Assignment 1

MAD

Modelling and Analysis of Data

NDAB16012U

Department of Computer Science

University of Copenhagen

Jack Bjerregaard - `ldh587`

23/11/2025

1 Partial Derivatives

a) To find the partial derivate of $f(x, y) = x^4y^3 + 7x^5 - e^{xy}$ we begin by finding the partial derivative with respect to x, where we treat y as a constant, and then do the same with respect to y:

$$f(x, y) = x^4y^3 + 7x^5 - e^{xy} \tag{1}$$

$$\frac{\partial f}{\partial x} = \frac{\partial}{\partial x}\left(x^4y^3 + 7x^5 - e^{xy}\right) \tag{2}$$

$$= 4x^3y^3 + 35x^4 - ye^{xy} \tag{3}$$

$$\frac{\partial f}{\partial y} = \frac{\partial}{\partial y}\left(x^4y^3 + 7x^5 - e^{xy}\right) \tag{4}$$

$$= 3x^4y^2 - xe^{xy} \tag{5}$$

b) Similar to how we solved the previous partial derivatives, we again attempt to find the partial derivatives of each variable, treating the other as a constant. However, to simplify it, we begin by rewriting the function with exponent $-\frac{1}{2}$ and then using the chain rule:

$$f(x, y) = \frac{1}{\sqrt{x^3 + xy + y^2}} = \left(x^3 + xy + y^2\right)^{-1/2} \tag{6}$$

$$\frac{\partial f}{\partial x} = \frac{\partial}{\partial x}\left(x^3 + xy + y^2\right)^{-1/2} \tag{7}$$

$$= -\frac{1}{2}\left(x^3 + xy + y^2\right)^{-3/2} \cdot \frac{\partial}{\partial x}(x^3 + xy + y^2) \tag{8}$$

$$= -\frac{1}{2}\left(x^3 + xy + y^2\right)^{-3/2}(3x^2 + y) \tag{9}$$

$$= -\frac{3x^2 + y}{2\sqrt{\left(x^3 + xy + y^2\right)^3}} \tag{10}$$

$$\frac{\partial f}{\partial y} = \frac{\partial}{\partial y}\left(x^3 + xy + y^2\right)^{-1/2} \tag{11}$$

$$= -\frac{1}{2}\left(x^3 + xy + y^2\right)^{-3/2} \cdot \frac{\partial}{\partial y}(x^3 + xy + y^2) \tag{12}$$

$$= -\frac{1}{2}\left(x^3 + xy + y^2\right)^{-3/2}(x + 2y) \tag{13}$$

$$= -\frac{x + 2y}{2\sqrt{\left(x^3 + xy + y^2\right)^3}} \tag{14}$$

c) To solve the last partial derivative problem, we do the same as in b), however instead of the chain rule we use the quotient rule:

$$f(x,y) = \frac{x^3+y^2}{x+y} \tag{15}$$

$$\frac{\partial f}{\partial x} = \frac{\partial}{\partial x}\left(\frac{x^3+y^2}{x+y}\right) \tag{16}$$

$$= \frac{(3x^2)(x+y)-(x^3+y^2)(1)}{(x+y)^2} \tag{17}$$

$$= \frac{3x^3+3x^2y-x^3-y^2}{(x+y)^2} \tag{18}$$

$$= \frac{2x^3+3x^2y-y^2}{(x+y)^2} \tag{19}$$

$$\frac{\partial f}{\partial y} = \frac{\partial}{\partial y}\left(\frac{x^3+y^2}{x+y}\right) \tag{20}$$

$$= \frac{(2y)(x+y)-(x^3+y^2)(1)}{(x+y)^2} \tag{21}$$

$$= \frac{2xy+2y^2-x^3-y^2}{(x+y)^2} \tag{22}$$

$$= \frac{-x^3+2xy+y^2}{(x+y)^2} \tag{23}$$

2 Gradients

a) We have a function of a vector, $f(\bar{x}) = \bar{x}^T\bar{x} + c$. To compute the gradient ∇f with respect to $\bar{x}$ we note the definition from `Table 1.4` (Rogers & Girolami, 2012, p.23), where we have that the partial derivative with respect to w of $w^Tw = 2w$. Thus, we have:

$$\frac{\partial}{\partial w}(w^Tw) = 2w.$$

Substituting the definition for our question so $w = \bar{x}$,

$$\nabla_{\bar{x}} f = 2\bar{x}.$$

Since our constant c is independent of $\bar{x}$, its derivative is zero and out final gradient is:

$$\nabla f = 2\bar{x}.$$

b) We now consider the linear function of a vector

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

Where we have that $\bar{b}$ is a constant vector. To compute the gradient ∇f with respect to $\bar{x}$, we use the identity from `table 1.4` (rogers & Girolami, 2012, p.23), which states that

for a given vector w and a constant vector x we have:

$$\frac{\partial}{\partial w}(w^T x) = x.$$

Then we substitute for our question so $w = \bar{x}$ and $x = \bar{b}$;

$$\nabla_{\bar{x}} f = \frac{\partial}{\partial \bar{x}}(\bar{x}^T \bar{b}) = \bar{b}.$$

Thus, the gradient of f with respect to $\bar{x}$ is

$$\nabla f = \bar{b}.$$

c) For our final function, we have a function of a quadratic term, linear term and constant:

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

where A is a matrix, $\bar{b}$ is a constant vector, and c is a constant scalar. We compute the gradient ∇f with respect to $\bar{x}$ by differentiating each term separately.

From `Table 1.4` we use the identity that for a vector w and a matrix C

$$\frac{\partial}{\partial w}(w^T C w) = (C + C^T)w.$$

Taking $w = \bar{x}$ and $C = A$ gives

$$\frac{\partial}{\partial \bar{x}}(\bar{x}^T A \bar{x}) = (A + A^T)\bar{x}.$$

For the linear term, we get the same result as part (b):

$$\frac{\partial}{\partial \bar{x}}(\bar{b}^T \bar{x}) = \bar{b}.$$

The constant c does not depend on $\bar{x}$, so its derivative is zero:

$$\frac{\partial}{\partial \bar{x}} c = 0.$$

thus, we get the full gradient:

$$\nabla_{\bar{x}} f = (A + A^T)\bar{x} + \bar{b}.$$

So the gradient of f with respect to $\bar{x}$ is

$$\nabla f = (A + A^T)\bar{x} + \bar{b}.$$

3 Estimating House Prices I

a) To implement the naive approach, we resorted to using the average price for all the houses in the training set. The mean was found to be $\approx \$22.02k$.

b) After implementing and using the `rmse(t,tp)` function, the computed RMSE was found to be $\approx \$9.67k$

C) Using the values computed, and `matplotlib` a 2D scatter plot was created to visualize this naive approach. This can be found in the following figure:

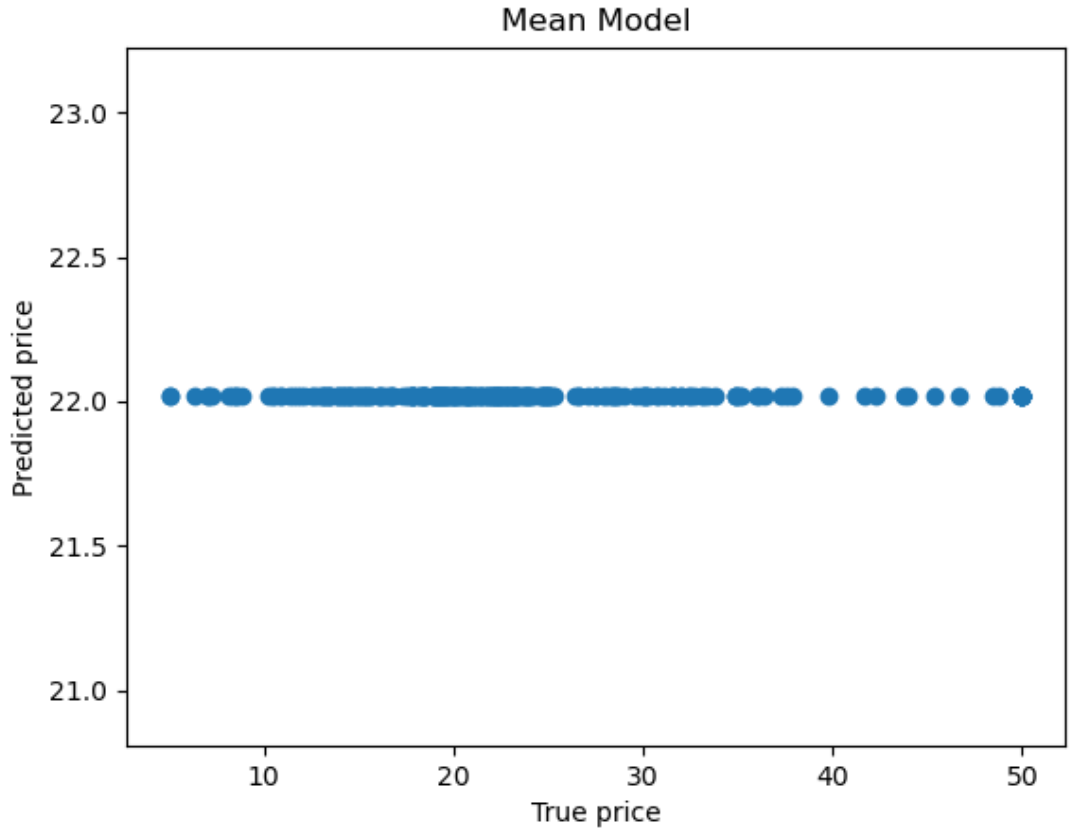

Figure 1: Estimating House Prices using Naive approach

4 Estimating House prices II

a) Our solution to `a)` can be found in the respective `src` files (`housing_2.py` and `linreg.py`. Our `fit` function is as follows:

```
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]
    """
    if X.ndim == 1:
        X = X.reshape(-1, 1)  # converts 1d vector to matrix

    ones = numpy.ones((X.shape[0], 1))
    new_X = numpy.hstack([ones, X])  # make new matrix with 1s

    # bulidng the normal equation - Aw = b, solving for w
    A = new_X.T @ new_X
    b = new_X.T @ t
    self.w = numpy.linalg.solve(A, b)

```

b) Using the code from `a)` a linear regression model was fitted on the training set only containing the first feature. From this we found the 'intercept' to be $\approx \$23.6k$ and the 'CRIM' to be ≈ -0.433. The intercept can be interpreted as being a representation of the predicted house prices (per \$1000) when the crime rate is zero, while the coefficient (CRIM) indicates that for each unit increase in crime rate, the house price decreases by approximately \$433.

c) The computed weights are in the following table:

Index	Weight	Feature
w_0	31.39	Intercept
w_1	-0.0596	CRIM
w_2	0.0294	ZN
w_3	-0.0291	INDUS
w_4	2.293	CHAS
w_5	-17.33	NOX
w_6	3.994	RM
w_7	0.00323	AGE
w_8	-1.287	DIS
w_9	0.3548	RAD
w_{10}	-0.0156	PTRATIO
w_{11}	-0.8146	TAX
w_{12}	0.0118	B
w_{13}	-0.4649	LSTAT

Table 1: Linear regression weights for the full feature model.

d) After implementing the `predict` function, we used it to compute predictions for both models. From this we found that the RMSE for single was $\approx \$8.96k$ and the RMSE for all features was $\approx \$4.69k$. The scatter plots for both features are in the following plots:

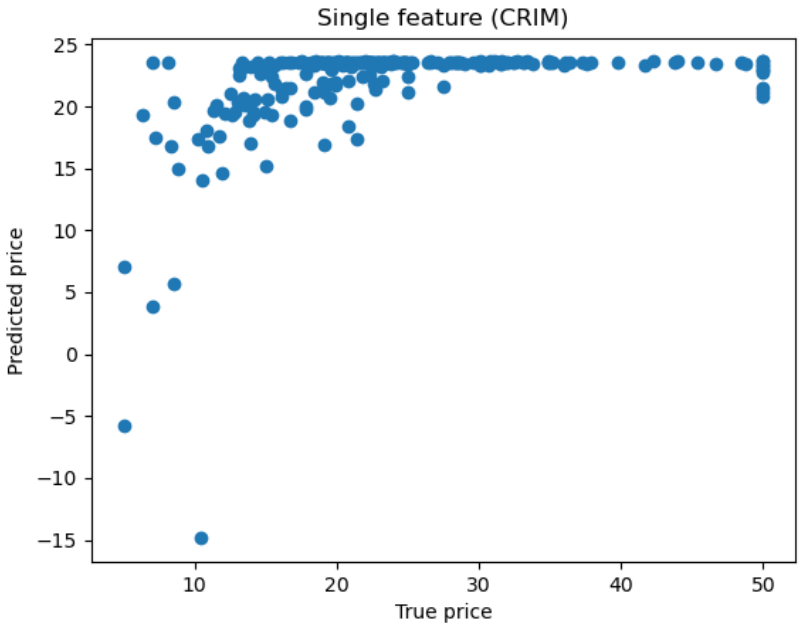

Figure 2: Scatter single

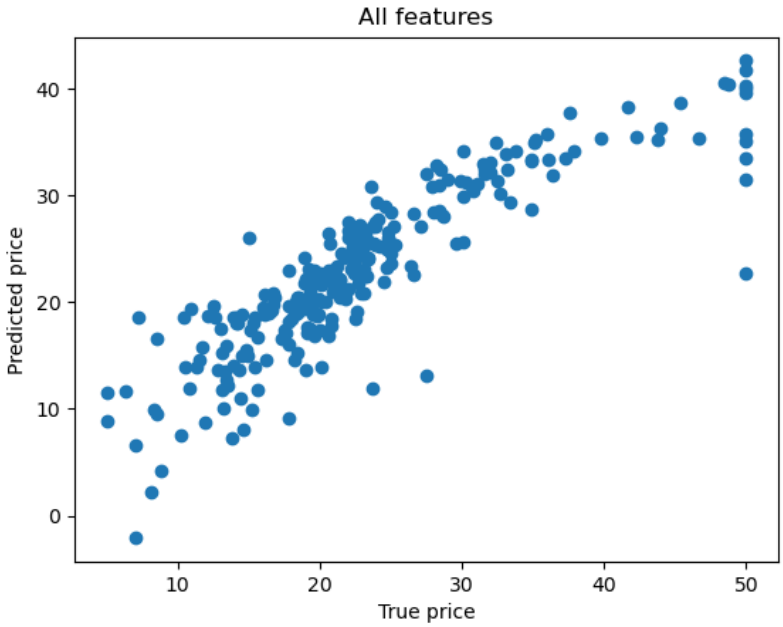

Figure 3: Scatter all

5 Total Training Loss

a) Given the total loss function:

$$\mathcal{L}(\mathbf{w}) = \sum_{n=1}^{N} \left(\mathbf{w}^T \mathbf{x}_n - t_n\right)^2 \tag{24}$$

we begin by rewriting it in matrix for to simplify differentiation and computation. Following section 1.3, (Rogers & Girolami) , we define the data matrix X and the target vector t: We begin by rewriting it in matrix form to simplify differentiation and computation.

$$X = \begin{bmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \vdots \\ \mathbf{x}_N^T \end{bmatrix}, \qquad \mathbf{t} = \begin{bmatrix} t_1 \\ t_2 \\ \vdots \\ t_N \end{bmatrix} \tag{25}$$

Using this notation, the loss can be expressed compactly as (see Equation 1.13)(Rogers & Girolami, p.21):

$$\mathcal{L}(\mathbf{w}) = (X\mathbf{w} - \mathbf{t})^T (X\mathbf{w} - \mathbf{t}) \tag{26}$$

This expression can be expanded using the transpose of a product (Rogers & Girolami, comment 1.8, p.21):

$$\begin{aligned} \mathcal{L}(\mathbf{w}) &= ((X\mathbf{w})^T - \mathbf{t}^T)(X\mathbf{w} - \mathbf{t}) && (27)\\ &= (X\mathbf{w})^T X\mathbf{w} - \mathbf{t}^T X\mathbf{w} - (X\mathbf{w})^T \mathbf{t} + \mathbf{t}^T \mathbf{t} && (28)\\ &= \mathbf{w}^T X^T X\mathbf{w} - 2\,\mathbf{t}^T X\mathbf{w} + \mathbf{t}^T \mathbf{t} && (29) \end{aligned}$$

To optimize we aim to find the minimum, thus we different with respect to w using the identities from Table 1.4 (Rogers & Girolami, p.23:

$$\begin{aligned} \frac{\partial}{\partial \mathbf{w}}\left(\mathbf{w}^T X^T X\mathbf{w}\right) &= 2X^T X\mathbf{w} && (30)\\ \frac{\partial}{\partial \mathbf{w}}\left(-2\,\mathbf{t}^T X\mathbf{w}\right) &= -2X^T \mathbf{t} && (31)\\ \frac{\partial}{\partial \mathbf{w}}\left(\mathbf{t}^T \mathbf{t}\right) &= \mathbf{0} && (32) \end{aligned}$$

Thus, we find the the gradient of the loss to be:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = 2X^T X\mathbf{w} - 2X^T \mathbf{t} \tag{33}$$

Setting the gradient to zero to find the minimum:

$$\begin{aligned} 2X^T X\mathbf{w} - 2X^T \mathbf{t} &= \mathbf{0} && (34)\\ X^T X\mathbf{w} &= X^T \mathbf{t} && (35) \end{aligned}$$

Assuming that X^TX is invertible, we solve for w:

$$(X^TX)^{-1}X^TX\mathbf{w} = (X^TX)^{-1}X^T\mathbf{t} \tag{36}$$

$$I\mathbf{w} = (X^TX)^{-1}X^T\mathbf{t} \tag{37}$$

Sine $Iw = w$ (identity matrix times an vector equals that vector) we have:

$$\hat{\mathbf{w}} = (X^TX)^{-1}X^T\mathbf{t} \tag{38}$$

If we look at the average loss equation from the literature (Rogers & Girolami, equation 1.13, p.21) we see that the equations differ by $\frac{1}{N}$:

$$L = \frac{1}{N}(\mathbf{t} - X\mathbf{w})^T(\mathbf{t} - X\mathbf{w}) \tag{39}$$

Following the same derivation as above we get:

$$\frac{\partial L}{\partial \mathbf{w}} = \frac{2}{N}X^TX\mathbf{w} - \frac{2}{N}X^T\mathbf{t} \tag{40}$$

Setting this to zero and solving:

$$\frac{2}{N}X^TX\mathbf{w} - \frac{2}{N}X^T\mathbf{t} = \mathbf{0} \tag{41}$$

$$X^TX\mathbf{w} = X^T\mathbf{t} \tag{42}$$

This yields the identical solution: $\hat{\mathbf{w}} = (X^TX)^{-1}X^T\mathbf{t}$. This happens as the constant factor $\frac{1}{N}$ in the average loss appears in every term of the gradient. When we set the gradient to zero and solve for w, the factor cancels out.

References

[1] Simon Rogers and Mark Girolami. *A First Course in Machine Learning*. Machine Learning & Pattern Recognition. Chapman and Hall/CRC, Boca Raton, FL, 2 edition, 2016.

A housing_1.py

```python
# 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 = numpy.mean(t_train)
print(f"a) Mean house price on training set: ${mean:.2f}k")

# (b) RMSE function
def rmse(t, tp):
    return numpy.sqrt(numpy.mean((t-tp)**2)) # begin by finding differences, square
    ↪ the errors, and take the mean of the squared errors and finnaly take the squre
    ↪ root

# Create predictions (all equal to mean)
predictions = numpy.full_like(t_test, mean)

# Compute RMSE
test_rmse = rmse(t_test, predictions)
print(f"b) RMSE on test set: ${test_rmse:.2f}k")

# (c) visualization of results
plt.figure()
plt.scatter(t_test, predictions)
plt.xlabel("True price")
plt.ylabel("Predicted price")
plt.title("Mean Model")
plt.savefig("housing_1_scatter.png")
```

B housing_2.py

```python
numpy.set_printoptions(precision=3) #for .3f for arrays

```

```python
# 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) #  builds design matrix we = etc,

print(f"intercept: {model_single.w[0,0]:.3f}")
print(f"CRIM: {model_single.w[1,0]:.3f}")

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

print(f"All weights:\n{model_all.w}")

# (d) evaluation of results

single_prediction = model_single.predict(X_test[:,0])
all_prediction = model_all.predict(X_test)

#RMSE
single_rmse = numpy.sqrt(numpy.mean((single_prediction-t_test)**2))
all_rmse = numpy.sqrt(numpy.mean((all_prediction-t_test)**2))
print(f"RMSE single: {single_rmse:.3f}")
print(f"RMSE all: {all_rmse:.3f}")

plt.figure()
plt.scatter(t_test, single_prediction)
plt.xlabel("True price")
plt.ylabel("Predicted price")
plt.title("Single feature (CRIM)")
plt.savefig("scatter_single.png")

plt.figure()
plt.scatter(t_test, all_prediction)
plt.xlabel("True price")
plt.ylabel("Predicted price")
```

```
plt.title("All features")
plt.savefig("scatter_all.png")

plt.show()

```

C linreg.py

```
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]
    """
    if X.ndim == 1:
        X = X.reshape(-1, 1)  # converts 1d vector to matrix

    ones = numpy.ones((X.shape[0], 1))
    new_X = numpy.hstack([ones, X])  # make new matrix with 1s

    # bulidng the normal equation - Aw = b, solving for w
    A = new_X.T @ new_X
    b = new_X.T @ t
    self.w = numpy.linalg.solve(A, b)

def predict(self, X):
    """
    Computes predictions for a new set of points.

    Parameters
    ----------
    X : Array of shape [n_samples, n_features]

```

```
38      Returns
39      -------
40      predictions : Array of shape [n_samples, 1]
41      """
42      if X.ndim == 1:
43          X = X.reshape(-1,1)
44      ones = numpy.ones((X.shape[0], 1))
45      new_X = numpy.hstack([ones, X])
46
47      return new_X @self.w
```