const}} = 0,\ a = 0,\ b_{\text{lin}} = 0$$

The squared errors for the two models are

$$E_{\text{const}} = \frac{1}{3}\left[\left(0 - \frac{1}{2}\right)^2 + \left(0 - \frac{1}{2}\right)^2 + (1-0)^2\right] = \frac{1}{2}$$

$$\begin{aligned} E_{\text{lin}} &= \frac{1}{3}\left[\left(0 - \frac{1}{1+\rho} \times 1 - \frac{1}{1+\rho}\right)^2 + \left(0 - \frac{1}{\rho - 1} \times -1 - \frac{1}{1-\rho}\right)^2 + (1 - 0 \times \rho - 0)^2\right] \\ &= \frac{1}{3}\left[\frac{4}{(1+\rho)^2} + \frac{4}{(1-\rho)^2} + 1\right] \end{aligned}$$

Solving for ρ by setting $E_{\text{const}} = E_{\text{lin}}$,

$$\frac{1}{2} = \frac{1}{3}\left[\frac{4}{(1+\rho)^2} + \frac{4}{(1-\rho)^2} + 1\right]$$

$$1 = 8\left[\frac{1}{(1+\rho)^2} + \frac{1}{(1-\rho)^2}\right] = 8\left[\frac{(1+\rho)^2 + (1-\rho)^2}{(1+\rho)^2(1-\rho)^2}\right] = 16\left[\frac{1+\rho^2}{1 - 2\rho^2 + \rho^4}\right]$$

$$\rho^4 - 18\rho^2 = 15 \longrightarrow \rho = \sqrt{9 + 4\sqrt{6}}$$

PLA vs. SVM

Notice: Quadratic programming packages sometimes need tweaking and have numerical issues, and this is characteristic of packages you will use in practical ML situations. Your understanding of support vectors will help you get to the correct answers.

In the following problems, we compare PLA to SVM with hard margin[1] on linearly separable data sets. For each run, you will create your own target function f and data set $\mathcal{D}$. Take $d = 22$ and choose a random line in the plane as your target function f (do this by taking two random, uniformly distributed points on $[-1, 1] \times [-1, 1]$ and taking the line passing through them), where one side of the line maps to $+1$ and the other maps to -1. Choose the inputs $\mathbf{x}_n$ of the data set as random points in $\mathcal{X} = [-1, 1] \times [-1, 1]$, and evaluate the target function on each $\mathbf{x}_n$ to get the corresponding output y_n. If all data points are on one side of the line, discard the run and start a new run.

Start PLA with the all-zero vector and pick the misclassified point for each PLA iteration at random. Run PLA to find the final hypothesis g_{PLA} and measure the disagreement between f and g_{PLA} as $\mathbb{P}[f(\mathbf{x}) \neq g_{\text{PLA}}(\mathbf{x})]$ (you can either calculate this exactly, or approximate it by generating a sufficiently large, separate set of points to evaluate it). Now, run SVM on the same data to find the final hypothesis g_{SVM} by solving

$$\min_{\mathbf{w},b} \frac{1}{2}\mathbf{w}^\top\mathbf{w} \quad \text{s.t.} \quad \mathrm{y_n}(\mathbf{w}^\top\mathbf{x}_n + b) \geq 1$$

using quadratic programming on the primal[2] or the dual problem or using an SVM package. Measure the disagreement between f and g_{SVM} as $\mathbb{P}[f(\mathbf{x}) \neq g_{\text{SVM}}(\mathbf{x})]$, and count the number of support vectors you get in each run.

[1] For hard margin in SVM packages, set $C \to \infty$ and choose the “linear” kernel.
[2] Primal problem is the original formulation in slide 11 of Lecture 14.

8. For $N = 10$, repeat the above experiment for 1,000 runs. How often is g_{SVM} better than g_{PLA} in approximating f? The percent of time is closest to:

   Answer: [c] 60%

9. For $N = 100$, repeat the above experiment for 1,000 runs. How often is g_{SVM} better than g_{PLA} in approximating f? The percentage of time is closest to:

   Answer: [d] 70%

10. For the case $N = 100$, which of the following is closest to the average number of support vectors of g_{SVM} (averaged over the 1,000 runs)?

    Answer: [b] 3

See the following pages for the perceptron and support vector machine results and the Python 3 source code.

For the perceptron, the target function is a simple line $f(\mathbf{x}) = mx + b$, where x is the x-coordinate of point $\mathbf{x}_n$ in data set $\mathcal{D}$, and the misclassification rate is estimated by using a test data set with $N = 100{,}000$. The sample output used to answer Problems 8–10 is

```
[HW7 P8-10]
PLA vs. SVM with hard margins over 1,000 runs:
  N=10, prob_svm=0.645, N_sv_avg=2.894
  N=100, prob_svm=0.726, N_sv_avg=2.999
```

The Python 3 source code is below and continues on the following pages:

```python
import numpy as np
from sklearn import svm

def generate_data(
        N, f, d=2, lb=-1.0, ub=1.0, *, bias=False, rng=None, seed=None):
    if rng is None:
        rng = np.random.default_rng(seed)
    x = rng.uniform(lb, ub, (N, d))
    if bias:
        x = np.hstack((np.ones((N, 1)), x))
    return x, f(x)

def target_function_random_line(*, rng=None, seed=None):
    if rng is None:
        rng = np.random.default_rng(seed)
    line = rng.uniform(-1, 1, (2, 2))
    return lambda x: np.sign(
        x[:, -1] - line[0, 1]
        - np.divide(*(line[1] - line[0])[::-1]) * (x[:, -2] - line[0, 0])
    )

def validate_binary(w, x, y):
    return np.count_nonzero(np.sign(x @ w) != y, axis=0) / x.shape[0]
```

```
def perceptron(
        N=None, f=None, vf=None, *, w=None, x=None, y=None, N_test=1_000,
        x_test=None, y_test=None, rng=None, seed=None, hyp=False):
    if rng is None:
        rng = np.random.default_rng(seed)
    if y is None:
        if x is None:
            x, y = generate_data(N, f, bias=True, rng=rng)
        else:
            y = f(x)
    if w is None:
        w = np.zeros(x.shape[1], dtype=float)

    iters = 0
    while True:
        wrong = np.argwhere(np.sign(x @ w) != y)[:, 0]
        if wrong.size == 0:
            break
        i = np.random.choice(wrong)
        w += y[i] * x[i]
        iters += 1

    if vf is None:
         return (w, iters)[1 - hyp:]

    if x_test is None or y_test is None:
        if x_test is None:
            x_test, y_test = generate_data(N_test, f, bias=True, rng=rng)
        else:
            y_test = f(x_test)
    return (w, iters, vf(w, x_test, y_test))[1 - hyp:]
```

```
def support_vector_machine(
        N=None, f=None, vf=None, *, x=None, y=None, N_test=1_000, x_test=None,
        y_test=None, clf=None, rng=None, seed=None, hyp=False, **kwargs):
    if rng is None:
        rng = np.random.default_rng(seed)
    if y is None:
        if x is None:
            x, y = generate_data(N, f, bias=True, rng=rng)
        else:
            y = f(x)

    if clf is None:
        clf = svm.SVC(**kwargs)
    clf.fit(x[:, 1:], y)
    w = np.concatenate((clf.intercept_, clf.coef_[0]))
    N_sv = clf.n_support_.sum()

    if vf is None:
         return (w, N_sv)[1 - hyp:]

    if x_test is None or y_test is None:
        if x_test is None:
            x_test, y_test = generate_data(N_test, f, bias=True, rng=rng)
        else:
            y_test = f(x_test)
    return (w, N_sv, vf(w, x_test, y_test))[1 - hyp:]
```

```
if __name__ == "__main__":
    rng = np.random.default_rng()
    Ns = (10, 100)
    N_runs = 1_000
    N_test = 100_000

    print(f"\n[HW7 P8-10]\nPLA vs. SVM with hard margins over {N_runs:,} runs:")
    f = target_function_random_line(rng=rng)
    clf = svm.SVC(C=np.finfo(float).max, kernel="linear")
    for N in Ns:
        prob_svm = 0
        N_sv_avg = 0
        for _ in range(N_runs):
            while True:
                x, y = generate_data(N, f, bias=True, rng=rng)
                if not np.allclose(y, y[0]):
                    break
            x_test, y_test = generate_data(N_test, f, bias=True, rng=rng)
            _, E_out_pla = perceptron(N, f, vf=validate_binary, x=x, y=y,
                                        x_test=x_test, y_test=y_test, rng=rng)
            N_sv, E_out_svm = support_vector_machine(
                N, f, vf=validate_binary, x=x, y=y, x_test=x_test, y_test=y_test,
                clf=clf, rng=rng
            )
            prob_svm += E_out_svm < E_out_pla
            N_sv_avg += N_sv
        prob_svm /= N_runs
        N_sv_avg /= N_runs
        print(f"  {N=}, {prob_svm=:.3f}, {N_sv_avg=:.3f}")
```