MAD Assignment 1

Andreas V. W. Zacchi (nzl169)

November 29, 2024

Exercise 1

a)

The weighted total loss is given by

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

This can be rephrased to matrix-vector form in the same way as in Section 1.3 in Rogers and Girolami:

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

Now before calculating the gradient we can expand $(Xw - t)^T A(Xw - t)$:

$$\begin{aligned}(Xw - t)^T A(Xw - t) &= (Xw)^T A(Xw) - (Xw)^T A(t) - t^T A(Xw) + t^T At \\ &= w^T X^T AXW - (Xw)^T A(t) - t^T A(Xw) + t^T At \\ &= w^T X^T AXW - w^T X^T At - t^T AXw + t^T At\end{aligned}$$

And since $(w^T X^T At)^T = t^T AXw$ we can further simplify to

$$w^T X^T AXw - 2w^T X^T At + t^T At$$

Inserting it back into the original expression we get:

$$\begin{aligned}\mathcal{L}(w) &= \frac{1}{N}(w^T X^T AXW - 2w^T X^T At + t^T At) \\ &= \frac{1}{N}w^T X^T AXW - \frac{2}{N}w^T X^T At + \frac{1}{N}t^T At\end{aligned}$$

Now we need to calculate the gradient and can make use of Table 1.4 from Rogers and Girolami:

$$\begin{aligned}\nabla\mathcal{L}(w) &= \nabla\left(\frac{1}{N}w^T X^T AXW - \frac{2}{N}w^T X^T At + \frac{1}{N}t^T At\right) \\ &= \nabla\left(\frac{1}{N}w^T X^T AXW - \frac{2}{N}w^T X^T At\right) \\ &= \frac{2}{N}X^T AXW - \frac{2}{N}X^T At\end{aligned}$$

Now we need to set the gradient equal to zero and solve for w, we can ignore the constants as they only affect the magnitude of the solution:

$$
\begin{aligned}
X^T A X w - X^T A t = 0 \Rightarrow \\
X^T A X w = X^T A t \Rightarrow \\
w = (X^T A X)^{-1} X^T A t
\end{aligned}
$$

b)

See Figure 1 for the scatter plot. Using the additional weight I would expect the model to be more focused towards greater t values, which is also seen on the figure where predictions greater than 30 is more close to the actual price. Comparing using the RMSE of the new model against the old model on the same data the new one performs worse.

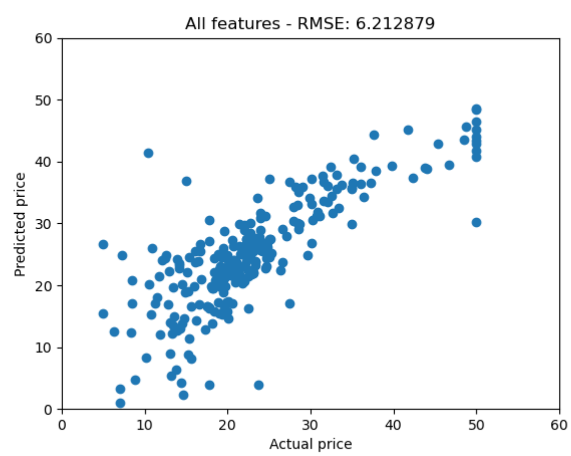

Figure 1: Scatter plot using $\alpha_n = t_n^2$

Exercise 2

a)

The best calculated lambda value is $\lambda = 0.0000097701$ with coefficients:

$$w = (36.3755018, -0.0133099158)^T$$

The calculated coefficients for $\lambda = 0$ is:

$$w = (36.4164559, -0.0133308857)^T$$

See Figure 3 for the plotted function.

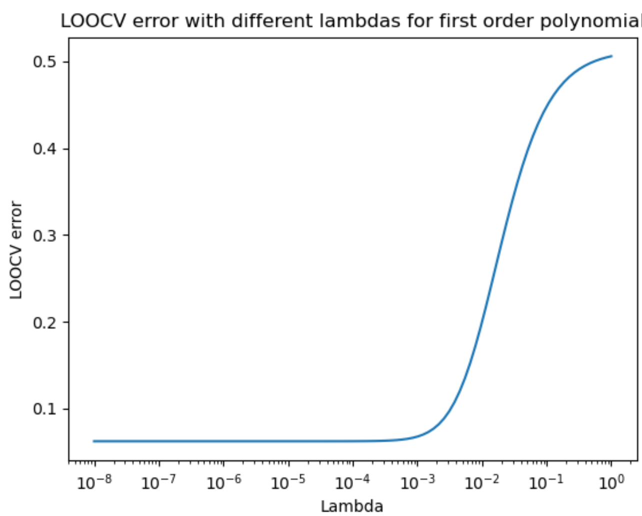

Figure 2: Function plotting the LOOCV error against λ

b)

The best calculated lambda value is $\lambda = 0.0005857021$ with coefficients:

$w = (0.0185611763, 8.98531621, -0.0136641912, 0.00000693345065, -0.00000000117322729)^T$

The calculated coefficients for $\lambda = 0$ is:

$w = (507369.014, -1027.66582, 0.780506042, -0.000263433309, 0.0000000333384222)^T$

See Figure 3 for the plotted function.

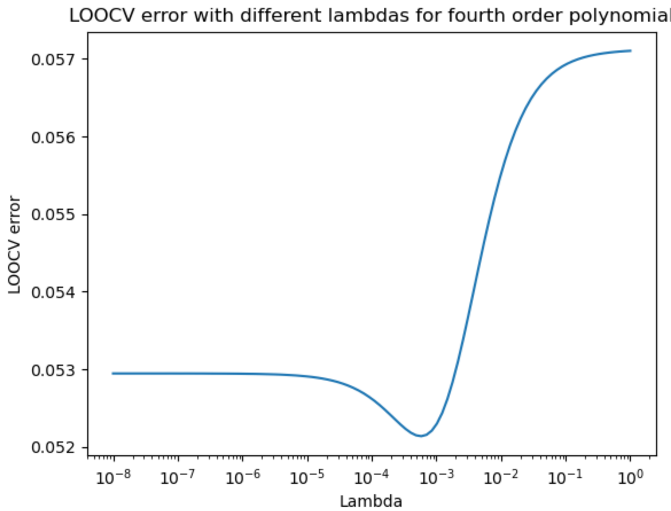

Figure 3: Function plotting the LOOCV error against λ

Exercise 3

a)

To calculate the PDF we need to take the derivative of F(x), we only need to do it for $x > 0$. With $g(x) = exp(-\beta x^\alpha)$ and $F(x) = 1 - g(x)$ using the chain rule

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

$$\begin{aligned}\frac{dF}{dx} &= \frac{d}{dx}(1) - \frac{dg}{dx} \\ &= 0 - \frac{dg}{dx} \\ &= 0 - (-\beta\alpha x^{\alpha-1}) \cdot exp(-\beta x^\alpha) \\ &= \beta\alpha x^{\alpha-1} \cdot exp(-\beta x^\alpha)\end{aligned}$$

For $x \leq 0$ the derivative of 0 is 0.

b)

For this subtask we have $F(x) = 1 - exp(-\frac{1}{4}x^2)$ (The CDF) First we calculate the probability that the chip works longer than 4 years:

$$\begin{aligned}P(X > 4) &= 1 - F(4) \\ &= 1 - (1 - exp(-\frac{1}{4}4^2)) \\ &= 1 - (1 - exp(-4)) \\ &= 0.01831563888\end{aligned}$$

The probability is roughly 1.8%

Now we calculate the probability that it stops working in the interval [5;10]:

$$\begin{aligned}P(5 \leq x \leq 10) &= F(10) - F(5) \\ &= (1 - exp(-\frac{1}{4}10^2)) - (1 - exp(-\frac{1}{4}5^2)) \\ &= (1 - exp(-25)) - (1 - exp(-6.25)) \\ &= 0.00193045412\end{aligned}$$

The probability of this is roughly 0.19%

c)

The median is the point with 50% so for general choices of α and β we have to solve the following: $F(x) = 0.5$

Exercise 4

For this exercise it's assumed that a years is always 365 days.

a)

We start by writing up all the given probabilities for a person (NC):

$$P(court \mid silent) = 0.001$$
$$P(court \mid \neg silent) = 0.0015$$
$$P(\neg convicted \mid \neg silent) = 0.8$$
$$P(\neg convicted \mid silent) = 0.2$$
$$P(convicted \mid silent) = 1 - P(\neg convicted \mid silent) = 0.8$$
$$P(convicted \mid \neg silent) = 1 - P(\neg convicted \mid \neg silent) = 0.2$$

Now we need to calculate the two possible scenarios out:
For remaining silent the expected prison time can be calculated as the probability of going to court when remaining silent, times the probability of being convicted when remaining silent times the 5 years (in days).

$$\begin{aligned}&P(court \mid silent) \cdot P(convicted \mid silent) \cdot (5 \cdot 365)\\&= 0.001 \cdot 0.8 \cdot 1825\\&= 1.46\end{aligned}$$

For talking the expected prison time can be calculated as the probability of going to court when having talked, times the probability of being convicted when having talked, multiplied by the 5 years (in days) reduced by 50%.

$$\begin{aligned}&P(court \mid \neg silent) \cdot P(convicted \mid \neg silent) \cdot (5 \cdot 365 \cdot 0.5)\\&= 0.0015 \cdot 0.2 \cdot 912.5\\&= 0.27375\end{aligned}$$

b)

We start by writing up all the given probabilities for a person (C):

$$P(court \mid silent) = 0.001$$
$$P(court \mid \neg silent) = 0.005$$
$$P(\neg convicted \mid \neg silent) = 0.2$$
$$P(\neg convicted \mid silent) = 0.05$$
$$P(convicted \mid silent) = 1 - P(\neg convicted \mid silent) = 0.95$$
$$P(convicted \mid \neg silent) = 1 - P(\neg convicted \mid \neg silent) = 0.8$$

Now we need to calculate the two possible scenarios out:
For remaining silent the expected prison time can be calculated as the probability of going to court when remaining silent, times the probability of being convicted when remaining silent times the 5 years (in days).

$$
\begin{aligned}
&P(court \mid silent) \cdot P(convicted \mid silent) \cdot (5 \cdot 365) \\
&= 0.001 \cdot 0.95 \cdot 1825 \\
&= 1.73375
\end{aligned}
$$

For talking the expected prison time can be calculated as the probability of going to court when having talked, times the probability of being convicted when having talked, multiplied by the 5 years (in days) reduced by 50%.

$$
\begin{aligned}
&P(court \mid \neg silent) \cdot P(convicted \mid \neg silent) \cdot (5 \cdot 365 \cdot 0.5) \\
&= 0.005 \cdot 0.8 \cdot 912.5 \\
&= 3.65
\end{aligned}
$$

Appendix

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, alpha=None, lambdaVal=0):
        self.alpha = alpha
        self.lambdaVal = lambdaVal
        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]
        alpha : Array of shape [n_samples, 1], optional
        """

        # Ensure the arrays are N-dimensional numpy arrays
        X = numpy.array(X).reshape((len(X), -1))
        t = numpy.array(t).reshape((len(t), 1))

        # Add a column at the beginning of the feature matrix
        oneCol = numpy.ones((X.shape[0], 1))
        X = numpy.concatenate((oneCol, X), axis=1)

        # If no alpha is provided use the identity matrix to not affect the result
        if self.alpha is None:
            A = numpy.identity(X.shape[0])
        # Otherwise use it
        else:
            self.alpha = numpy.array(self.alpha).reshape((len(self.alpha), 1))
            A = numpy.diag(self.alpha.flatten())

        diagLambda = self.lambdaVal * numpy.identity(X.shape[1])

        # calculate the weights using the formula from the lecture
        firstPart = ((X.T @ A) @ X) + diagLambda
        secondPart = (X.T @ A) @ t
        #self.w = numpy.linalg.inv(firstPart) @ secondPart
```

```
        self.w = numpy.linalg.solve(firstPart, secondPart)

    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]
        """

        # Ensure the array is a N-dimensional numpy array
        X = numpy.array(X).reshape((len(X), -1))

        # Add a column at the beginning of the feature matrix
        oneCol = numpy.ones((X.shape[0], 1))
        X = numpy.concatenate((oneCol, X), axis=1)

        # calculate the predictions
        predictions = X @ self.w

        return predictions
```

exercis_1.py

```
import numpy
import pandas
import linweighreg as linreg
import matplotlib.pyplot as plt

# load data
train_data = numpy.loadtxt("boston_train.csv", delimiter=",")
test_data = numpy.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 model using all features
model_all = linreg.LinearRegression((t_train ** 2))
model_all.fit(X_train, t_train)
print("Weights: %s" % model_all.w)

def rmse(t, tp):
    t = t.reshape((len(t), 1))
    tp = tp.reshape((len(tp), 1))
    return numpy.sqrt(numpy.mean((t - tp) ** 2))

```

```python
# evaluate on test data
# all features
pred_all = model_all.predict(X_test)
print("RMSE using the model (all features): %f" % rmse(pred_all, t_test))
rmse_all = rmse(pred_all, t_test)
plt.figure()
plt.scatter(t_test, pred_all)
plt.ylim(0, 60)
plt.xlim(0, 60)
plt.xlabel("Actual price")
plt.ylabel("Predicted price")
plt.title("All features - RMSE: %f" % rmse_all)

plt.savefig("1b.png")
plt.show()
```

exercise_2.py

```python
import numpy
import linweighreg as linreg
import matplotlib.pyplot as plt

# Load the data
raw = numpy.genfromtxt('men-olympics-100.txt', delimiter=' ')

t = raw[:, 1]
t = t.reshape((len(raw), 1))

lambdaVals = numpy.logspace(-8, 0, 100, base=10)

def loocv(X, t, lambdaVal):
    loss = 0

    for i in range(len(X)):
        # Remove the i-th element from the data
        X_train = numpy.delete(X, i, axis=0)
        t_train = numpy.delete(t, i, axis=0)

        # Use the i-th element as test (leave one out cross validation)
        X_test = X[i].reshape(1, -1)
        t_test = t[i]

        # Fit model, predict and add the loss
        model = linreg.LinearRegression(lambdaVal=lambdaVal)
        model.fit(X_train, t_train)
        predictions = model.predict(X_test)
        loss += (predictions - t_test) ** 2

    # Return the average loss for the given lambda
    return (loss / len(X)).flatten()
```

```
print("=== First order polynomial ===")
X = raw[:, 0]
X = X.reshape((len(raw), 1))

# Calculate the loss for each lambda
results = numpy.array([loocv(X, t, lambdaVal) for lambdaVal in
    lambdaVals])

model_zero = linreg.LinearRegression(lambdaVal = 0)
model_zero.fit(X, t)
print("Calculated weights for lambda=0: %s" % model_zero.w)

bestLambda = lambdaVals[numpy.argmin(results)]
model_bestlambda = linreg.LinearRegression(lambdaVal = bestLambda)
model_bestlambda.fit(X, t)
print("Best lambda: %.10f with loss: %.10f" % (bestLambda, numpy.
    min(results)))
print("Calculated weights for lambda=%.10f: %s" % (bestLambda,
    model_bestlambda.w))

plt.plot(lambdaVals, results)
plt.xlabel("Lambda")
plt.ylabel("LOOCV error")
plt.xscale("log")
plt.title("LOOCV error with different lambdas for first order
    polynomial")
plt.savefig("LOOCV_error_firstorder.png")

plt.show()

print("=== Fourth order polynomial ===")
X = raw[:, 0]
X = X.reshape((len(raw), 1))
X = numpy.concatenate((X, X ** 2, X ** 3, X ** 4), axis=1)

# Calculate the loss for each lambda
results = numpy.array([loocv(X, t, lambdaVal) for lambdaVal in
    lambdaVals])

model_zero = linreg.LinearRegression(lambdaVal = 0)
model_zero.fit(X, t)
print("Calculated weights for lambda=0: %s" % model_zero.w)

bestLambda = lambdaVals[numpy.argmin(results)]
model_bestlambda = linreg.LinearRegression(lambdaVal = bestLambda)
model_bestlambda.fit(X, t)
print("Best lambda: %.10f with loss: %.10f" % (bestLambda, numpy.
    min(results)))
print("Calculated weights for lambda=%.10f: %s" % (bestLambda,
    model_bestlambda.w))

plt.plot(lambdaVals, results)
plt.xlabel("Lambda")
plt.ylabel("LOOCV error")
plt.xscale("log")
plt.title("LOOCV error with different lambdas for fourth order
    polynomial")
```

```
plt.savefig("LOOCV_error_fourthorder.png")

plt.show()
```