A2 - MAD

Julian Wulff

November 2019

Exercise 1

a)

In order to derive the optimal least squares parameter value $\hat{\mathbf{w}}$ of the function:

$$\mathcal{L} = \frac{1}{N}\sum_{n=1}^{N}\alpha_n(f(\mathbf{x}_n;\mathbf{w}) - t_n)^2 = \frac{1}{N}\sum_{n=1}^{N}\alpha_n(\mathbf{w}^T\mathbf{x}_n - t_n)^2$$

We will start by rewriting the function in terms of various vectors and matrices, which will be easier to manipulate. We already now how to express

$$\mathcal{L} = \frac{1}{N}\sum_{n=1}^{N}(\mathbf{w}^T\mathbf{x}_n - t_n)^2 = \frac{1}{N}(\mathbf{X}\mathbf{w} - \mathbf{t})^\top(\mathbf{X}\mathbf{w} - \mathbf{t})$$

In order to include α_n in this expression we start by defining a matrix $\mathbf{A}$ as diagonal matrix containing the weights $\alpha_1, \ldots, \alpha_n$, in order to multiply these onto our expression. In order to let each individual α_n appear one time each in the final scalar we will multiply $\mathbf{A}$ on to the expression as showm below:

$$\mathcal{L}\frac{1}{N}((\mathbf{X}\mathbf{w} - \mathbf{t})^\top\mathbf{A}(\mathbf{X}\mathbf{w} - \mathbf{t}))$$

Which gives us the desired expression. We will now multiply out the parenthesis in order continue:

$$\begin{aligned}
\mathcal{L} &= \frac{1}{N}(\mathbf{X}\mathbf{w} - \mathbf{t})^\top\mathbf{A}(\mathbf{X}\mathbf{w} - \mathbf{t}) && (1)\\
&= \frac{1}{N}((\mathbf{X}\mathbf{w}^\top) - \mathbf{t}^\top)\mathbf{A}(\mathbf{X}\mathbf{w} - \mathbf{t}) && (2)\\
&= \frac{1}{N}((\mathbf{X}\mathbf{w}^\top)\mathbf{A} - \mathbf{t}^\top\mathbf{A})(\mathbf{X}\mathbf{w} - \mathbf{t}) && (3)\\
&= \frac{1}{N}(\mathbf{X}\mathbf{w}^\top)\mathbf{A}\mathbf{X}\mathbf{w} - \frac{1}{N}\mathbf{t}^\top\mathbf{A}\mathbf{X}\mathbf{w} - \frac{1}{N}(\mathbf{X}\mathbf{w})^\top\mathbf{A}\mathbf{t} + \frac{1}{N}\mathbf{t}^\top\mathbf{A}\mathbf{t} && (4)\\
&= \frac{1}{N}\mathbf{w}^\top\mathbf{X}^\top\mathbf{A}\mathbf{X}\mathbf{w} - \frac{1}{N}\mathbf{t}^\top\mathbf{A}\mathbf{X}\mathbf{w} - \frac{1}{N}\mathbf{w}^\top\mathbf{X}^\top\mathbf{A}\mathbf{t} + \frac{1}{N}\mathbf{t}^\top\mathbf{A}\mathbf{t} && (5)
\end{aligned}$$

Since the two terms $\mathbf{t}^\top \mathbf{AXw}$ and $\mathbf{w}^\top \mathbf{X}^\top \mathbf{At}$ are the transpose of one another, and also scalars, we conclude that they must be the same and therefore we combine them.

$$\mathcal{L} = \frac{1}{N}\mathbf{w}^\top \mathbf{X}^\top \mathbf{AXw} - \frac{2}{N}\mathbf{w}^\top \mathbf{X}^\top \mathbf{At} + \frac{1}{N}\mathbf{t}^\top \mathbf{At} \tag{6}$$

$$= \frac{1}{N}(\mathbf{w}^\top \mathbf{X}^\top \mathbf{AXw} - 2\mathbf{w}^\top \mathbf{X}^\top \mathbf{At} + \mathbf{t}^\top \mathbf{At}) \tag{7}$$

From this expression we find the derivative using the identities in Table 1.4.[1] Loking at the first term of $\mathcal{L}$ we have $\mathbf{w}^\top \mathbf{X}^\top \mathbf{AXw}$. We know that $\mathbf{X}^\top \mathbf{X}$ gives a symmetric matrix. Since A is a diagonal matrix we now that $\mathbf{X}^\top \mathbf{AX}$ must still be symmetric. For this reason we can use the 4th identity from table 1.4 which gives us $2\mathbf{X}^\top \mathbf{AXw}$. Looking at the second term of $\mathcal{L}$ we have $2\mathbf{w}^\top \mathbf{X}^\top \mathbf{At}$. We observe that $\mathbf{X}^\top \mathbf{At}$ equals a vector that does not depend on $\mathbf{w}$. Using the 1st identity from table 1.4 we get $\mathbf{X}^\top \mathbf{At}$. Looking at the last term of $\mathcal{L}$ we see that $\mathbf{t}^\top \mathbf{At}$ does not depend on $\mathbf{w}$, which is why it disappears when deriving with respect to $\mathbf{w}$. From this we get the following derivative and solve for 0:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = \frac{2}{N}\mathbf{X}^\top \mathbf{AXw} - \frac{2}{N}\mathbf{X}^\top \mathbf{At} = 0 \tag{8}$$

$$\mathbf{X}^\top \mathbf{AXw} = \mathbf{X}^\top \mathbf{At} \tag{9}$$

In order to isolate $\mathbf{w}$ we will cancel $\mathbf{X}^\top \mathbf{AX}$ out by premultiplying its inverse of both sides giving us:

$$\mathbf{Iw} = (\mathbf{X}^\top \mathbf{AX})^{-1}\mathbf{X}^\top \mathbf{At} \tag{10}$$

As $\mathbf{Iw} = \mathbf{w}$ we are left with a matrix equation for $\hat{\mathbf{w}}$, the value of $\mathbf{w}$ that minimises the loss:

$$\hat{\mathbf{w}} = (\mathbf{X}^\top \mathbf{AX})^{-1}\mathbf{X}^\top \mathbf{At} \tag{11}$$

b)

Since we use $\alpha_n = t_n^2$ as the additional weights i expect that the big t_n values will have a bigger impact on the w_n values such that the spread will be less for high t_n values and and bigger for small t_n values. This is indeed what we observe when we look at figure 1.

See section 1.2 to see code for the solution.

[1] S. Rogers Mark Girolami, A First Course In Machine Learning, 2nd edtion, p.23

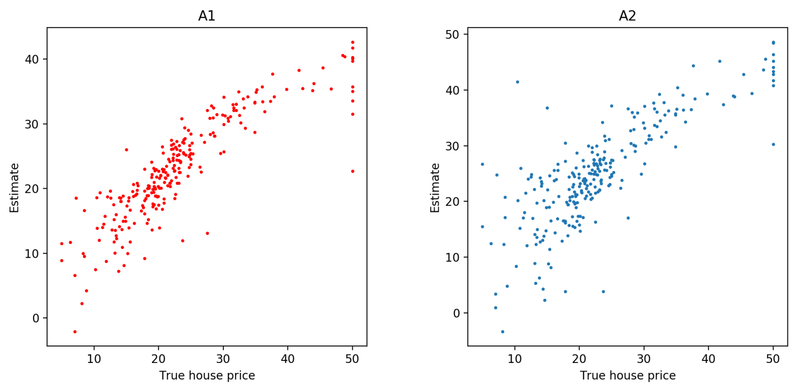

Figure 1: Figures containing scatter plot for true house prices vs estimated house prices. One the left is the plot from assignment 1, to the left is the plot using the weighted average loss found using the deriation found in a).

Exercise 2

a)

The best value of $lamda = 0.0000003430$ and its $loss = 0.0624312040$
For this model $w0 = 36.3776282284$ and $w1 = -0.0133110046$
For the model with $lamda = 0$, $w0 = 36.4164559025$ and $w1 = -0.0133308857$

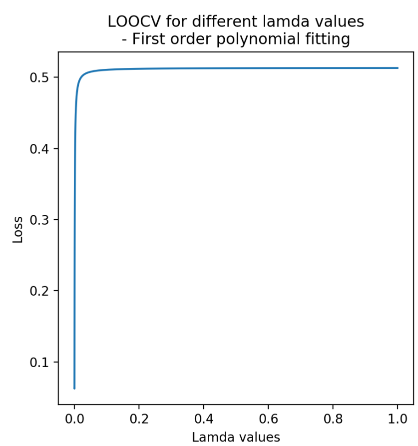

Figure 2

b)

The best value of $lamda = 0.0000205651$ and its $loss = 0.0521430656$
For this model $w0 = 0.0188300350$, $w1 = 9.1127782080$, $w2 = -0.0138600662$, $w3 = 0.0000070338$, $w4 = -0.0000000012$

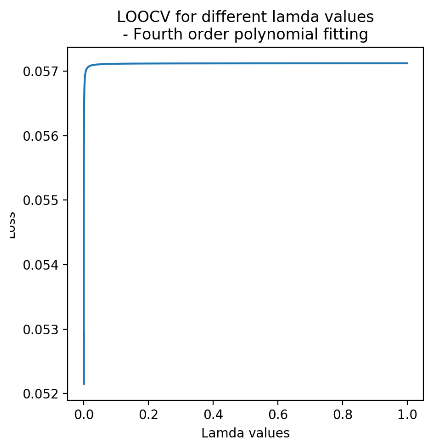

Figure 3

Exercise 3

a)

We are given the CDF:

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

Since the CDF $F(X)$ is a continuous functions, we obtain the PDF by taking its derivative. Therefore we have

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

b)

With $\alpha = 2$ and $\beta = \frac{1}{4}$. To find the probability that the chip works long than four years we use the CDF to compute

$$1 - F(4) \approx 0.01831563888873422$$

To find the probability that the chip stops working in the time interval $[5; 10]$ years we use the CDF to compute

$$F(10) - F(4) \approx 0.0019304541223398308$$

Exercise 4

a)

Given a person with no history of conviction (NC). The expected mean sentence duration he will have to spend in prison if

- NC talks to the police: $0.002 * (1 - 0.5) * 5 * 0.75 = 0.00375$ years
- NC stays silent: $0.001 * (1 - (0.5 * \frac{1}{4})) * 5 = 0.004375$ years

Given a person with a history of conviction (C). The expected mean sentence duration he will have to spend in prison is:

- C talks to the police: $0.005 * (1 - 0.1) * 0.5 * 0.75 = 0.003125$
- C stays silent: $0.001 * (1 - (0.1 * \frac{1}{4})) * 5 = 0.004875$

1 Appendix

1.1 Exercise 1 files

```python
import numpy as np

class LinearWeightedRegression():
    def __init__(self):

        pass

    def fit(self, X, t):
        # Make sure that we have N-dimensional Numpy arrays (
    ndarray)
        X = np.array(X).reshape((len(X), -1))
        t = np.array(t).reshape((len(t), 1))

        # Prepend a column of ones til the X matrix
        ones = np.ones((X.shape[0], 1))
        X = np.concatenate((ones, X), axis=1)

        # Transform the w vector into a diagonal vector
        A = np.diag(t[:,0] ** 2)

        self.w = np.linalg.solve(X.T @ A @ X, X.T @ A @ t)

    def predict(self, X):
        X = np.array(X).reshape((len(X), -1))

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

```
        predictions = np.dot(X, self.w)

        return predictions
```

lineweighreg.py

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

# Load data
train_data = np.loadtxt("boston_train.csv", delimiter=",")
test_data = np.loadtxt("boston_test.csv", delimiter=",")

# Choose all rows and all cols minus the last, choose all rows and
    the last col
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))

# Fit the model
model = linweighreg.LinearWeightedRegression()
model.fit(X_train, t_train)

# Make predictions
predictions = model.predict(X_test)

# Plot the data
plt.figure(figsize=(5,5))
plt.scatter(t_test, predictions, s=3)
plt.title("A2")
plt.xlabel("True house price")
plt.ylabel("Estimate")
plt.show()
```

exercise1.py

1.2 Exercise 2 files

```
import numpy as np

class LinearRegression():
    def __init__(self, lam=0.0):

        self.lam = lam

    def fit(self, X, t):

        X = np.array(X).reshape((len(X), -1))
        t = np.array(t).reshape((len(t), 1))

```

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

        diag = self.lam * len(X) * np.identity(X.shape[1])
        a = np.dot(X.T, X) + diag
        b = np.dot(X.T, t)
        self.w = np.linalg.solve(a,b)

    def predict(self, X):
        X = np.array(X).reshape((len(X), -1))

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

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

        return predictions
```

linereg.py

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

# DATA
data = np.loadtxt("men-olympics-100.txt")

X, t = data[:, 0], data[:, 1]
X = X.reshape((len(X), 1))
t = t.reshape((len(t), 1))

lamdaValues = np.logspace(-8, 0, 100, base=10)

def LOOCV(X, t, l):
    errors = []
    for lam in lamdaValues:
        model = linereg.LinearRegression(lam=lam)
        modelError = 0
        for i in range(X.shape[0]):
            X_train = np.delete(X, i, 0)
            t_train = np.delete(t, i, 0)

            X_pred = X[i]
            X_pred = X_pred.reshape((-1, len(X_pred)))
            t_pred = t[i]
            t_pred = t_pred.reshape((len(t_pred), 1))

            model.fit(X_train, t_train)

            prediction = model.predict(X_pred)

            error = (prediction - t_pred)**2
            modelError += error[0][0]
        modelError = modelError / X.shape[0]
        errors.append(modelError)
    return errors

```

```
# First order polynomial
firstOrderLOOCV = LOOCV(X, t, lamdaValues)
bestValueindex = np.argmin(firstOrderLOOCV)
bestLamda = lamdaValues[bestValueindex]

model = linereg.LinearRegression(lam=bestLamda)
model.fit(X, t)
print("======= First order polynomial fitting =======")
print("The best value of lamda=%.10f and its loss=%.10f" %
      (bestLamda, firstOrderLOOCV[bestValueindex]))
print("For this model w0=%.10f and w1=%.10f \n" %
      (model.w[0][0], model.w[1][0]))

modelLam0 = linereg.LinearRegression()
modelLam0.fit(X, t)
print("For the model with lamda=0 w0=%.10f and w1=%.10f\n" %
      (modelLam0.w[0][0], modelLam0.w[1][0]))

# Plot data
plt.figure(figsize=(5,5))
plt.plot(lamdaValues, firstOrderLOOCV)
plt.title("LOOCV for different lamda values\n- First order polynomial fitting")
plt.xlabel("Lamda values")
plt.ylabel("Loss")
plt.show()

# Fourth order polynomial
def augment(X, max_order):
    X_augmented = X

    for i in range(2, max_order+1):
        X_augmented = np.concatenate([X_augmented, X**i], axis=1)
    return X_augmented


Xnew = augment(X, 4)
fourthOrderLOOCV = LOOCV(Xnew, t, lamdaValues)
bestValueindex = np.argmin(fourthOrderLOOCV)
bestLamda = lamdaValues[bestValueindex]

modelFourthOrder = linereg.LinearRegression(lam=bestLamda)
modelFourthOrder.fit(Xnew, t)
print("======= Fourth order polynomial fitting =======")
print("The best value of lamda=%.10f and its loss=%.10f" %
      (bestLamda, fourthOrderLOOCV[bestValueindex]))
print("For this model w0=%.10f, w1=%.10f, w2=%.10f, w3=%.10f, w4=%.10f \n" %
      (modelFourthOrder.w[0][0], modelFourthOrder.w[1][0],
       modelFourthOrder.w[2][0], modelFourthOrder.w[3][0],
    modelFourthOrder.w[4][0]))

# Plot data
plt.figure(figsize=(5,5))
plt.plot(lamdaValues, fourthOrderLOOCV)
plt.title("LOOCV for different lamda values\n- Fourth order polynomial fitting")
```

```
plt.xlabel("Lamda values")
plt.ylabel("Loss")
plt.show()
```

exercise2.py