Benjamin Ye

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

October 30, 2023

**Homework 5**

|  |  |
| --- | --- |
| **Problem** | **Answer** |
| 1 | [c] |
| 2 | [d] |
| 3 | [c] |
| 4 | [e] |
| 5 | [d] |
| 6 | [e] |
| 7 | [a] |
| 8 | [d] |
| 9 | [a] |
| 10 | [e] |

**Linear Regression Error**

Consider a noisy target , where (with the added coordinate ), , is an unknown vector, and is a noise term with zero mean and variance. Assume is independent of and of all other ’s. If linear regression is carried out using a training data set , and outputs the parameter vector , it can be shown that the expected in-sample error with respect to is given by

1. For and , which among the following choices is the smallest number of examples that will result in an expected greater than ?

**Answer: [c] 100**

Solving for by turning the equation above into an inequality,

**Nonlinear Transforms**

In linear classification, consider the feature transform (plus the added zeroth coordinate) given by

1. Which of the following sets of constraints on the weights in the space could correspond to the hyperbolic decision boundary in depicted in the following figure?

A diagram of a mathematical equation

Description automatically generated

You may assume that can be selected to achieve the desired boundary.

**Answer: [d]**

When has been determined, should give the correct classification for in .

Let us take the center of the figure above to be an arbitrary reference point, i.e., the hyperbola has its center at , since can always be translated such that this is the case. As increases (moving away horizontally from the origin), the function tends to so should be negative. As increases (moving away vertically from the origin), the function tends to +1, so should be positive.

Analytically, this is equivalent to considering the general form for a hyperbola:

where , , and are arbitrary non-zero constants.

By moving all terms to the right-hand side such that the left-hand side of the equation above is (to determine where the hyperbolic decision boundary is) and taking the sign of the right-hand sign, i.e., , we obtain a formulation equivalent in form to for . We can see that is positive, is negative, and is positive.

Now, consider the 4th order polynomial transform from the input space :

1. What is the smallest value among the following choices that is *not* smaller than the VC dimension of a linear model in this transformed space?

**Answer: [c] 15**

For , has dimensions (not counting the zeroth coordinate), so the VC dimension of is .

**Gradient Descent**

Consider the nonlinear error surface . We start at the point and minimize this error using gradient descent in the space. Use (learning rate, not step size).

1. What is the partial derivative of with respect to , i.e., ?

**Answer: [e]**

By applying the chain rule, the partial derivative is

1. How many iterations (among the given choices) does it take for the error to fall below for the first time? In your programs, make sure to use double precision to get the needed accuracy.

**Answer: [d] 10**

1. After running enough iterations such that the error has just dropped below , what are the closest values (in Euclidean distance) among the following choices to the final you got in Problem 5?

**Answer: [e]**

1. Now, we will compare the performance of “coordinate descent”. In each iteration, we have two steps along the two coordinates. Step 1 is to move only along the coordinate to reduce the error (assume first-order approximation holds like in gradient descent), and step 2 is the reevaluate and move along only the coordinate to reduce the error (again, assume first-order approximation holds). Use the same learning rate of as we did in gradient descent. What will the error be closest to after 15 full iterations (30 steps)?

**Answer: [a]**

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

**Derivations**

The other missing piece required for the gradient descent method is the partial derivative of with respect to , which is

**Sample program output**

[Homework 5 Problems 5–7]

Descent methods (eta=0.1):

method iterations x E(x)

gradient descent  **10** **[0.044736, 0.023959]** 1.208683e-15

coordinate descent 15 [6.297076, -2.852307] **1.398138e-01**

**Python 3 source code**

import numpy as np

import pandas as pd

def gradient\_descent(E, dE, x, \*, eta=0.1, tol=1e-14, max\_iters=100\_000):

    iters = 0

    while E(x) > tol and iters < max\_iters:

        x -= eta \* dE(x)

        iters += 1

    return x, iters

def coordinate\_descent(E, dE\_dx, x, \*, eta=0.1, tol=1e-14, max\_iters=100\_000):

    iters = 0

    while E(x) > tol and iters < max\_iters:

        for i in range(len(x)):

            x[i] -= eta \* dE\_dx[i](x)

        iters += 1

    return x, iters

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

    rng = np.random.default\_rng()

    eta = 0.1

    E = lambda x: (x[0] \* np.exp(x[1]) - 2 \* x[1] \* np.exp(-x[0])) \*\* 2

    dE\_du = lambda x: (2 \* (x[0] \* np.exp(x[1]) - 2 \* x[1] \* np.exp(-x[0]))

                       \* (np.exp(x[1]) + 2 \* x[1] \* np.exp(-x[0])))

    dE\_dv = lambda x: (2 \* (x[0] \* np.exp(x[1]) - 2 \* x[1] \* np.exp(-x[0]))

                       \* (x[0] \* np.exp(x[1]) - 2 \* np.exp(-x[0])))

    df = pd.DataFrame(columns=["method", "iterations", "x", "E(x)"])

    x, iters = gradient\_descent(E, lambda x: np.array((dE\_du(x), dE\_dv(x))),

                                np.array((1, 1), dtype=float), eta=eta)

    df.loc[len(df)] = "gradient descent", iters, np.round(x, 6), E(x)

    x, iters = coordinate\_descent(E, (dE\_du, dE\_dv),

                                  np.array((1, 1), dtype=float), eta=eta,

                                  max\_iters=15)

    df.loc[len(df)] = "coordinate descent", iters, np.round(x, 6), E(x)

    print(f"\n[Homework 5 Problems 5–7]\nDescent methods ({eta=}):\n",

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

**Logistic Regression**

In this problem, you will create your own target function (probability in this case) and data set to see how logistic regression works. For simplicity, we will take to be a 0/1 probability so is a deterministic function of .

Take so you can visualize the problem and let with uniform probability of picking each . Choose a line in the plane as the boundary between (where must be +1) and (where must be ) by taking two random, uniformly distributed points from and taking the line passing through them as the boundary between . Pick training points at random from and evaluate the outputs for each of these points .

Run logistic regression with stochastic gradient descent to find and estimate (the cross-entropy error) by generating a sufficiently large, separate set of points to evaluate the error. Repeat the experiment for 100 runs with different targets and take the average. Initialize the weight vector of logistic regression to all zeros in each run. Stop the algorithm when , where denotes the weight vector at the end of epoch . An epoch is a full pass through the data points (use a random permutation of to present the data points to the algorithm within each epoch and use different permutations for different epochs). Use a learning rate of .

1. Which of the following is closest to for ?

**Answer: [d] 0.100**

1. How many epochs does it take on average for logistic regression to converge for using the above initialization and termination rules and the specified learning rate? Pick the value that is closest to your results.

**Answer: [a] 350**

For the stochastic gradient descent, the target function is a simple line , where is the -coordinate of point in data set .

**Sample program output**

[Homework 5 Problems 8–9]

Using stochastic gradient descent with eta=0.01, the average number of epochs and out-of-sample error over 100 runs are **330** and **0.105525**, respectively.

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

**Python 3 source code**

import numpy as np

from sklearn.utils import shuffle

class StochasticGradientDescent:

    def \_\_init\_\_(self, eta=0.01, tol=0.01, \*, rng=None, seed=None):

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

        self.vf = lambda w, x, y: np.log(1 + np.exp(-y[:, None] \* x @ w)).mean()

        self.set\_parameters(eta, tol)

    def get\_error(self, x, y):

        if self.w is not None:

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

    def set\_parameters(self, eta=None, tol=None, update=False):

        if update:

            self.eta = eta or self.eta

            self.tol = tol or self.tol

        else:

            self.eta = eta

            self.tol = tol

    def train(self, x, y) -> float:

        self.w = np.zeros(x.shape[1], dtype=float)

        self.epochs = 0

        while True:

            w = self.w.copy()

            for x\_, y\_ in zip(\*shuffle(x, y)):

                w += self.eta \* y\_ \* x\_ / (1 + np.exp(y\_ \* x\_ @ w))

            dw = w - self.w

            self.w = w.copy()

            self.epochs += 1

            if np.linalg.norm(dw) < self.tol:

                break

        return self.vf(self.w, x, y)

def target\_function\_random\_line(x=None, \*, rng=None, seed=None):

    if rng is None:

        rng = np.random.default\_rng(seed)

    line = rng.uniform(-1, 1, (2, 2))

    f = lambda x: np.sign(

        x[:, -1] - line[0, 1]

        - np.divide(\*(line[1] - line[0])[::-1]) \* (x[:, -2] - line[0, 0])

    )

    return f if x is None else f(x)

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)

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

    rng = np.random.default\_rng()

    eta = 0.01

    N\_runs = N\_train = 100

    N\_test = 9 \* N\_train

    sgd = StochasticGradientDescent(eta, rng=rng)

    counters = np.zeros(2, dtype=float)

    for \_ in range(N\_runs):

        f = target\_function\_random\_line(rng=rng)

        sgd.train(\*generate\_data(N\_train, f, bias=True, rng=rng))

        counters += (

            sgd.epochs,

            sgd.get\_error(\*generate\_data(N\_test, f, bias=True, rng=rng))

        )

    counters /= N\_runs

    print("\n[Homework 5 Problems 8–9]\n"

          f"Using stochastic gradient descent with {eta=}, the average "

          f"number of epochs and out-of-sample error over {N\_runs} runs "

          f"are {counters[0]:.0f} and {counters[1]:.6f}, respectively.")

**PLA as SGD**

1. The perceptron learning algorithm can be implemented as SGD using which of the following error functions ? Ignore the points at which is not twice differentiable.

**Answer: [e]**

In the PLA, the weight vector update for point is

In SGD, the weight vector update for point is

To implement the PLA as SGD, we set (as there is no “learning rate” in the PLA) and seek an error function that satisfies . The obvious candidate is , but it is important to note that is negative only when is misclassified using the weights in the current iteration.

Therefore, the error function should be so that no update is performed when is correctly classified.