A1

MASD 2021
Department of Computer Science
University of Copenhagen

Mikkel/pwz854 | partners = mfh144 & qfh987

29/11/2021

Contents

Exercise 1

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

Create new expression using vectors and matrices, assuming that α_n is a fixed value and that A is a diagonal matrix. This means that $A = A^T$

$$\mathcal{L} = \frac{1}{N}\mathbf{A}(\mathbf{Xw} - \mathbf{t})^T(\mathbf{Xw} - \mathbf{t})$$

Rewrite the expression to

$$\mathcal{L} = \frac{1}{N}\mathbf{A}((\mathbf{Xw})^T - \mathbf{t}^T)(\mathbf{Xw} - \mathbf{t})$$

$$\mathcal{L} = \frac{1}{N}\mathbf{A}(\mathbf{Xw})^T\mathbf{Xw} - \frac{1}{N}\mathbf{At}^T\mathbf{Xw} - \frac{1}{N}\mathbf{A}(\mathbf{Xw})^T\mathbf{t} + \frac{1}{N}\mathbf{At}^T\mathbf{t}$$

Now using that we know $A = A^T$ we can rewrite this to

$$\mathcal{L} = \frac{1}{N}(\mathbf{AXw})^T\mathbf{Xw} - \frac{1}{N}(\mathbf{At})^T\mathbf{Xw} - \frac{1}{N}(\mathbf{AXw})^T\mathbf{t} + \frac{1}{N}\mathbf{At}^T\mathbf{t}$$

$$\mathcal{L} = \frac{1}{N}\mathbf{w}^T\mathbf{X}^T\mathbf{AXw} - \frac{1}{N}\mathbf{t}^T\mathbf{AXw} - \frac{1}{N}\mathbf{w}^T\mathbf{X}^T\mathbf{A}^T\mathbf{t} + \frac{1}{N}\mathbf{At}^T\mathbf{t}$$

$$\mathcal{L} = \frac{1}{N}\mathbf{w}^T\mathbf{X}^T\mathbf{AXw} - \frac{2}{N}\mathbf{w}^T\mathbf{X}^T\mathbf{At} + \frac{1}{N}\mathbf{At}^T\mathbf{t}$$

Now taking the derivative and equating it to 0

$$\frac{\partial\mathcal{L}}{\partial\mathbf{w}} = \frac{2}{N}\mathbf{X}^T\mathbf{AXw} - \frac{2}{N}\mathbf{X}^T\mathbf{At} = 0$$

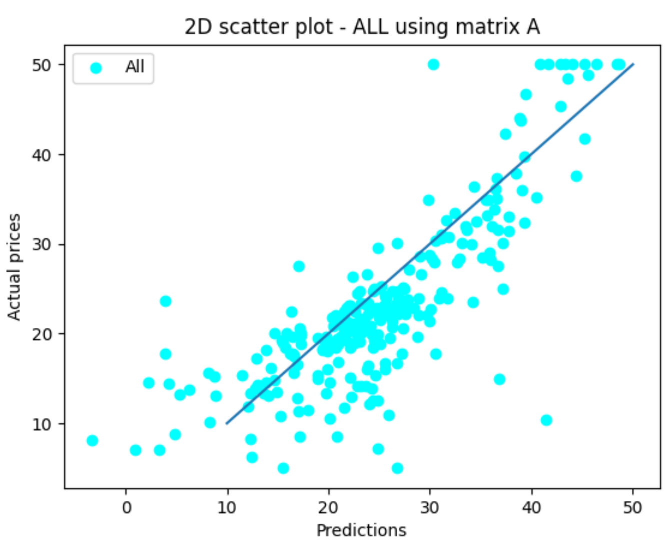
Multiplying this by N, dividing by 2 and adding $\mathbf{X}^T\mathbf{At}$ we get

$$\mathbf{X}^T\mathbf{AXw} = \mathbf{X}^T\mathbf{At}$$

Lastly premultiply both sides with $(\mathbf{X}^T\mathbf{AX})^{-1}$ so we get

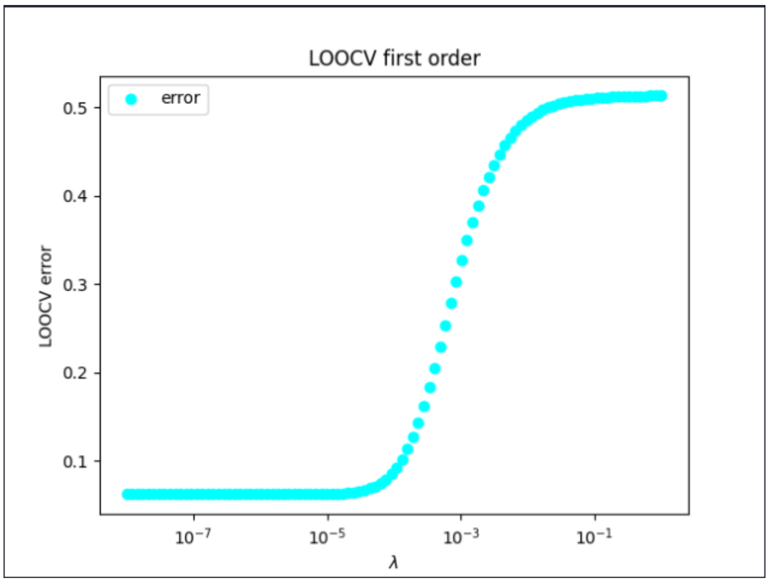
$$\mathbf{w} = (\mathbf{X}^T\mathbf{AX})^{-1}\mathbf{X}^T\mathbf{At}$$

b) We see that the extra weight $\alpha_n = t_n^2$. This means that the higher the true house prices are, the more affection the weight will have and the more precise our predictions will become, since we try to minimise our loss function. This however also means that the predictions for the lower prices will be less precise. We see on the plot that the high prices has a bigger tendency to get near the straight line while the lower prices are more scattered out, as expected.



Exercise 2

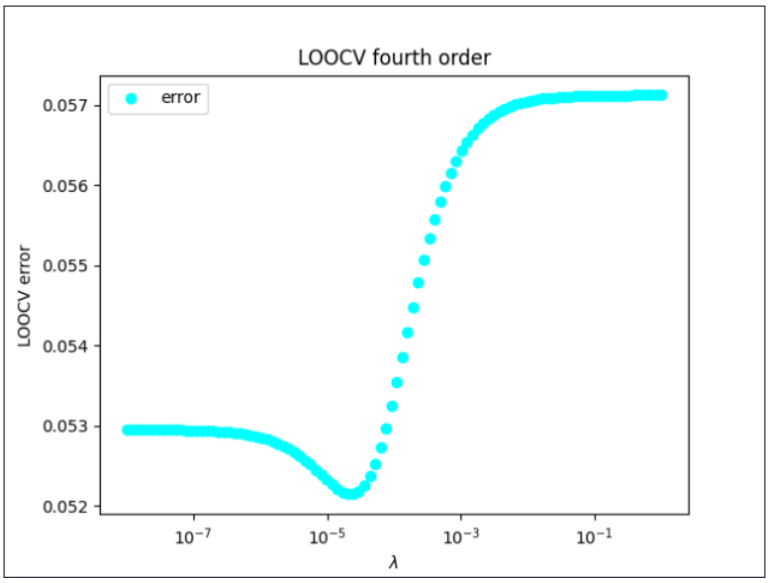
a) The plot looks as the following:



and the best lambda value was found to be 3.4304692863149193$e-07$.
The regression coefficients **w** was found to be:

```
w of best lam: ([ 3.63776282e+01], [-1.33110046e-02])
w of lam 0: ([ 3.64153229e+01], [-1.33303056e-02])
```

b) The plot looks as the following:



and the best lambda value was found to be $2.0565123083486515e - 05$.
The regression coefficients **w** was found to be:

```
w of best lam:
        ([ 1.88300350e-02],
        [ 9.11277821e+00],
        [-1.38600662e-02],
        [ 7.03376807e-06],
        [-1.19035006e-09])
w of lam 0:
        ([ 3.35833540e-01],
        [ 1.22969323e+01],
        [-1.87537476e-02],
        [ 9.54015317e-06],
        [-1.61816192e-09])
```

Exercise 3

a) The life span x of a chip(in years) can be modelled with the following CDF

$$F(x) = \begin{cases} 0 & x \leq 0 \\ \text{1-e}^{-\beta x^\alpha} & x\text{>}0 \end{cases}$$

The PDF f(x) can be found by differentiating the CDF. Note that the first part($x \leq 0$) is just 0 so differentiating here wont make a difference and hence the start of the PDF will be

$$f(x) = \begin{cases} 0 & x \leq 0 \\ (\text{d 1-e}^{-\beta x^\alpha})\dfrac{1}{dx} & x\text{>}0 \end{cases}$$

Now it is needed to differentiate $1 - e^{-\beta x^\alpha}$ to obtain the rest of the PDF

$$\frac{d}{dx}(1 - e^{-\beta x^\alpha})$$

first apply the sum the rule

$$\frac{d}{dx}(1 - e^{-\beta x^\alpha}) = \frac{d}{dx}(1) - \frac{d}{dx}(e^{-\beta x^\alpha})$$

now apply the chain rule

$$-\frac{d}{dx}(e^{-\beta x^\alpha}) = -e^{-\beta x^\alpha}\frac{d}{dx}(-\beta x^\alpha) = -e^{-\beta x^\alpha} - \beta\frac{d}{dx}(x^\alpha) = -e^{-\beta x^\alpha} \cdot (-\beta\alpha x^{\alpha-1})$$

This results in the full PDF f(x) looking as the following

$$f(x) = \begin{cases} 0 & x \leq 0 \\ \text{e}^{-\beta x^\alpha} \cdot \beta\alpha x^{\alpha-1} & x\text{>}0 \end{cases}$$

b) A way to figure out the probability of the chip lasting more than four years would be to use the PDF so we have

$$P(X > 4) = \int_4^\infty f(x)dx = F(\infty) - F(4)$$

Since we already have the CDF we can simply input the values for x, α, β so we get

$$(1 - e^{-\frac{1}{4}\infty^2}) - (1 - e^{-\frac{1}{4}4^2})$$

which is similar to

$$1 - e^{-\frac{1}{4}\infty^2} - 1 + e^{-\frac{1}{4}4^2}$$

which can be written as

$$e^{-\frac{16}{4}} - e^{-\frac{1}{4}\infty^2}$$

now using that $\frac{1}{x^1} = x^{-1}$ we get

$$e^{-\frac{16}{4}} - \frac{1}{e^{\frac{1}{4}\infty^2}}$$

now we can see that $e^{\frac{1}{4}\infty^2}$ will become infinitely large resulting in $\frac{1}{e^{\frac{1}{4}\infty^2}}$ becoming infinitely small and therefore we can treat it as a 0 so we get

$$P(X > 4) = e^{-\frac{16}{4}}$$

Now the same can be done to figure out the probability for the interval [5;10] years so we get

$$P(5 \leq X \leq 10) = \int_5^{10} f(x)dx = F(10) - F(5)$$

repeating the same method so we have

$$(1 - e^{-\frac{1}{4}10^2}) - (1 - e^{-\frac{1}{4}5^2})$$

$$e^{-\frac{25}{4}} - e^{-\frac{100}{4}}$$

So the probability of the chip lasting longer than 5 years but shorter than 10 is $P(5 \leq X \leq 10) = 0.00193045412$

c) The median m is defined as the value that satisfies that $P(X \geq m) \geq \frac{1}{2}$ and $P(X \leq m) \geq \frac{1}{2}$. A way to find this median m is to solve the equation

$$\int_0^m (f(x)dx = \frac{1}{2} = F(m) - F(0)$$

Inputting these values in the CDF we get

$$e^{-\beta \cdot 0^\alpha} - e^{-\beta m^\alpha} = \frac{1}{2}$$

which, since $\alpha > 0$, is the same as

$$1 - e^{-\beta \cdot m^\alpha} = \frac{1}{2}$$

subtract one on both sides and multiply by -1 after

$$e^{-\beta \cdot m^\alpha} = \frac{1}{2}$$

now taking the natural logarithm on both sides we get

$$-\beta \cdot m^\alpha = \ln \frac{1}{2}$$

Divide by $-\beta$ on both sides

$$m^{\alpha} = \frac{\ln \frac{1}{2}}{-\beta}$$

Take the α'th root on both sides

$$m = \sqrt[\alpha]{\frac{\ln \frac{1}{2}}{-\beta}}$$

Now we have en expression for the median being $m = \sqrt[\alpha]{\frac{\ln \frac{1}{2}}{-\beta}}$. Given specific Values for α and β one can figure out how large the median of a life span exactly is.

Exercise 4

a)

Appendix

weigh.py

```python
import numpy as np
import pandas as pd
import linweighreg  as 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]

# fit linear regression model using all features
model_all = linreg.LinearRegression()
model_all.fit(X_train, t_train)
for i in range(len(model_all.w)):
    print("All model weight w%a: %a"% (i, model_all.w[i][0]))

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

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

plt.subplot()
plt.title("2D scatter plot - ALL using matrix A")
plt.plot([10,20,30,40,50],[10,20,30,40,50])
plt.scatter(model_all.predict(X_test), t_test, color='cyan', label="All")
plt.ylabel('Actual prices')
plt.xlabel('Predictions')
plt.legend(loc=2)
#plt.xlim(-18,45)
plt.savefig("2DscatterPlotModelALL.png")
```

linweighreg.py

```python
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 A
        A = numpy.diag(t.T[0] ** 2) #
        # compute weights
        a = numpy.dot(X.T, numpy.dot(A,X)) #left side
        b = numpy.dot(X.T, numpy.dot(A,t)) #right side

        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
```

linreg.py

```
from matplotlib.pyplot import axes
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, Lam, k):
        """
        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))

        constx = X #copy X, so we can alter it without losing its values
        #k'th order matrix to get k'th order polynomial
        if (k>1):
            X = X ** 0 #0'th order polynomial, column of ones
            for i in range(1,k+1):
                X = numpy.concatenate((X, constx ** i), axis=1)
        else:
            ones = numpy.ones((X.shape[0], 1))
            X = numpy.concatenate((ones, X), axis=1)

        #create identity matrix
        I = numpy.eye(len(X[0]))

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

    #
    def predict(self, X, k):
        """
        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]
        """

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

        constx = X
        if (k>1):
            X = X ** 0 #0'th order polynomial, column of ones
            for i in range(1,k+1):
                X = numpy.concatenate((X, constx ** i), axis=1)
        else:
            ones = numpy.ones((X.shape[0], 1))
            X = numpy.concatenate((ones, X), axis=1)

        predictions = numpy.dot(X, self.w)

        return predictions

    #
    def LOOCV(self, x, t, lamValue, k):
        x = numpy.array(x).reshape((len(x), -1))
        t = numpy.array(t).reshape((len(t), 1))

        errors = []


        for n in range(len(x)):
            self.fit(numpy.delete(x,n), numpy.delete(t,n), lamValue, k)#find w_{-n}
            error = t[n] - self.predict(x[n], k)
            errors = numpy.append(errors, error)

        square = errors ** 2
        mean = numpy.mean(square)
        return mean
```

LOO-CV.py

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

#load data
raw = np.genfromtxt('men-olympics-100.txt', delimiter=' ')

FPRunTimes, InputVar = raw[:, 1], raw[:, 0]

lam = np.logspace(-8, 0, 100, base=10)
#print(lam)

regularised_model = linreg.LinearRegression()

def GetBestLamb(x, t, k):
    loss = regularised_model.LOOCV(x, t, lam[0], k)
    for i in range(1,100):
        lossOfi = regularised_model.LOOCV(x, t, lam[i], k)
        if loss > lossOfi:
            loss = lossOfi
            result = lam[i]
            #print("lamInside = %.10f"% lam[i])
    return result

print("lam = %a" % GetBestLamb(InputVar,FPRunTimes,1))
regularised_model.fit(InputVar, FPRunTimes, GetBestLamb(InputVar,FPRunTimes,1), 1)
print("w of best lam: %a"% regularised_model.w)
regularised_model.fit(InputVar, FPRunTimes, lam[0], 1)
print("w of lam 0: %a"% regularised_model.w)


LOOCVList = np.array([regularised_model.LOOCV(InputVar, FPRunTimes, i, 1) for i in lam])

plt.subplot(211)
plt.title("LOOCV first order")
plt.scatter(lam,LOOCVList , color='cyan', label="error")
plt.ylabel('LOOCV error')
plt.xlabel("$\lambda$")
plt.legend(loc=2)
ax = plt.gca()
ax.xaxis.set_visible(False)
plt.xscale("log")
#plt.savefig("LOOCVofFirstOrder.png")

#b
fourth_model = linreg.LinearRegression()
LOOCVList2 = np.array([fourth_model.LOOCV(InputVar, FPRunTimes, i, 4) for i in lam])

print("lam = %a" % GetBestLamb(InputVar,FPRunTimes,4))
fourth_model.fit(InputVar, FPRunTimes, GetBestLamb(InputVar,FPRunTimes,4), 4)
print("w of best lam: %a"% fourth_model.w)
fourth_model.fit(InputVar, FPRunTimes, lam[0], 4)
print("w of lam 0: %a"% fourth_model.w)

plt.subplot(212)
plt.title("LOOCV fourth order")
plt.scatter(lam,LOOCVList2 , color='cyan', label="error")
plt.ylabel('LOOCV error')
plt.xlabel("$\lambda$")
plt.legend(loc=2)
plt.xscale("log")
plt.savefig("LOOCVofFourthOrder.png")
```