Assignment 1

MASD 2021

Department of Computer Science
University of Copenhagen

Mikkel/pwz854 | partners = mfh144 & qfh987

29/11/2021

Contents

Exercise 1 2

Exercise 2 4

Exercise 3 6

Exercise 4 7

Exercise 5 8

Appendix 9

Housing_1.py 9

Housing_2.py 10

linreg.py 11

Exercise 1

Finding the partial derivatives $\frac{\partial f}{\partial x}$ and $\frac{\partial f}{\partial x}$ stepwise of each given function

a) $f(x,y) = x^4y^3 + x^5 - e^y$

$$\frac{\partial f}{\partial x} = (x^4y^3 + x^5 - e^y)'$$

Apply sum rule

$$\frac{\partial f}{\partial x} = (x^4y^3)' + (x^5)' - (e^y)'$$

$$\frac{\partial f}{\partial x} = 4x^3 \cdot y^3 + 5x^4$$

Now to the other partial derivative

$$\frac{\partial f}{\partial y} = (x^4y^3 + x^5 - e^y)'$$

Apply sum rule again

$$\frac{\partial f}{\partial y} = (x^4y^3)' + (x^5)' - (e^y)'$$

$$\frac{\partial f}{\partial y} = 3y^3 \cdot x^4 - e^y$$

b) $f(x,y) = \frac{1}{\sqrt{x^3 + xy + y^2}}$

$$\frac{\partial f}{\partial x} = (\frac{1}{\sqrt{x^3 + xy + y^2}})'$$

First rewrite the expression using $\frac{1}{a} = a^{-1}$

$$\frac{\partial f}{\partial x} = ((x^3 + xy + y^2)^{-\frac{1}{2}})'$$

Apply power rule combined with the chain rule where $f(u) = (u)^{-\frac{1}{2}}$ and $g(x) = (x^3 + xy + y^2)$ then

$$f'(u) = -\frac{1}{2}u^{-\frac{3}{2}}$$

$$g'(x) = 3x^2 + y$$

$$\frac{\partial f}{\partial x} = -\frac{1}{2}(x^3 + xy + y^2)^{-\frac{3}{2}} \cdot (3x^2 + y)$$

Now to the other partial derivative

$$\frac{\partial f}{\partial y} = ((x^3 + xy + y^2)^{-\frac{1}{2}})'$$

Apply power rule combined with the chain rule where $f(u) = (u)^{-\frac{1}{2}}$ and $g(y) = (x^3 + xy + y^2)$ then

$$f'(u) = -\frac{1}{2}u^{-\frac{3}{2}}$$

$$g'(y) = 2y + x$$

$$\frac{\partial f}{\partial x} = -\frac{1}{2}(x^3 + xy + y^2)^{-\frac{3}{2}} \cdot (2y + x)$$

c) $f(x,y) = \dfrac{x^3 + y^2}{x + y}$

$$\frac{\partial f}{\partial x} = (\frac{x^3 + y^2}{x + y})'$$

Apply the quotient rule where $f(x,y) = x^3 + y^2$ and $g(x,y) = x + y$ then

$$f'_x(x,y) = 3x^2$$

$$g'_x(x,y) = 1$$

$$\frac{\partial f}{\partial x} = \frac{(x+y)(3x^2) - (x^3 + y^2)(1)}{(x+y)^2}$$

Simplify the expression

$$\frac{\partial f}{\partial x} = \frac{2x^3 + 3x^2y - y^2}{(x+y)^2}$$

Now to the other partial derivative

$$\frac{\partial f}{\partial y} = (\frac{x^3 + y^2}{x + y})'$$

Apply the quotient rule where $f(x,y) = x^3 + y^2$ and $g(x,y) = x + y$ then

$$f'_y(x,y) = 2y$$

$$g'_y(x,y) = 1$$

$$\frac{\partial f}{\partial x} = \frac{(x+y)(2y) - (x^3 + y^2)(1)}{(x+y)^2}$$

Simplify the expression

$$\frac{\partial f}{\partial x} = \frac{-x^3 + 2yx + y^2}{(x+y)^2}$$

Exercise 2

Let $\overline{x}$ be a vector, A a matrix, $\overline{b}$ a vector, and c a scalar. Compute the gradient ∇f with respect to $\overline{x}$

a) $f(\overline{x}) = \overline{x}^T\overline{x} + c$

$$\nabla f = \left[\frac{\partial f}{\partial x_1} \cdots \frac{\partial f}{\partial x_n}\right]^T$$

The function then shows that we can write

$$x_1^2 + x_2^2 + \cdots + x_n^2 + c$$

Then the gradient ∇f becomes

$$\nabla f = [2x_1, 2x_2, \cdots, 2x_n]^T$$

b) $f(\overline{x}) = \overline{x}^T\overline{b}$

$$\nabla f = \left[\frac{\partial f}{\partial x_1} \cdots \frac{\partial f}{\partial x_n}\right]^T$$

The function then shows that we can write

$$x_1 \cdot b_1 + x_2 \cdot b_2 + \cdots + x_n \cdot b_n$$

Then the gradient ∇f becomes

$$\nabla f = [b_1, b_2, \cdots, b_n]^T$$

c) $f(\overline{x}) = \overline{x}^T A\overline{x} + \overline{b}^T\overline{x} + c$

$$\nabla f = \left[\frac{\partial f}{\partial x_1} \cdots \frac{\partial f}{\partial x_n}\right]^T$$

Start by making a new matrix $C = \overline{x}^T A$ that will look like

$$C = [x_1A_{11} + x_2A_{21} + \cdots + x_nA_{n1},\ x_1A_{12} + x_2A_{22} + \cdots + x_nA_{n2},\ \cdots,$$
$$x_1A_{1n} + x_2A_{2n} + \cdots + x_nA_{nn}]$$

Then $C\overline{x}$ becomes

$$C\overline{x} = (x_1A_{11} + x_2A_{21} + \cdots + x_nA_{n1})x_1 + (x_1A_{12} + x_2A_{22} + \cdots + x_nA_{n2})x_2 + \cdots +$$
$$(x_1A_{1n} + x_2A_{2n} + \cdots + x_nA_{nn})x_n$$

Then write out $\overline{b}^T\overline{x}$

$$b_1 \cdot x_1 + b_2 \cdot x_2 + \cdots + b_n \cdot x_n$$

This will result in the gradient ∇f becoming: (For clarification there will be put parentheses into the expression in order to show what $'\cdots'$ resembles)

$$\frac{\partial f}{\partial x_1} = (2x_1A_{11} + x_2A_{21} + \cdots + x_nA_{n1}) + (x_2A_{12} + \cdots + x_nA_{1n}) + b_1$$

$$\frac{\partial f}{\partial x_2} = (x_1A_{12} + 2x_2A_{22} + \cdots + x_nA_{n2}) + (x_1A_{21} + \cdots + x_nA_{2n} - x_2A_{22}) + b_2$$

$$\cdots$$

$$\frac{\partial f}{\partial x_n} = (x_1A_{1n} + x_2A_{2n} + \cdots + 2x_nA_{nn}) + (x_1A_{n1} + \cdots + x_nA_{nn} - x_nA_{nn}) + b_n$$

In the partial derivative with respect to x_2, the second parentheses has the extra element $-x_2A_{22}$ because this element has already been taken account of in the first parentheses and should therefore be taken out of the second parentheses. The same is done in the partial derivative with respect to x_n, in order to show that all partial derivatives between these two also has this extra expression in respect to their x.
For the partial derivative with respect to x_n, one could have left out the element $-x_nA_{nn}$ and then only summed up to $x_{n-1}A_{n-1n-1}$ instead.
The gradient ∇f could also have been computed using $\sum$ expressions, however i **personally** found it more comprehensible and easier understandable the way it is written now.
Using the differentiation table 1.4, p.23, in the book i get the result

$$\frac{\partial f}{\partial x} = 2A\overline{x} + \overline{b}$$

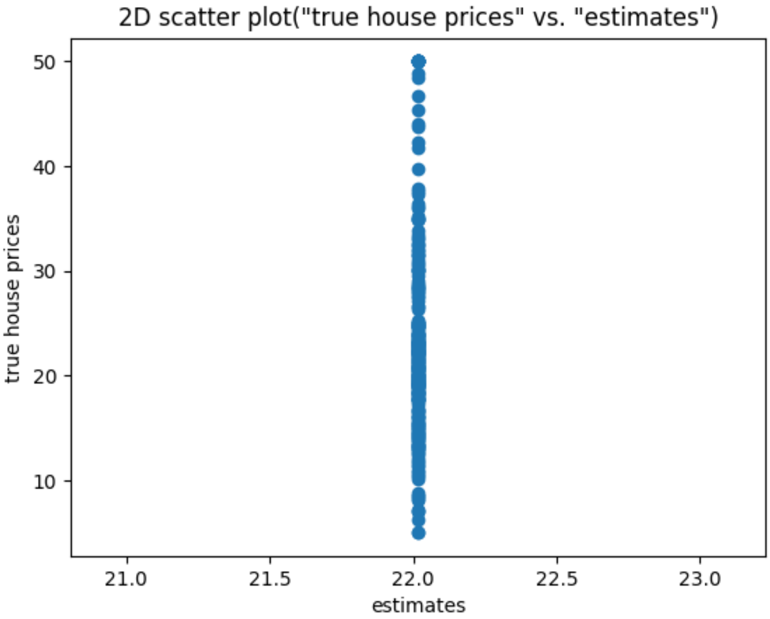
This is simply achieved by the sum rule.

Exercise 3

a) The mean was found to be 22.01660079

b) The RMSE between the true house prices and the estimates obtained via the simple ′mean′ model from a) was found to be 9.672477972746309

c) See plot below



Exercise 4

a) See file linreg.py at lines 40-42

b) The two weight is found to be:
$w_0 = 23.63506195$
$w_1 = -0.43279318$

Via the weight the linear expression $f(x) = ax + b$ can be rewritten as $f(x) = w_1 x + w_0$. Then from the function we get the estimated value of a house based on the crime rate.

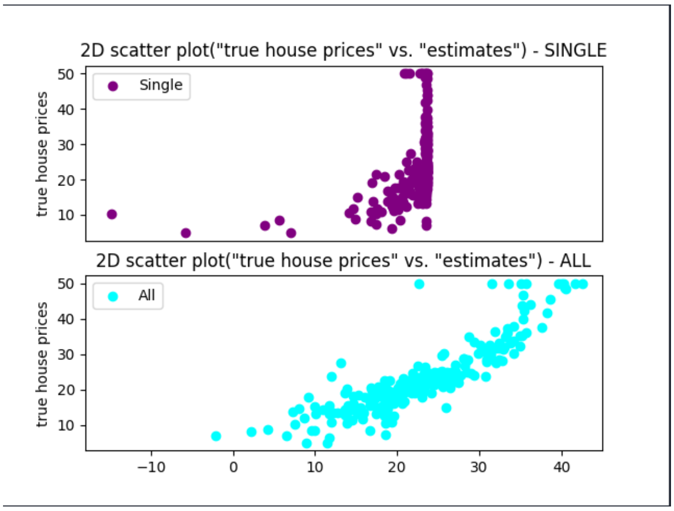
c) The weights w_i from the output of the fit function on all the features

```
All model weight w0: 31.388697820648627
All model weight w1: −0.05961691265965143
All model weight w2: 0.029367279233102644
All model weight w3: −0.0290605833675573
All model weight w4: 2.292561812630046
All model weight w5: −17.32636551510381
All model weight w6: 3.9937599595396325
All model weight w7: 0.0032307776111671262
All model weight w8: −1.2872450831596014
All model weight w9: 0.3547801905994277
All model weight w10: −0.015581919104618914
All model weight w11: −0.814647713302235
All model weight w12: 0.011782020829573633
All model weight w13: −0.4648690144560556
```

d) The RMSE was found to be

```
RMSE of all: 4.68833365362767
RMSE of single: 8.954859906611233
```

And the plots looks as the following:



Exercise 5

a) Start by just writing out the total training loss method

$$\mathcal{L} = \sum_{n=1}^{N} (w^T x_n - t_n)^2$$

Following the steps in the book this can be rewritten to

$$\mathcal{L} = (\mathbf{Xw} - \mathbf{t})^T (\mathbf{Xw} - \mathbf{t})$$

$$\mathcal{L} = ((\mathbf{Xw})^T - \mathbf{t}^T)(\mathbf{Xw} - \mathbf{t})$$

$$\mathcal{L} = ((\mathbf{Xw})^T - \mathbf{t}^T)(\mathbf{Xw} - \mathbf{t})$$

$$\mathcal{L} = (\mathbf{Xw})^T \mathbf{Xw} - \mathbf{t}^T \mathbf{Xw} - (\mathbf{Xw})^T \mathbf{t} + \mathbf{t}^T \mathbf{t}$$

$$\mathcal{L} = \mathbf{w}^t \mathbf{X}^T \mathbf{Xw} - 2\mathbf{w}^T \mathbf{X}^T \mathbf{t} + \mathbf{t}^T \mathbf{t}$$

Now we an expression for the total training loss that is easier to work with. Then equating the derivative of this to zero we get:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = 2\mathbf{X}^T \mathbf{Xw} - 2\mathbf{X}^T \mathbf{t} = 0$$

by dividing with 2 and then adding $\mathbf{X}^T \mathbf{t}$ on both sides of the equation we get:

$$\mathbf{X}^T \mathbf{Xw} = \mathbf{X}^T \mathbf{t}$$

Now we have an expression where we can find $\hat{w}$ using the numpy.linalg.solve and we notice that it is similar to the one that was derived from the average loss. We can also manipulate the expression a bit more and get

$$\hat{\mathbf{w}} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{t}$$

Now we also have an expression to find the optimal least squares parameter values $\hat{\mathbf{w}}$ for the total training loss

Appendix

Housing_1.py

```python
import numpy as np
import matplotlib.pyplot as plt

# load data
train_data = np.loadtxt("boston_train.csv", delimiter=",")
test_data = np.loadtxt("boston_test.csv", delimiter=",")
X_train, t_train = train_data[:,:-1], train_data[:,-1]
X_test, t_test = test_data[:,:-1], test_data[:,-1]
# make sure that we have N-dimensional Numpy arrays (ndarray)
t_train = t_train.reshape((len(t_train), 1)) #house prices
t_test = t_test.reshape((len(t_test), 1))
print("Number of training instances: %i" % X_train.shape[0])
print("Number of test instances: %i" % X_test.shape[0])
print("Number of features: %i" % X_train.shape[1])

# (a) compute mean of prices on training set -
Avg_price = sum(t_train)/len(t_train) #numpy.mean #22.01660079


# (b) RMSE function
def rmse(t, tp):
    error = t - tp
    square = error ** 2
    mean = np.mean(square)
    root = np.sqrt(mean)
    return root

estimates = np.full((len(t_test), 1), Avg_price)
rmse(t_test, estimates)

# (c) visualization of results
plt.title("2D scatter plot(\"true house prices\" vs. \"estimates\")")
plt.scatter(estimates, t_test)
plt.ylabel('true house prices')
plt.xlabel('estimates')
plt.savefig("2DscatterPlot.png")
```

Housing_2.py

```
import numpy as np
import pandas as pd
import linreg
import matplotlib.pyplot as plt

# load data
train_data = np.loadtxt("boston_train.csv", delimiter=",")
test_data = np.loadtxt("boston_test.csv", delimiter=",")
X_train, t_train = train_data[:,:-1], train_data[:,-1]
X_test, t_test = test_data[:,:-1], test_data[:,-1]
# make sure that we have N-dimensional Numpy arrays (ndarray)
t_train = t_train.reshape((len(t_train), 1))
t_test = t_test.reshape((len(t_test), 1))
print("Number of training instances: %i" % X_train.shape[0])
print("Number of test instances: %i" % X_test.shape[0])
print("Number of features: %i" % X_train.shape[1])

# (b) fit linear regression using only the first feature
model_single = linreg.LinearRegression()
model_single.fit(X_train[:,0], t_train)
print("Single model weight: %a\n"% model_single.w)

# (c) fit linear regression model using all features
model_all = linreg.LinearRegression()
model_all.fit(X_train, t_train)
#for-loop to make more readable output, simply type print(model_all.w) if the
#look of the output doesn't matter
for i in range(len(model_all.w)):
    print("All model weight w%a: %a\n"% (i, model_all.w[i][0]))

# (d) evaluation of results
def rmse(t, tp):
    error = t - tp
    square = error ** 2
    mean = np.mean(square)
    root = np.sqrt(mean)
    return root

print("RMSE of all: %a"% rmse(t_test, model_all.predict(X_test)))

print("RMSE of single: %a"% rmse(t_test, model_single.predict(X_test[:,0])))

plt.subplot(212)
plt.title("2D scatter plot(\"true house prices\" vs. \"estimates\") - ALL")
plt.scatter(model_all.predict(X_test), t_test, color='cyan', label="All")
plt.ylabel('true house prices')
plt.legend(loc=2)
plt.xlim(-18,45)
plt.savefig("2DscatterPlotModelALL.png")

plt.subplot(211)
plt.title("2D scatter plot(\"true house prices\" vs. \"estimates\") - SINGLE")
plt.scatter(model_single.predict(X_test[:,0]), t_test, color='purple', label="Single")
plt.ylabel('true house prices')
plt.xlabel('Predictions')
plt.legend(loc=2)
plt.xticks([-18,0,10,20,30,40,45])
ax = plt.gca()
ax.xaxis.set_visible(False)
plt.savefig("2DscatterPlotModelSingle.png")
```

linreg.py

```
import numpy

# NOTE: This template makes use of Python classes. If
# you are not yet familiar with this concept, you can
# find a short introduction here:
# http://introtopython.org/classes.html

class LinearRegression():
    """
    Linear regression implementation.
    """

    def __init__(self):

        pass

    def fit(self, X, t):
        """
        Fits the linear regression model.

        Parameters
        ----------
        X : Array of shape [n_samples, n_features]
        t : Array of shape [n_samples, 1]
        """

        # make sure that we have Numpy arrays; also
        # reshape the target array to ensure that we have
        # a N-dimensional Numpy array (ndarray), see
        # https://docs.scipy.org/doc/numpy-1.13.0/reference/arrays.ndarray.html
        X = numpy.array(X).reshape((len(X), -1))
        t = numpy.array(t).reshape((len(t), 1))

        # prepend a column of ones
        ones = numpy.ones((X.shape[0], 1))
        X = numpy.concatenate((ones, X), axis=1)

        # compute weights
        a = numpy.dot(X.T,X) #X.T is X transposed
        b = numpy.dot(X.T,t)
        self.w = numpy.linalg.solve(a, b)

        # (2) TODO: Make use of numpy.linalg.solve instead!
        # Reason: Inverting the matrix is not very stable
        # from a numerical perspective; directly solving
        # the linear system of equations is usually better.

    def predict(self, X):
        """
        Computes predictions for a new set of points.

        Parameters
        ----------
        X : Array of shape [n_samples, n_features]

        Returns
        -------
        predictions : Array of shape [n_samples, 1]
        """

        # (1) TODO: Compute the predictions for the
        # array of input points

        X = numpy.array(X).reshape((len(X), -1))

        ones = numpy.ones((X.shape[0], 1))
        X = numpy.concatenate((ones, X), axis=1)

        predictions = numpy.dot(X, self.w)
        #ones = numpy.ones((predictions.shape[0], 1))
        #predictions = numpy.concatenate((ones, predictions), axis=1)
        return predictions
```