Assignment 1

Marion Rosec

November 2023

Exercise 1

a)

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

b)

We can calculate the partial derivatives using the derivation formulae for the inverse function and the square root function :

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

c)

Here, we can use the quotient rule :

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

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

Exercise 2

a)

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

$$\nabla f(\bar{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \dots \\ \frac{\partial f}{\partial x_i} \\ \dots \\ \frac{\partial f}{\partial x_N} \end{bmatrix}$$

With $\bar{x}^T\bar{x} = \sum_{i=1}^{N} x_i^2$ So

$$\frac{\partial f}{\partial x_i} = 2x_i$$

Then we have :

$$\nabla f(\bar{x}) = \begin{bmatrix} 2x_1 \\ \dots \\ 2x_N \end{bmatrix} = 2\bar{x}$$

b)

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

With $\bar{x}^T\bar{b} = \sum_{i=1}^{N} x_i b_i$

$$\frac{\partial f}{\partial x_i} = b_i$$

Finally, we have :

$$\nabla f(\bar{x}) = \begin{bmatrix} b_1 \\ \dots \\ b_N \end{bmatrix} = \bar{b}$$

c)

With the previous question, we have

$$\nabla(\bar{b}^T\bar{x}) = \bar{b}$$

Next we can compute $\nabla(\bar{x}^T A\bar{x}))$

$$\begin{aligned} \frac{\partial \bar{x}^T A\bar{x})}{\partial \bar{X}} &= \frac{\partial \bar{x^T}}{\bar{x}} A\bar{x} + \bar{x}^T \frac{\partial A\bar{x}}{\partial \bar{x}} \text{ using the product rule} \\ &= A\bar{x} + \bar{x}^T A \\ &= A\bar{x} + A^T\bar{x} \\ &= (A + A^T)\bar{x} \end{aligned}$$

Finally, we have :

$$\begin{aligned}\nabla f &= \nabla(\bar{b}^T\bar{x}) + \nabla(\bar{x}^T A\bar{x}) + 0 \\ &= \bar{b} + (A + A^T)\bar{x}\end{aligned}$$

Exercise 3

In this exercise, we found for the training dataset a mean of the house prices equal to 22.02 (a). In order to compute the RMSE loss, we needed to convert this scalar into an N-Dimensional vector. Lastly, after computing a RMSE of 9.67 (b), we visualized the results (figure 1). The RMSE value means that there is almost 10,000$ between the true prices and the computed average of our training set.

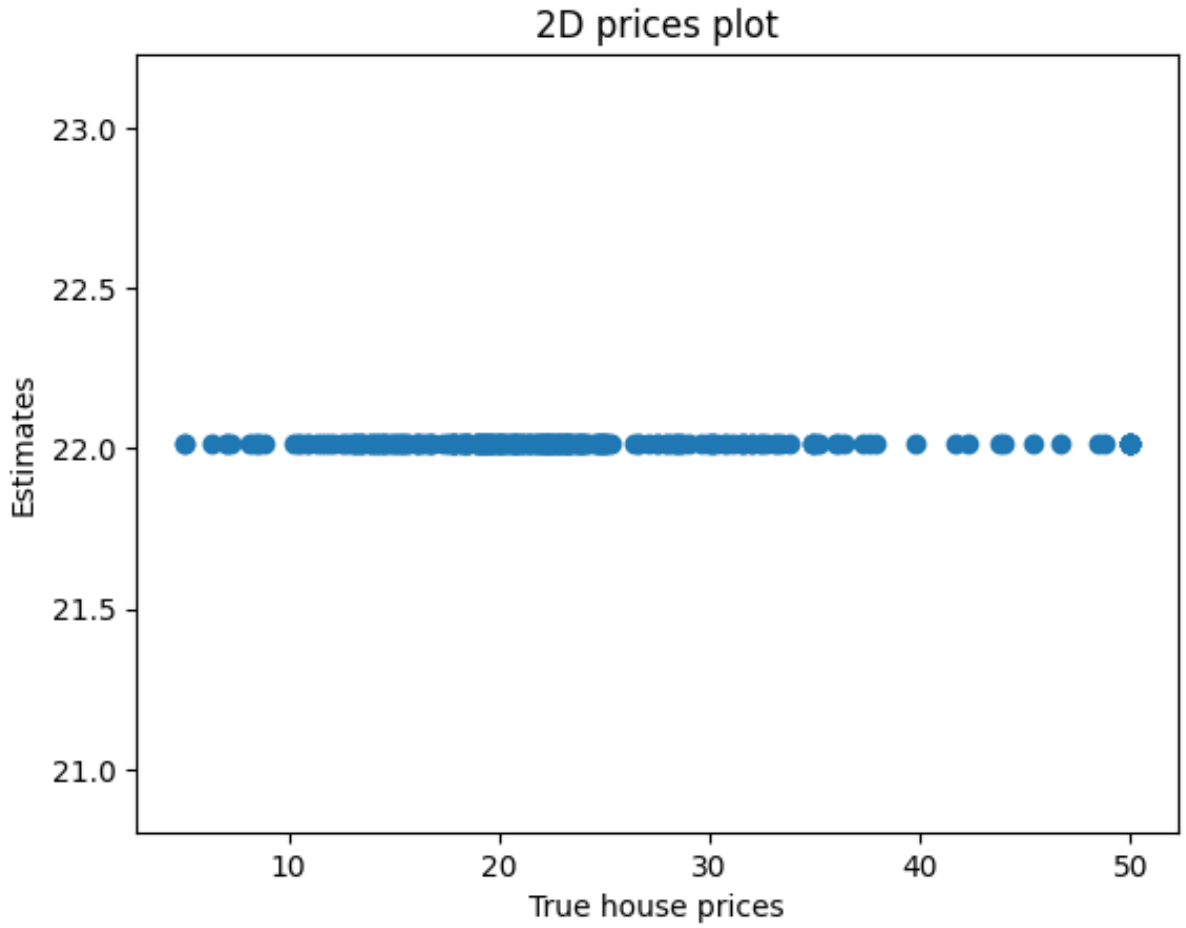

Figure 1 – Estimated prices average on true house prices

Python

```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]
```

```python
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
print(np.mean(t_train))

# (b) RMSE function
v_t_train = np.full(t_train.size, np.mean(t_train))

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

print(rmse(t_test, v_t_train))

# (c) visualization of results
plt.scatter(t_test, v_t_train)
plt.xlabel("True house prices")
plt.ylabel("Estimates")
plt.title("2D prices plot")
plt.show()
```

Exercise 4

For this exercise, we firstly have to complete the function fit from the LinearRegression class in order to compute the optimal coefficient $\hat{w}$ (a). Using that function, we find the optimal coefficient value for our training set linear regression model with only the first feature CRIM (b) :

Python

```
>>> [[23.63506195]
     [-0.43279318]]
```

w_1 being negative, we can say that the house prices decrease when the crime rate is higher. The value of w_0, called bias, is the estimated house price for a crime rate of 0.

We can also compute the coefficient considering all the features of our training set (c) :

Python

```
>>> [[ 3.13886978e+01] [-5.96169127e-02] [ 2.93672792e-02] [-2.90605834e-02]
     [ 2.29256181e+00] [-1.73263655e+01] [ 3.99375996e+00] [ 3.23077761e-03]
     [-1.28724508e+00] [ 3.54780191e-01] [-1.55819191e-02] [-8.14647713e-01]
     [ 1.17820208e-02] [-4.64869014e-01]]
```

We can see that some of the features have a higher weight than others, and some of the features may increase or decrease the prices of housing. For instance, w_5 being negative, the expected house prices decrease with the RM (average number of rooms in the dwelling). In the same way, the estimated prices increase with the nitric oxide concentration (because w_4 is positive).

With the computed optimal coefficient, we can then create the function predict which predict the estimated house prices from either the CRIM feature alone (figure 2) or from all the different features (figure 3) of our testing dataset. We can then compare the true prices of our testing set with the predicted ones by plotting them (d).

Figure 2 – Prices estimated by CRIM feature on true house prices

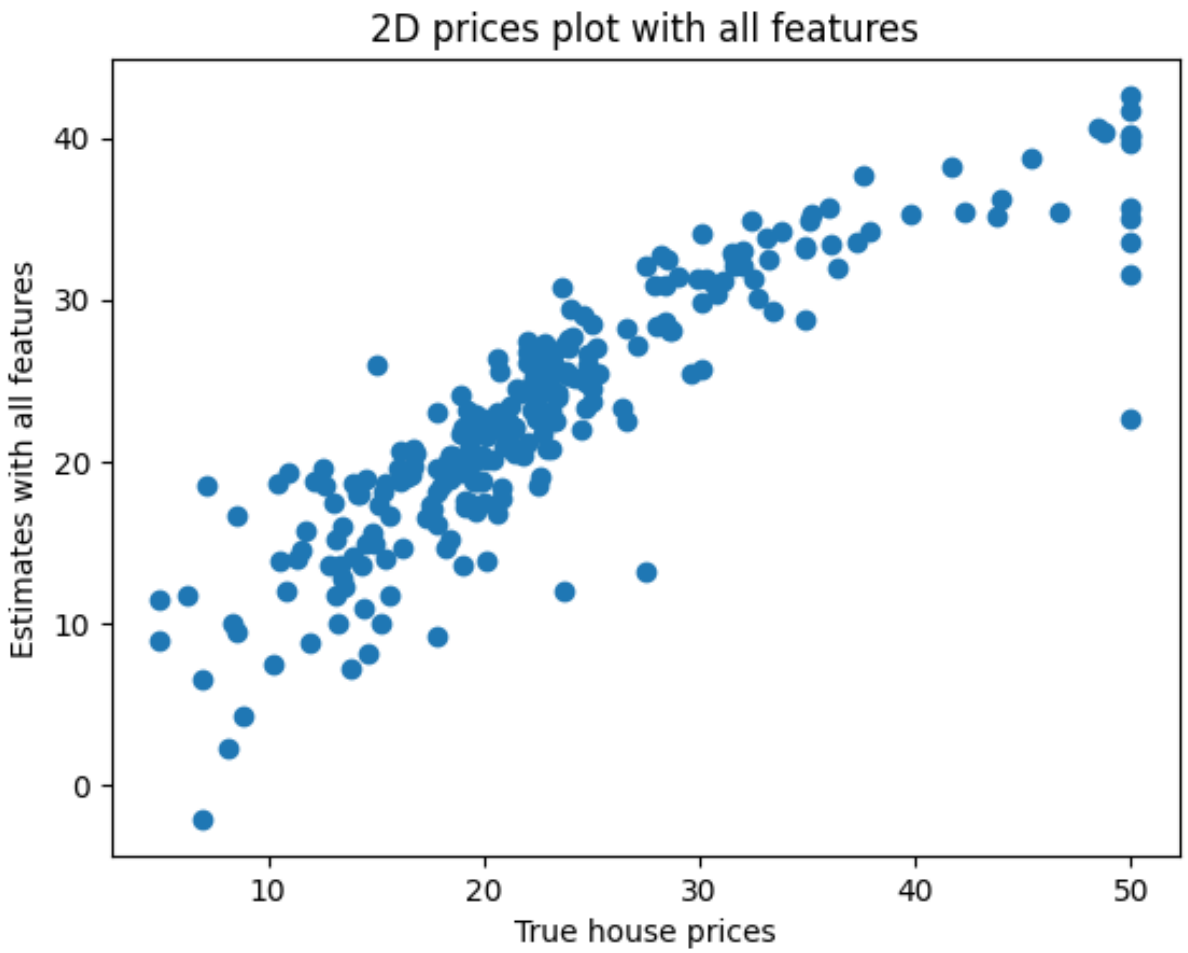

Figure 3 – Prices estimated by all features on true house prices

We can also compute the RMSE for both of the models (with one, or all features) : Single feature RMSE = 8.95, All features RMSE = 4.69. We can see that the difference between the estimated prices and the true prices is smaller when all the features are used in the process, which could mean that, in this case, the model is more accurate with more parameters.

python

```python
import numpy as np

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]
```

```
        """

        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)

        # self.w = np.linalg.pinv((np.dot(X.T, X)))
        # self.w = np.dot(self.w, X.T)
        # self.w = np.dot(self.w, t)
        self.w = np.linalg.solve(X.T @ X, X.T @ t)

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

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

        t_new = np.dot(X, self.w)
        return t_new
```

Python

```
import numpy as np
import pandas
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(model_single.w)

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


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

print(model_single.predict)
t_predict = model_single.predict(X_test[:, 0])
```

```
print(rmse(t_test, t_predict))


plt.scatter(t_test, t_predict)
plt.xlabel("True house prices")
plt.ylabel("Estimates with first feature")
plt.title("2D prices plot with first feature")
plt.savefig('2_feature.png', bbox_inches='tight')
plt.show()

t_predict = model_all.predict(X_test)
print(rmse(t_test, t_predict))

plt.scatter(t_test, t_predict)
plt.xlabel("True house prices")
plt.ylabel("Estimates with all features")
plt.title("2D prices plot with all features")
plt.savefig('all_feature.png', bbox_inches='tight')
plt.show()
```

Exercise 5

Let $t = \begin{bmatrix} t_1 & \cdots & t_N \end{bmatrix}$ Let $X = \begin{bmatrix} x_1 \\ \cdots \\ x_N \end{bmatrix}$ with $x_N = \begin{bmatrix} x_{1,N} & \cdots & x_{M,N} \end{bmatrix}$ Let $W = \begin{bmatrix} w_0 & \cdots & w_N \end{bmatrix}$

The sum is then :

$$L = (WX - t)^2 = (WX - t)^T (WX - t)$$

$$L = W^T X^T WX - W^T X^T t - t^T WX + t^T t = X^T XW^2 - 2W^T X^T t + t^T t$$

In order to compute the optimal parameters, we then need to solve this equation :

$$\frac{\partial L}{\partial W} = 2X^T XW - 2X^T t + 0 = 0$$

$$X^T XW = X^T t \Leftrightarrow IW = (X^T X)^{-1} X^T t$$

The optimal least square values are given by : $\bar{w} = (X^T X)^{-1} X^T t$