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 | [c] |
| 9 | [d] |
| 10 | [b] |

**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 through which transform into

To illustrate how taking out points for validation affects the performance, we will consider the hypotheses trained on (without restoring the full 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 through , with . For which model is the classification error on the validation set smallest?

**Answer: [d]**

1. 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]**

1. 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]**

1. 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]**

1. 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:

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]

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**

1. Let and be independent random variables, distributed uniformly over the interval . Let . The expected values of , , and are closest to

**Answer: [d] 0.5, 0.5, 0.4**

For a continuous random variable over the interval , the expected value is given by

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

For ,

Therefore, the expected values of , , and are 0.5, 0.5, and , respectively.

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

[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.

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**

1. You are given the data points , , and a choice between two models: constant and linear . For which value of would the two models be tied using leave-one-out cross-validation with the squared error measure?

**Answer: [c]**

With the leave-one-out cross-validation approach, one of the three data points will be discarded. Let us take generally and to be the two data points used in the model.

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

For the linear model, the line of best fit gives

(as derived in Homework 4 Problem 7).

With the three possible unique combinations of points, we have

The squared errors for the two models are

Solving for by setting ,

**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 margin1 on linearly separable data sets. For each run, you will create your own target function and data set . Take and choose a random line in the plane as your target function (do this by taking two random, uniformly distributed points on and taking the line passing through them), where one side of the line maps to +1 and the other maps to . Choose the inputs of the data set as random points in , and evaluate the target function on each to get the corresponding output . 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 and measure the disagreement between and as (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 by solving

using quadratic programming on the primal2 or the dual problem or using an SVM package. Measure the disagreement between and as , and count the number of support vectors you get in each run.

1For hard margin in SVM packages, set and choose the “linear” kernel.

2Primal problem is the original formulation in slide 11 of Lecture 14.

1. For , repeat the above experiment for 1,000 runs. How often is better than in approximating ? The percent of time is closest to:

**Answer: [c] 60%**

1. For , repeat the above experiment for 1,000 runs. How often is better than in approximating ? The percentage of time is closest to:

**Answer: [d] 70%**

1. For the case , which of the following is closest to the average number of support vectors of (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 , where is the -coordinate of point in data set , and the misclassification rate is estimated by using a test data set with . 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:

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}")