Benjamin Ye

CS/CNS/EE 156a: Learning Systems (Fall 2023)

December 1, 2023

**Final Exam**

|  |  |  |  |
| --- | --- | --- | --- |
| **Problem** | **Answer** | **Problem** | **Answer** |
| 1 | [e] | 11 | [c] |
| 2 | [d] | 12 | [c] |
| 3 | [d] | 13 | [a] |
| 4 | [d] | 14 | [e] |
| 5 | [a] | 15 | [d] |
| 6 | [b] | 16 | [d] |
| 7 | [d] | 17 | [c] |
| 8 | [b] | 18 | [a] |
| 9 | [e] | 19 | [b] |
| 10 | [a] | 20 | [c] |

**Nonlinear Transforms**

1. The polynomial transform of order applied to of dimension resuilts in a space of what dimensionality (not counting the constant coordinate or )?

**Answer: [e] None of the above**

A second-order polynomial transformation gives :

A third-order polynomial transformation gives :

A fourth-order polynomial transformation gives (from Homework 5 Problem 3).

The examples above show that each additional order adds terms. By continuing this pattern for higher-order polynomial transformations, the dimensionality for order is given by

Therefore, for , the space has dimensionality

**Bias and Variance**

1. Recall that the average hypothesis was based on training the same model on different data sets to get , and taking the expected value of with respect to to get . Which of the following models could result in ?

**Answer: [d] is the logistic regression model**

Choices [a] through [c] cannot give :

1. The average of a singleton hypothesis is itself, so .
2. The average of constant, real-valued hypotheses is a constant, real-valued number, so .
3. In the linear regression model, a hypothesis has the form of a linear combination . The average of these linear combinations is another linear combination, so .

However, in the logistic regression model, a hypothesis has the form of a logistic (sigmoidal) function . The average of these logistic functions is not necessarily another logistic function, so could be true for [d].

For example, consider the simple hypothesis set containing only

The average is

which cannot be reduced to a logistic function.

**Overfitting**

1. Which of the following statements is false?

**Answer: [d] We can always determine if there is overfitting by comparing the values of .**

Overfitting occurs when a hypothesis with the lowest is selected despite its higher .

Choices [a] through [c] are all true:

1. There must be at least two hypotheses with different values to have overfitting.
2. There must be at least two hypotheses with different values to have overfitting.
3. For overfitting to occur, the optimal and overfitted hypotheses must have different values. For example, consider two hypotheses with in-sample and out-of-sample error and , with the first hypothesis selected for its lower value. If the two hypotheses have the same values, then for the second hypothesis must be 0.4, which would then mean that no overfitting occurred.

Now, let’s change the errors of the first hypothesis in the example in [c] to . The values for the old and new hypotheses are the same, but there is no longer overfitting. Therefore, the magnitudes of the values cannot be used to determine if there is overfitting, and [d] is false.

1. Which of the following statements is true?

**Answer: [d] Stochastic noise does not depend on the hypothesis set.**

Choices [a], [b], [c], and [e] are all false:

1. Deterministic and stochastic noise can be present simultaneously; the data set can have inherent fluctuations or measurement errors, and certain parts of the target function may be impossible to reproduce using the chosen model.
2. Deterministic noise is caused by the inability of the hypothesis set to model a target function (e.g., hypothesis set of linear combinations used to fit a quadratic target function).
3. Same explanation as [b].
4. Stochastic noise is dependent on the target distribution, as a distribution with a lower variance will have less stochastic noise.

Since stochastic noise is a property of only the input data, [d] is true.

**Regularization**

1. The regularized weight is a solution to:

where is a Tikhonov matrix. If , where is the linear regression solution, then what is ?

**Answer: [a]**

As the linear regression solution minimizes the squared error and satisfies the same constraint used to determine , it is equal to the regularized weight .

1. Soft-order constraints that regularize polynomial models can be

**Answer: [b] translated into augmented error**

Soft-order constraints prevent overfitting by encouraging the weights to be small, but not necessarily zero like hard constraints, and does not change the order of the polynomial. This means that

1. is false by definition,
2. is false since the VC dimension does not change despite the effectively smaller model with regularization, and
3. is false since is expected to increase for better generalization (lower ).

A soft-order constraint yields the optimization problem

For to be optimal,

By grouping into the undetermined variable , satisfies

That is, locally minimizes , which is the augmented error where the second term represents a penalty term. Therefore, [b] is true.

**Regularized Linear Regression**

We are going to experiment with linear regression for classification on the processed U.S. Postal Service Zip Code data set from Homework 8. Download the data (extracted features of intensity and symmetry) for training and testing:

<http://www.amlbook.com/data/zip/features.train>

<http://www.amlbook.com/data/zip/features.test>

(The format of each row is: **digit, intensity, symmetry**.) We will train two types of binary classifiers; one-versus-one (one digit is class +1 and another digit is class , with the rest of the digits disregarded), and one-versus-all (one digit is class +1 and the rest of the digits are class ). When evaluating and of the resulting classifier, use binary classification error. Implement the regularized least-squares linear regression for classification that minimizes

where includes .

1. Set and dop not apply a feature transform (i.e., use ). Which among the following classifiers has the lowest ?

**Answer: [d] 8 versus all**

1. Now, apply a feature transform , and set . Which among the following classifiers has the lowest ?

**Answer: [b] 1 versus all**

1. If we compare using the transform versus not using it, and applying that to “0 versus all” through “9 versus all”, which of the following statements is correct for ?

**Answer: [e] The transform improves the out-of-sample performance of “5 versus all”, but by less than 5%.**

The transform improves the out-of-sample error from 0.079721 to 0.079223, a 0.6% decrease, for the “5 versus all” classifier.

1. Train the “1 versus 5” classifier with with and . Which of the following statements is correct?

**Answer: [a] Overfitting occurs (from to ).**

From to , decreases from 0.005125 to 0.004484, but increases from 0.025943 to 0.028302. This signifies overfitting since the hypothesis will be selected due for its lower in-sample error despite its worse out-of-sample performance.

(The sample program output and Python 3 source code are on the following pages. )

**Sample program output**

[Final Exam Problems 7–9]

Linear regression with regularization (lambda=1):

classifier E\_in E\_out transform E\_in transform E\_out

0 vs. all 0.109313 0.115097 0.102318 0.106627

1 vs. all 0.015224 0.022422 0.012344 **0.021923**

2 vs. all 0.100261 0.098655 0.100261 0.098655

3 vs. all 0.090248 0.082711 0.090248 0.082711

4 vs. all 0.089425 0.099651 0.089425 0.099651

5 vs. all 0.076258 **0.079721** 0.076258 **0.079223**

6 vs. all 0.091071 0.084704 0.091071 0.084704

7 vs. all 0.088465 0.073244 0.088465 0.073244

8 vs. all 0.074338 0.082711 **0.074338** 0.082711

9 vs. all 0.088328 0.088191 0.088328 0.088191

[Final Exam Problem 10]

Linear regression with regularization and transform for 1 vs. 5 classifier:

lambda in-sample error out-of-sample error

0.01 **0.004484 0.028302**

1.00 **0.005125 0.025943**

**Python 3 source code**

from pathlib import Path

import numpy as np

import pandas as pd

import requests

DATA\_DIR = (Path(\_\_file\_\_).resolve().parent / "data").resolve()

rng = np.random.default\_rng()

class LinearRegression:

    def \_\_init\_\_(

            self, \*, vf=None, regularization=None, transform=None, noise=None,

            rng=None, seed=None, \*\*kwargs):

        self.rng = np.random.default\_rng(seed) if rng is None else rng

        self.set\_parameters(vf=vf, regularization=regularization,

                            transform=transform, noise=noise, \*\*kwargs)

    def get\_error(self, x, y):

        if self.transform:

            x = self.transform(x)

        if self.noise:

            N = x.shape[0]

            index = self.rng.choice(N, round(self.noise[0] \* N), False)

            y[index] = self.noise[1](y[index])

        if self.vf is not None and self.w is not None:

            return self.vf(self.w, x, y)

    def set\_parameters(

            self, \*, vf=None, regularization=None, transform=None, noise=None,

            update=False, \*\*kwargs):

        self.\_reg\_params = {}

        self.w = None

        if update:

            self.noise = noise or self.noise

            self.regularization = regularization or self.regularization

            if self.regularization == "weight\_decay" \

                    and "weight\_decay\_lambda" in kwargs:

                self.\_reg\_params["lambda"] = kwargs["weight\_decay\_lambda"]

            self.transform = transform or self.transform

            self.vf = vf or self.vf

        else:

            self.noise = noise

            self.regularization = regularization

            if regularization == "weight\_decay":

                self.\_reg\_params["lambda"] = kwargs["weight\_decay\_lambda"]

            self.transform = transform

            self.vf = vf

    def train(self, x, y):

        if self.transform:

            x = self.transform(x)

        if self.noise:

            N = x.shape[0]

            index = self.rng.choice(N, round(self.noise[0] \* N), False)

            y[index] = self.noise[1](y[index])

        if self.regularization is None:

            self.w = np.linalg.pinv(x) @ y

        elif self.regularization == "weight\_decay":

            self.w = np.linalg.inv(

                x.T @ x

                + self.\_reg\_params["lambda"] \* np.eye(x.shape[1], dtype=float)

            ) @ x.T @ y

        if self.vf is not None:

            return self.vf(self.w, x, y)

def validate\_binary(w, x, y):

    return np.count\_nonzero(np.sign(x @ w) != y, axis=0) / x.shape[0]

if \_\_name\_\_ == "\_\_main\_\_":

    DATA\_DIR.mkdir(exist\_ok=True)

    data = {}

    for dataset in ["train", "test"]:

        file = f"features.{dataset}"

        if not (DATA\_DIR / file).exists():

            r = requests.get(f"http://www.amlbook.com/data/zip/{file}")

            with open(DATA\_DIR / file, "wb") as f:

                f.write(r.content)

        data[dataset] = np.loadtxt(DATA\_DIR / file)

    weight\_decay\_lambda = 1

    transform = lambda x: np.hstack((x, x[:, 1:2] \* x[:, 2:], x[:, 1:2] \*\* 2,

                                    x[:, 2:] \*\* 2))

    reg = LinearRegression(vf=validate\_binary, regularization="weight\_decay",

                           weight\_decay\_lambda=weight\_decay\_lambda, rng=rng)

    reg\_transform = LinearRegression(vf=validate\_binary,

                                     regularization="weight\_decay",

                                     transform=transform,

                                     weight\_decay\_lambda=weight\_decay\_lambda,

                                     rng=rng)

    df = pd.DataFrame(columns=["classifier", "E\_in", "E\_out",

                               "transform E\_in", "transform E\_out"])

    for digit in range(10):

        x\_train = np.hstack((np.ones((len(data["train"]), 1), dtype=float),

                            data["train"][:, 1:]))

        y\_train = 2 \* (data["train"][:, 0] == digit) - 1

        x\_test = np.hstack((np.ones((len(data["test"]), 1), dtype=float),

                            data["test"][:, 1:]))

        y\_test = 2 \* (data["test"][:, 0] == digit) - 1

        E\_in = reg.train(x\_train, y\_train)

        E\_in\_transform = reg\_transform.train(x\_train, y\_train)

        df.loc[digit] = (f"{digit} vs. all", E\_in,

                         reg.get\_error(x\_test, y\_test), E\_in\_transform,

                         reg\_transform.get\_error(x\_test, y\_test))

    print("\n[Final Exam Problems 7–9]\n"

          "Linear regression with regularization "

          f"(lambda={weight\_decay\_lambda}):\n", df.to\_string(index=False),

          sep="")

    subset\_train = data["train"][np.isin(data["train"][:, 0], (1, 5))]

    x\_train = np.hstack((np.ones((subset\_train.shape[0], 1), dtype=float),

                         subset\_train[:, 1:]))

    y\_train = (subset\_train[:, 0] == 1).astype(int) - (subset\_train[:, 0] == 5)

    subset\_test = data["test"][np.isin(data["test"][:, 0], (1, 5))]

    x\_test = np.hstack((np.ones((subset\_test.shape[0], 1), dtype=float),

                        subset\_test[:, 1:]))

    y\_test = (subset\_test[:, 0] == 1).astype(int) - (subset\_test[:, 0] == 5)

    df = pd.DataFrame(columns=["lambda", "in-sample error",

                               "out-of-sample error"])

    for weight\_decay\_lambda in (0.01, 1):

        reg\_transform.set\_parameters(weight\_decay\_lambda=weight\_decay\_lambda,

                                     update=True)

        E\_in = reg\_transform.train(x\_train, y\_train)

        df.loc[len(df)] = (weight\_decay\_lambda, E\_in,

                           reg\_transform.get\_error(x\_test, y\_test))

    print("\n[Final Exam Problem 10]\n"

          "Linear regression with regularization and transform for "

          "1 vs. 5 classifier:\n", df.to\_string(index=False), sep="")

**Support Vector Machines**

1. Consider the following training set generated from a target function , where :

Transform this training set into another two-dimensional space :

Using geometry (not quadratic programming), what values of (without ) and specify the separating plane that maximizes the margin in the space? The values of , , and are:

**Answer: [c]**

A graph of a number of numbers and points

Description automatically generated

In space, the separating plane should be placed between the two groups of blue squares and orange circles above. The ideal placement is at and any ( is a point on the plane). This arrangement maximizes the margin between the plane and the three closest points at , , and , with each point having a Euclidean distance of 0.5 from the plane.

The form of suggests that and (since the value is irrelevant), and can be found using

1. Consider the same training set of the previous problem, but instead of explicitly transforming the input space , apply the hard-margin SVM algorithm with the kernel

(which corresponds to a second-order polynomial transformation). Set up the expression for and solve for the optimal (numerically, using a quadratic programming package). The number of support vectors you get is in what range?

**Answer: [c] 4–5**

**Sample program output**

[Final Exam Problem 12]

The second-order polynomial hard margin support vector machine (SVM) uses **5** support vectors.

(The Python 3 source code is on the following page. )

**Python 3 source code**

import matplotlib as mpl

import matplotlib.pyplot as plt

import numpy as np

from sklearn import svm

mpl.rcParams.update(

    {

        "axes.labelsize": 14,

        "figure.autolayout": True,

        "figure.figsize": (4.875, 3.65625),

        "font.size": 12,

        "legend.columnspacing": 1,

        "legend.edgecolor": "1",

        "legend.framealpha": 0,

        "legend.fontsize": 12,

        "legend.handlelength": 1.25,

        "legend.labelspacing": 0.25,

        "xtick.labelsize": 12,

        "ytick.labelsize": 12,

        "text.usetex": True

    }

)

if \_\_name\_\_ == "\_\_main\_\_":

    x = np.array(((1, 0), (0, 1), (0, -1), (-1, 0), (0, 2), (0, -2), (-2, 0)),

                 dtype=float)

    y = np.array((-1, -1, -1, 1, 1, 1, 1), dtype=int)

    z = np.hstack((x[:, 1:] \*\* 2 - 2 \* x[:, :1] - 1,

                   x[:, :1] \*\* 2 - 2 \* x[:, 1:] + 1))

    ticks = np.arange(-6, 7)

    \_, ax = plt.subplots()

    ax.grid(ls=":")

    ax.plot(\*z[y == 1].T, "s", label="$+1$")

    ax.plot(\*z[y == -1].T, "o", label="$-1$")

    ax.set\_aspect("equal", "box")

    ax.set\_xlabel("$z\_1$")

    ax.set\_xlim(-6, 6)

    ax.set\_xticks(ticks)

    ax.set\_ylabel("$z\_2$")

    ax.set\_ylim(-6, 6)

    ax.set\_yticks(ticks)

    ax.legend(title="classification", loc="lower left")

    plt.show()

    clf = svm.SVC(C=np.finfo(float).max, kernel="poly", degree=2, gamma=1,

                  coef0=1)

    clf.fit(x, y)

    print("\n[Final Exam Problem 12]\n"

          "The second-order polynomial hard margin support vector "

          f"machine (SVM) uses {clf.n\_support\_.sum()} support vectors.")

**Radial Basis Functions**

We experiment with the RBF model, both in regular form (Lloyd + pseudo-inverse) with centers:

(notice that there is a bias term), and in kernel form (using the RBF kernel in hard-margin SVM):

The input space is with uniform probability distribution, and the target is

which is slightly nonlinear in the space. In each run, generate 100 training points at random using this target, and apply both forms of RBF to these training points. Here are some guidelines:

* Repeat the experiment for as many runs as needed to get the answer to be stable (statistically away from flipping to the closest competing answer).
* In case a data set is not separable in the “ space” by the RBF kernel using hard-margin SVM, discard the run but keep track of how often this happens, if ever.
* When you use Lloyd’s algorithm, initialize the centers to random points in and iterate until there is no change from iteration to iteration. If a cluster becomes empty, discard the run and repeat.

1. For , how often do you get a data set that is not separable by the RBF kernel (using hard-margin SVM)? Hint: Run the hard-margin SVM, then check if the solution has .

**Answer: [a] of the time**

1. If we use for regular RBF and take , how often does the kernel form beat the regular form (excluding runs mentioned in Problem 13 and runs with empty clusters, if any) in terms of ?

**Answer: [e] of the time**

1. If we use for regular RBF and take , how often does the kernel form beat the regular form (excluding runs mentioned in Problem 13 and runs with empty clusters, if any) in terms of ?

**Answer: [d] but of the time**

1. Now we focus on regular RBF only, with . If we go from clusters to clusters (only 9 and 12), which of the following 5 cases happens most often in your runs (excluding runs with empty clusters, if any)? Up or down means strictly so.

**Answer: [d] Both and go down.**

1. For regular RBF with , if we go from to (only 1.5 and 2), which of the following 5 cases happens most often in your runs (excluding runs with empty clusters, if any)? Up or down means strictly so.

**Answer: [c] Both and go up.**

1. What is the percentage of time that regular RBF achieves with and (excluding runs with empty clusters, if any)?

**Answer: [a] of the time**

**Sample output**

[Final Exam Problems 13–18]

Radial basis function (RBF) model (100 runs):

gamma K nonseparable outperform E\_in=0 E\_in E\_out

1.5 9.0 **0.0 0.83** **0.04** **0.0273 0.04678**

1.5 12.0 0.0 **0.84** 0.12 **0.0185 0.04045**

2.0 9.0 0.0 0.86 0.06 **0.0296 0.05223**

(The Python 3 source code is on the following pages. )

**Python 3 source code**

import numpy as np

import pandas as pd

from sklearn import svm

from sklearn.cluster import k\_means

rng = np.random.default\_rng()

class RBFRegular:

    def \_\_init\_\_(self, gamma, K, \*, vf) -> None:

        self.set\_parameters(gamma, K, vf=vf)

    def get\_error(self, x, y):

        if self.vf is not None and self.w is not None:

            return self.vf(self.w, self.get\_phi(x, self.centers), y)

    def get\_phi(self, x, centers):

        return np.hstack((

            np.ones((x.shape[0], 1), dtype=float),

            np.exp(-self.gamma

                   \* np.linalg.norm((x[:, None] - centers), axis=2) \*\* 2)

        ))

    def set\_parameters(self, gamma, K, \*, vf=None, update=False) :

        self.centers = None

        self.w = None

        if update:

            self.gamma = gamma or self.gamma

            self.K = K or self.K

            self.vf = vf or self.vf

        else:

            self.gamma = gamma

            self.K = K

            self.vf = vf

    def train(self, x: np.ndarray[float], y: np.ndarray[float]) -> float:

        self.centers = k\_means(x, self.K, n\_init="auto")[0]

        phi = self.get\_phi(x, self.centers)

        self.w = np.linalg.pinv(phi) @ y

        if self.vf is not None:

            return self.vf(self.w, phi, y)

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 target\_function\_final\_exam(x):

    f = lambda x: np.sign(np.diff(x[:, -2:], axis=1)[:, 0]

                          + 0.25 \* np.sin(np.pi \* x[:, -1]))

    return f if x is None else f(x)

if \_\_name\_\_ == "\_\_main\_\_":

    N\_runs = 100

    N\_train = 100

    N\_test = 1\_000

    columns = ["gamma", "K", "% nonseparable", "% outperform",

               "% E\_in=0", "E\_in", "E\_out"]

    df = pd.DataFrame(columns=columns)

    for gamma, K in [(1.5, 9), (1.5, 12), (2, 9)]:

        clf = svm.SVC(C=np.finfo(float).max, gamma=gamma)

        rbf = RBFRegular(gamma, K, vf=validate\_binary)

        counters = np.zeros(5, dtype=float)

        for \_ in range(N\_runs):

            x\_train, y\_train = generate\_data(N\_train,

                                             target\_function\_final\_exam,

                                             rng=rng)

            x\_test, y\_test = generate\_data(N\_test,

                                           target\_function\_final\_exam,

                                           rng=rng)

            clf.fit(x\_train, y\_train)

            if not np.isclose(clf.score(x\_train, y\_train), 1):

                counters[0] += 1

                continue

            E\_in = rbf.train(x\_train, y\_train)

            E\_out = rbf.get\_error(x\_test, y\_test)

            counters[1:5] += (1 - clf.score(x\_test, y\_test) < E\_out,

                              E\_in == 0, E\_in, E\_out)

        counters /= N\_runs

        counters[:3] \*= 100

        df.loc[len(df)] = gamma, K, \*counters

    print("\n[Final Exam Problems 13–18]\n"

          f"Radial basis function (RBF) model ({N\_runs:,} runs):\n",

          df.to\_string(index=False), sep="")

**Bayesian Priors**

1. Let be the unknown probability of getting a heart attack for people in a certain population. Notice that is just a constant, not a function, for simplicity. We want to model using a hypothesis . Before we see any data, we assume that is uniform over (the prior). We pick one person from the population, and it turns out that he or she had a heart attack. Which of the following is true about the posterior probability that given this sample point?

**Answer: [b] The posterior increases linearly over .**

Using Bayes’ theorem, the posterior can be related to the prior via

is expected to be a constant probability, akin to one used in a binomial distribution. The randomly chosen person either has or has not had a heart attack.

is the probability that a randomly chosen person has had a heart attack given the likelihood of getting a heart attack in that population. Intuitively, this probability increases linearly with ; the higher the chance of getting a heart attack, the more likely the randomly selected person has had one.

From the problem statement, it is known that is a uniform distribution, i.e., constant across .

Multiplying a linearly increasing distribution by a uniform distribution gives back a linearly increasing distribution. Therefore, the posterior is expected to increase linearly over .

**Aggregation**

1. Given two learned hypotheses and , we construct the aggregate hypothesis given by for all . If we use the mean-squared error, which of the following statements is true?

**Answer: [c] cannot be worse than the average of and .**

Choices [a], [b], and [d] are all false:

1. Consider the case where , or that is further away from the target function than is, with both and overestimating . By taking their average to get , the resulting hypothesis will model less accurately and have a higher mean-squared error than .
2. Same explanation as [a].
3. Consider the case where underestimates the target function while overestimates it. The averaged hypothesis could then be much closer to the target function and result in being much lower than both and .

Using the definition of the mean-squared error,

it can be shown that the fully expanded mean-squared error of is

The average of the mean-squared errors of and is

The second term in the last line is greater than or equal to zero since it is squared. Therefore, is always smaller than the average of and , so [c] is true.