MAD Assignment 1

Andreas V. W. Zacchi (nzl169)

November 24, 2024

Exercise 1

a)

$$f(x,y) = x^4y^3 + 7x^5 - e^{xy}$$

First we calculate $\frac{\partial f}{\partial x}$ where y is treated as a constant:
For $-e^{xy}$ we use the chain rule to get $-ye^{xy}$

$$\begin{aligned}
&\frac{\partial f}{\partial x} x^4y^3 + 7x^5 - e^{xy} \\
&= 4y^3x^3 + (7 \cdot 5)x^4 - ye^{xy} \\
&= 4y^3x^3 + 35x^4 - ye^{xy}
\end{aligned}$$

Meaning the partial derivative with respect to x is: $4y^3x^3 + 35x^4 - ye^{xy}$

Then we calculate $\frac{\partial f}{\partial y}$ where x is treated as a constant:
For $-e^{xy}$ we use the chain rule to get $-xe^{xy}$

$$\frac{\partial f}{\partial y} x^4y^3 + 7x^5 - e^{xy} = 3x^4y^2 - xe^{xy}$$

Meaning the partial derivative with respect to y is: $3x^4y^2 - xe^{xy}$

b)

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

We first rewrite the function: Using that $\frac{1}{x} = x^{-1}$ and $\sqrt{x} = x^{\frac{1}{2}}$ we can rewrite the function
$f(x,y) = (x^3 + xy + y^2)^{-\frac{1}{2}}$

Now we calculate $\frac{\partial f}{\partial x}$ where y is treated as a constant:
We can use the chain rule with

$$\begin{aligned}
g &= u^{-\frac{1}{2}} \\
h &= x^3 + xy + y^2 \\
\frac{\partial g}{\partial u} &= -\frac{1}{2}u^{-\frac{3}{2}} \\
\frac{\partial h}{\partial x} &= 3x^2 + y
\end{aligned}$$

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

Meaning the partial derivative with respect to x is: $-\frac{3x^2+y}{2(x^3+xy+y^2)^{\frac{3}{2}}}$

Then we calculate $\frac{\partial f}{\partial y}$ where x is treated as a constant:
We can use the chain rule with

$$
\begin{aligned}
g &= u^{-\frac{1}{2}} \\
h &= x^3+xy+y^2 \\
\frac{\partial g}{\partial u} &= -\frac{1}{2}u^{-\frac{3}{2}} \\
\frac{\partial h}{\partial y} &= x+2y
\end{aligned}
$$

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

Meaning the partial derivative with respect to y is: $-\frac{x+2y}{2(x^3+xy+y^2)^{\frac{3}{2}}}$

c)

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

First we calculate $\frac{\partial f}{\partial x}$ where y is treated as a constant:
We use the quotient rule where

$$\begin{aligned} g(x,y) &= x^3 + y^2 \\ h(x,y) &= x + y \\ \frac{\partial g}{\partial x} &= 3x^2 \\ \frac{\partial h}{\partial x} &= 1 \end{aligned}$$

$$\begin{aligned} \frac{\partial f}{\partial x} &= \frac{\frac{\partial g}{\partial x} h - g \frac{\partial h}{\partial x}}{h^2} \\ &= \frac{3x^2 \cdot (x+y) - (x^3 + y^2) \cdot 1}{(x+y)^2} \\ &= \frac{3x^2 \cdot (x+y) - x^3 - y^2}{(x+y)^2} \\ &= \frac{3x^3 + 3x^2 y - x^3 - y^2}{(x+y)^2} \\ &= \frac{2x^3 + 3x^2 y - y^2}{(x+y)^2} \end{aligned}$$

Meaning the partial derivative with respect to x is: $\frac{2x^3+3x^2y-y^2}{(x+y)^2}$

Then we calculate $\frac{\partial f}{\partial y}$ where x is treated as a constant:
We use the quotient rule where

$$\begin{aligned} g(x,y) &= x^3 + y^2 \\ h(x,y) &= x + y \\ \frac{\partial g}{\partial y} &= 2y \\ \frac{\partial h}{\partial y} &= 1 \end{aligned}$$

$$
\begin{aligned}
\frac{\partial f}{\partial y} &= \frac{\frac{\partial g}{\partial y} h - g \frac{\partial h}{\partial y}}{h^2} \\
&= \frac{2y \cdot (x+y) - (x^3 + y^2) \cdot 1}{(x+y)^2} \\
&= \frac{2y \cdot (x+y) - x^3 - y^2}{(x+y)^2} \\
&= \frac{2yx + 2y^2 - x^3 - y^2}{(x+y)^2} \\
&= \frac{2yx + y^2 - x^3}{(x+y)^2}
\end{aligned}
$$

Meaning the partial derivative with respect to y is: $\frac{2yx+y^2-x^3}{(x+y)^2}$

Exercise 2

a)

Using (3) from Table 1.4 in Rogers and Girolami ($\bar{x}^T\bar{x} = 2\bar{x}$) we get:

$$\nabla f(\bar{x}) = 2\bar{x}$$

As the constant scalar is removed when calculating the gradient.

b)

Using (2) from Table 1.4 in Rogers and Girolami ($\bar{x}^T\bar{b} = \bar{b}$) we get:

$$\nabla f(\bar{x}) = \bar{b}$$

c)

Assuming the matrix A is symmetric we can use (4) from Table 1.4 in Rogers and Girolami ($\bar{x}^T A\bar{x} = 2A\bar{x}$). We start by splitting it up so we get:

$$\begin{aligned}&\nabla(\bar{x}^T A\bar{x} + \bar{b}^T\bar{x} + c)\\ &= \nabla(\bar{x}^T A\bar{x}) + \nabla(\bar{b}^T\bar{x}) + \nabla c\end{aligned}$$

The first part is equal to the expression from the book, the second is equal to sub task b) as $\bar{b}^T\bar{x} = \bar{x}^T\bar{b}$ and the third is removed as it's constant. This means the gradient, assuming A is symmetric is:

$$\nabla f = 2A\bar{x} + \bar{b}$$

Exercise 3

a)

The mean price of the houses is calculated as 22.016601.

b)

Using the implemented RMSE function we find that the RMSE is 9.672478.

c)

To see the predicted results plotted against the actual prices see Figure 1.

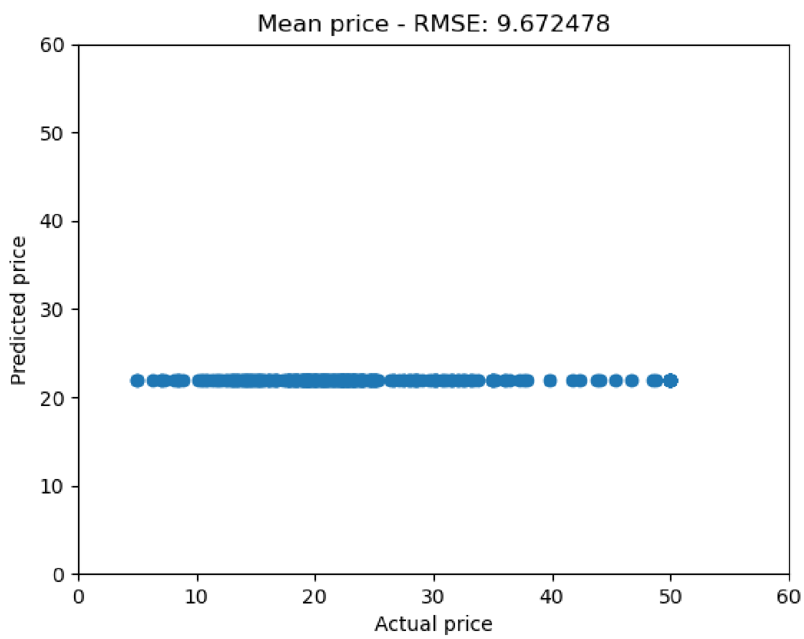

Figure 1: Scatter plot plotting the actual prices against the predicted

Exercise 4

a)

See the code

b)

The optimal weights only considering the first feature and rounded to 3 deciamls is $\hat{w} = (23.635, -0.433)^T$. So we have $\hat{w}_0 = 23.635$ $\hat{w}_1 = -0.433$. This means that the house price is negatively affected by a higher crime rate.

c)

The optimal weights considering all features and rounded to 3 decimals is:

$$\hat{w} = (31.389, -0.060, 0.029, -0.029, 2.293, -17.326, 3.994, 0.003,$$
$$- 1.287, 0.355, -0.016, -0.815, 0.012, -0.465)^T$$

d)

The RMSE for the single feature is rounded to 3 deciamls 8.955, and for the all features it is 4.688. See Figure 2 for single feature and Figure 3 for the one with all features.

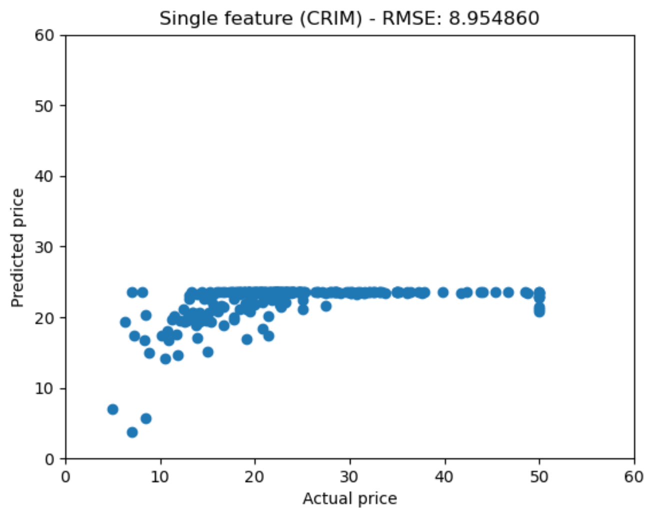

Figure 2: Scatter plot plotting the actual prices against the predicted

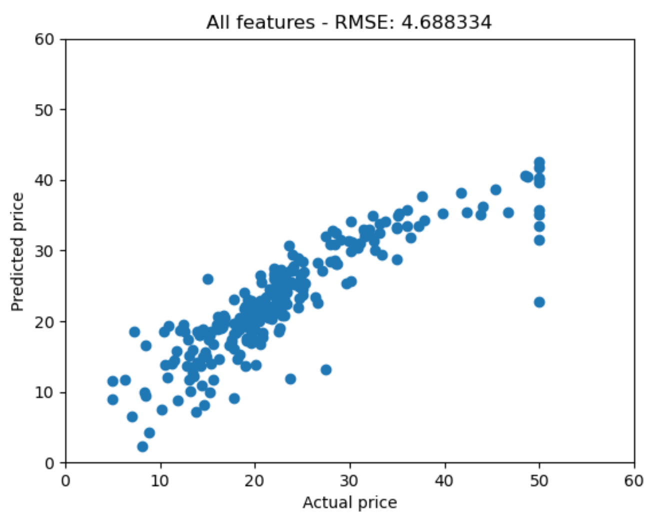

Figure 3: Scatter plot plotting the actual prices against the predicted

Exercise 5

The total loss is given by

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

We need to compute the derivative with respect to w for the expression inside the sum as the gradient of the sum is the sum of the gradients: We use the chain rule with $g = u^2$ and $h = (w^T x_n - t_n)$ so $\frac{\partial h}{\partial w} = x_n$ and $\frac{\partial g}{\partial u} = 2u$

$$\frac{\partial}{\partial w}(w^T x_n - t_n) = 2(w^T x_n - t_n) \cdot x_n$$

Now we add the sum again to get:

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

We factor out the constant:

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

Now to find the critical points we set the found gradient equal to zero, the 2 disappears as it only affects the magnitude of the solution:

$$\sum_{n=1}^{N} (w^T x_n - t_n) \cdot x_n = 0$$
$$\sum_{n=1}^{N} w^T x_n x_n - t_n x_n = 0$$
$$\sum_{n=1}^{N} w^T x_n x_n = \sum_{n=1}^{N} t_n x_n$$

This can be written in matrix form as

$$w^T X^T X = t^T X$$

We transpose both sides (so w^T becomes w)

$$X^T X w = X^T t$$

We take the inverse of $X^T X$ on both sides to get:

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

The average loss is given by

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

The only difference is $\frac{1}{N}$, which is a constant in regards to the gradient (it only changes the magnitude). This means that the derivation is the same and the solution therefore remains the same.

Appendix

Excerise 3

housing_1.py

```
import numpy
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])

# (a) compute mean of prices on training set
mean_price = numpy.mean(t_train)
print("The mean price of the houses is: %f" % mean_price)

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

# Create a list of the predicted prices
prediction = mean_price * numpy.ones(len(t_test))
print("RMSE using the model (mean): %f" % rmse(prediction, t_test))

# (c) visualization of results
plt.figure()
plt.scatter(t_test, prediction)
plt.ylim(0, 60)
plt.xlim(0, 60)
plt.xlabel("Actual price")
plt.ylabel("Predicted price")
plt.title("Mean price - RMSE: %f" % rmse(prediction, t_test))

plt.savefig("housing_1.png")
plt.show()
```

Excerise 4

housing_2.py

```python
import numpy
import pandas
import 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 using only the first feature
model_single = linreg.LinearRegression()
model_single.fit(X_train[:,0], t_train)
print("Weights: %s" % model_single.w)

# (c) fit linear regression model using all features
model_all = linreg.LinearRegression()
model_all.fit(X_train, t_train)
print("Weights: %s" % model_all.w)

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

# evaluate on test data
# single feature
pred_single = model_single.predict(X_test[:,0])
rmse_single = rmse(pred_single, t_test)
print("RMSE using the model (single feature (CRIM)): %f" % rmse_single)
plt.figure()
plt.scatter(t_test, pred_single)
plt.ylim(0, 60)
plt.xlim(0, 60)
plt.xlabel("Actual price")
plt.ylabel("Predicted price")
plt.title("Single feature (CRIM) - RMSE: %f" % rmse_single)
plt.savefig("housing_2_single.png")

# 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("housing_2_all.png")
plt.show()
```

linreg.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]
        """

        # 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)

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

```

```python
        # 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
```