MAD Assignment 1

dmx289, Jakob Legaard

November 22, 2025

Contents

1 Exercise 1 (Partial Derivatives)

For this exercise, we need to compute and find the partial derivatives $\frac{\partial f}{\partial x}$ and $\frac{\partial f}{\partial y}$. We follow the principle that in $\frac{\partial f}{\partial x}$, y is treated as a constant, and so on.

1.1 a)

We have the following function:

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

we start by finding $\frac{\partial f}{\partial x}$:

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

Now we find $\frac{\partial f}{\partial y}$

$$\frac{\partial f}{\partial y} = 3x^4y^2 - xe^{xy}.$$

1.2 b)

We now take a look at the following function:

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

Given

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

We define g as

$$g(x,y) = x^3 + xy + y^2.$$

Then

$$\frac{\partial f}{\partial x} = -\frac{1}{2}g^{-\frac{3}{2}}\frac{\partial g}{\partial x}, \qquad \frac{\partial f}{\partial y} = -\frac{1}{2}g^{-\frac{3}{2}}\frac{\partial g}{\partial y}.$$

Compute:

$$\frac{\partial g}{\partial x} = 3x^2 + y, \qquad \frac{\partial g}{\partial y} = x + 2y.$$

1.3 c)

We now need to solve the last problem for the following function:

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

we let

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

Using the quotient rule:

$$\frac{\partial f}{\partial x} = \frac{u_x v - u v_x}{v^2}, \qquad \frac{\partial f}{\partial y} = \frac{u_y v - u v_y}{v^2}.$$

Compute:

$$u_x = 3x^2, \quad u_y = 2y, \quad v_x = 1, \quad v_y = 1.$$

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

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

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

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

2 Excercise 2 (Gradients)

we let $\bar{x} \in \mathbb{R}^D$ be a vector, $A \in \mathbb{R}^{D \times D}$ a matrix, $\bar{b} \in \mathbb{R}^D$ a vector, and $c \in \mathbb{R}$ a scalar. We compute the gradient ∇f with respect to $\bar{x}$.

a)

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

We can write

$$\bar{x}^T \bar{x} = \sum_{i=1}^{D} x_i^2.$$

Then the k-th component of the gradient is

$$\frac{\partial f}{\partial x_k} = \frac{\partial}{\partial x_k}\left(\sum_{i=1}^{D} x_i^2\right) = 2x_k.$$

$$\nabla f(\bar{x}) = \begin{pmatrix} 2x_1 \\ \vdots \\ 2x_D \end{pmatrix} = 2\bar{x}.$$

2.1 b)

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

write it as

$$\bar{x}^T \bar{b} = \sum_{i=1}^{D} x_i b_i.$$

The k-th component of the gradient is

$$\frac{\partial f}{\partial x_k} = \frac{\partial}{\partial x_k}\left(\sum_{i=1}^{D} x_i b_i\right) = b_k.$$

then

$$\nabla f(\bar{x}) = \begin{pmatrix} b_1 \\ \vdots \\ b_D \end{pmatrix} = \bar{b}.$$

2.2 c)

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

We can treat the terms separately.
Gradient of $\bar{b}^T \bar{x}$: From part (b),

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

Gradient of $\bar{x}^T A \bar{x}$: w write $\bar{x}^T A \bar{x}$ in index notation:

$$\bar{x}^T A \bar{x} = \sum_{i=1}^{D} \sum_{j=1}^{D} x_i a_{ij} x_j.$$

The k-th component of the gradient is

$$\frac{\partial}{\partial x_k}\left(\sum_{i=1}^{D}\sum_{j=1}^{D} x_i a_{ij} x_j\right) = \sum_{i,j}\frac{\partial}{\partial x_k}(x_i a_{ij} x_j).$$

Using the product rule and the fact that a_{ij} is a constant:

$$\frac{\partial}{\partial x_k}(x_i a_{ij} x_j) = a_{ij}\left(\delta_{ik}x_j + x_i\delta_{jk}\right) = a_{kj}x_j + a_{ik}x_i,$$

Then

$$\frac{\partial}{\partial x_k}(\bar{x}^T A\bar{x}) = \sum_j a_{kj}x_j + \sum_i a_{ik}x_i.$$

In vector–matrix form this is

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

Putting it all together:

$$\nabla f(\bar{x}) = \nabla_{\bar{x}}(\bar{x}^T A\bar{x}) + \nabla_{\bar{x}}(\bar{b}^T\bar{x}) + \nabla_{\bar{x}}c = (A + A^T)\bar{x} + \bar{b} + 0.$$

3 Exercise 3 (Estimating House Prices I)

The python file can be exectudeed by directing to the a1 folder, and then typing python3 housing_1.py . The source code is also placed in the appendisx.

3.1 a)

$$\hat{t} = \frac{1}{N}\sum_{n=1}^{N} t_n.$$

Using the training data, we compute:

$$Meanofhouseprices = 22.0166007905(in1000dollars)$$

Python snip:

```
mean_price = numpy.mean(t_train)
```

3.2 b)

we get the root-mean-square error as defined as:

$$\mathrm{RMSE}(t, t') = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(t_i - t'_i)^2}.$$

Applying this to the test set with the constant prediction $mean = 22.0166007905$, we obtain:

$$\mathrm{RMSE} = 9.6724779727$$

again also in 1000$ Python snip:

```
     def rmse(t, tp):
    """
    return numpy.sqrt(numpy.mean((t - tp)**2))

t_pred_mean = numpy.full_like(t_test, mean_price)

rmse_value = rmse(t_test, t_pred_mean)
```

3.3 c)

The scatter plot clearly shows that all predictions fall on a horizontal line at y = 22.02, while the true prices vary significantly (from about $5,000 to 50,000$). The red dashed line represents perfect predictions.

Figure 1: scatter plot

4 Exercise 4 (Estimating House Prices II)

In this exercise, we implement multivariate linear regression to predict house prices using the Boston Housing dataset.

4.1 a)

We implement a `LinearRegression` class with `fit` and `predict` methods. The `fit` method computes the optimal weights using the normal equation, which provides the closed-form solution for linear least-squares regression.

1. augment the input matrix $\mathbf{X}$ with a column of ones to account for the intercept term w_0:

$$\mathbf{X}_{\text{augmented}} = [1, x_1, x_2, \ldots, x_D]$$

2. Compute the optimal weights using the normal equation:

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

This formula minimizes the sum of squared errors:

$$L = \sum_{i=1}^{N} (\mathbf{w}^T \mathbf{x}_i - t_i)^2$$

4.2 b)

We first fit a linear regression model using only the first feature The resulting model has the form:

$$t = w_0 + w_1 \cdot \text{CRIM}$$

Optimal weights:

$$\hat{w}_0 \text{ (intercept)} = 23.6351$$
$$\hat{w}_1 \text{ (CRIM)} = -0.4328$$

The $\hat{w}_0 = 23.6351$ represents the predicted median house price (1000s) when the crime rate is zero. This baseline value of approximately \$23,635 represents the expected price in a crime-free area.

The $\hat{w}_1 = -0.4328$ indicates the relationship between crime rate and housing prices. for each unit increase in the per capita crime rate, the median house price decreases by approximately \$433

4.3 c)

$$t = w_0 + w_1 \cdot \text{CRIM} + w_2 \cdot \text{ZN} + w_3 \cdot \text{INDUS} + \ldots + w_{13} \cdot \text{LSTAT}$$

1. **NOX** ($\hat{w}_5 = -17.33$): Nitric oxide concentration has the strongest effect on house prices. Each unit increase in NOX (parts per 10 million) decreases the median house price by approximately \$17,326.

2. **RM** ($\hat{w}_6 = 3.99$): The average number of rooms per dwelling has the strongest positive effect. Each additional room increases the house price by approximately \$3,994, which makes sense

3. **CHAS** ($\hat{w}_4 = 2.29$): Properties bordering the Charles River are valued approximately \$2,293 higher than comparable properties that don't border the river.

4. **DIS** ($\hat{w}_8 = -1.29$): Greater weighted distance to employment centers decreases house prices by about \$1,287 per unit

5. **PTRATIO** ($\hat{w}_{11} = -0.81$): Higher pupil-teacher ratios (indicating lower school quality) decrease house prices by approximately \$815 per unit increase.

6. **LSTAT** ($\hat{w}_{13} = -0.46$): Higher percentages of lower-status population decrease house prices by about \$465 per percentage point.

Weight	Value	Feature
$\hat{w}_0$	31.3887	(intercept)
$\hat{w}_1$	-0.0596	CRIM
$\hat{w}_2$	0.0294	ZN
$\hat{w}_3$	-0.0291	INDUS
$\hat{w}_4$	2.2926	CHAS
$\hat{w}_5$	-17.3264	NOX
$\hat{w}_6$	3.9938	RM
$\hat{w}_7$	0.0032	AGE
$\hat{w}_8$	-1.2872	DIS
$\hat{w}_9$	0.3548	RAD
$\hat{w}_{10}$	-0.0156	TAX
$\hat{w}_{11}$	-0.8146	PTRATIO
$\hat{w}_{12}$	0.0118	B
$\hat{w}_{13}$	-0.4649	LSTAT

Table 1:

4.4 d)

We evaluate both models using the RMSE metric on the test set:

$$\text{RMSE}(\mathbf{t}, \mathbf{t}') = \sqrt{\frac{1}{N}\sum_{i=1}^{N} \|t_i - t'_i\|^2}$$

Model: RMSA
Single feature model CRIM only : 8.9549
All features model: 4.6883
Improvement: 4.2665 (47.6% reduction)

The single-feature model achieves an RMSE of 8.9549, which represents a notable improvement over the naive mean-based model from Exercise 3 (RMSE = 9.6725). This demonstrates that crime rate alone provides valuable predictive information about house prices.

4.5 The scatter plots

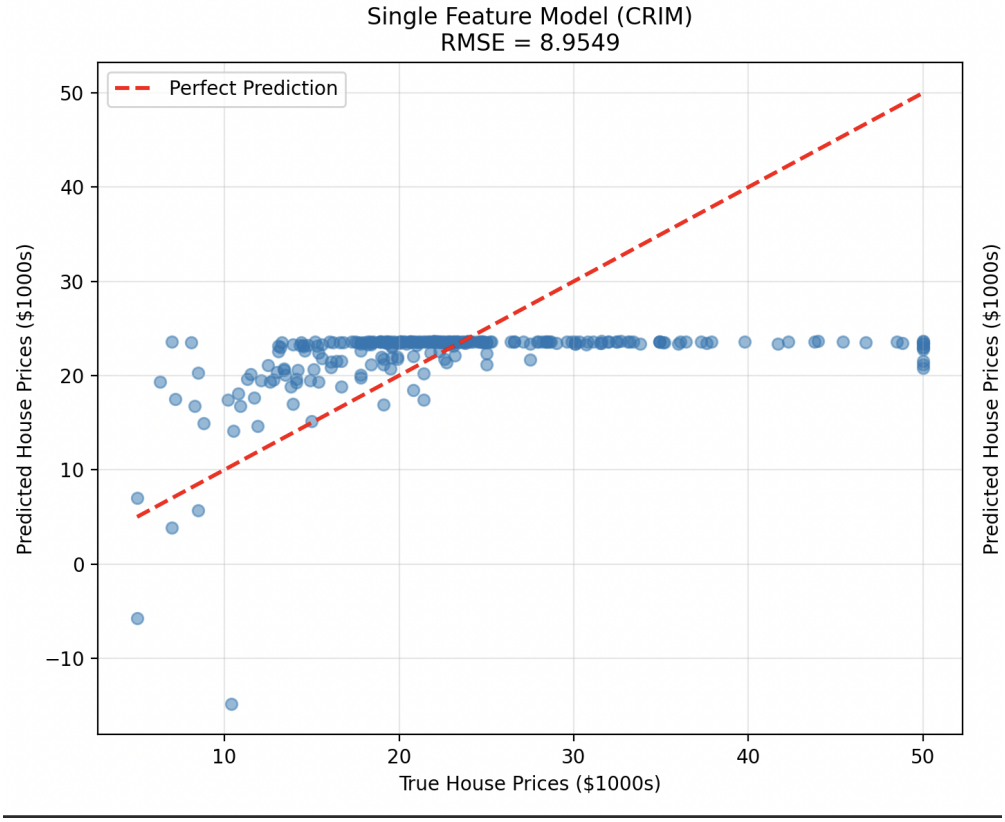

Figure 2: Enter Caption

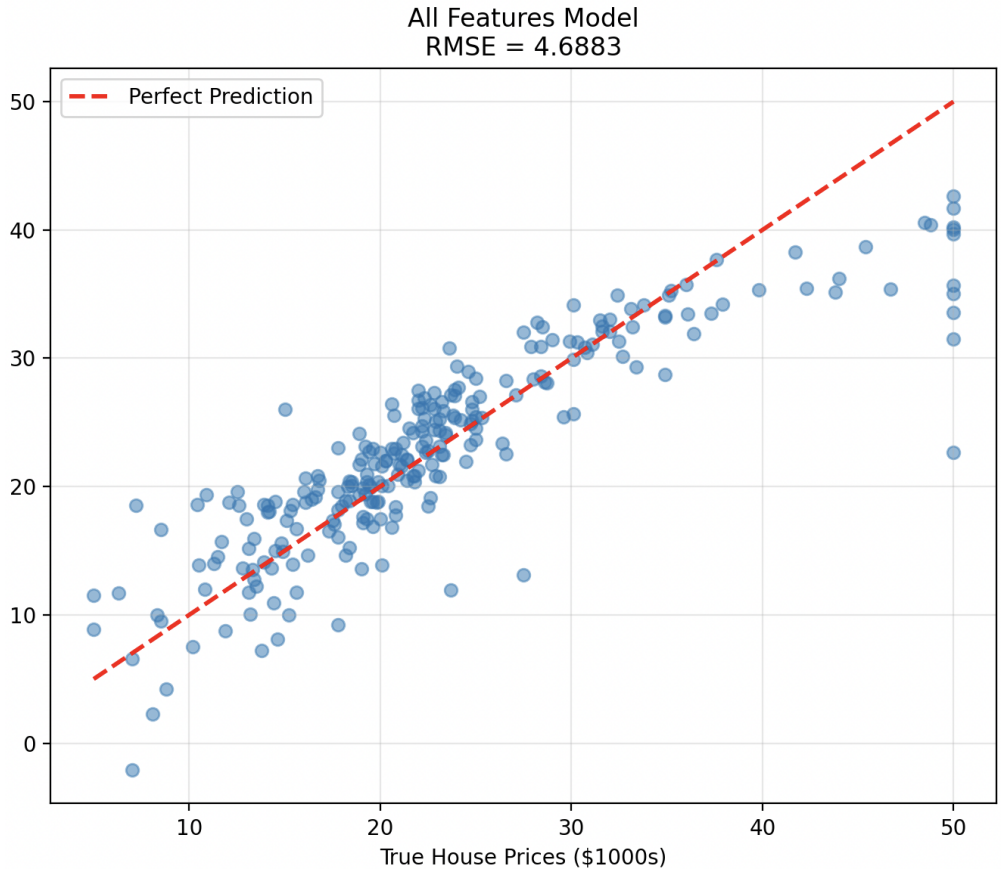

Figure 3: Enter Caption

left plotshows predictions that are somewhat correlated with true values, but with considerable scatter. the model produces some unrealistic predictions, including negative prices for areas with very high crime rates, which is physically impossible.

Right plot shows predictions much more clustered around the perfect prediction line. The points follow the ideal prediction line more closely across the entire range of house prices, from low-priced to high-priced properties.

5 Exercise 5 (Total Training Loss)

We need to find the optimal weights $\hat{\mathbf{w}}$ that minimize:

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

First we write this in matrix form. we let $\mathbf{X}$ be the design matrix where each row is $\mathbf{x}_n^T$, and $\mathbf{t}$ is the vector of targets. Then:

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

$$= \mathbf{w}^T \mathbf{X}^T \mathbf{X}\mathbf{w} - 2\mathbf{w}^T \mathbf{X}^T \mathbf{t} + \mathbf{t}^T \mathbf{t}$$

Taking the gradient with respect to $\mathbf{w}$:

$$\nabla_{\mathbf{w}} L = 2\mathbf{X}^T \mathbf{X}\mathbf{w} - 2\mathbf{X}^T \mathbf{t}$$

gradient to zero:

$$2\mathbf{X}^T \mathbf{X}\mathbf{w} - 2\mathbf{X}^T \mathbf{t} = 0$$

$$\mathbf{X}^T \mathbf{X}\mathbf{w} = \mathbf{X}^T \mathbf{t}$$

Solving for $\mathbf{w}$:

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

For the average loss L_{avg}, the gradient is:

$$\nabla_{\mathbf{w}} L_{avg} = \frac{1}{N} (2\mathbf{X}^T \mathbf{X}\mathbf{w} - 2\mathbf{X}^T \mathbf{t})$$

Setting this to zero gives the same equation $\mathbf{X}^T \mathbf{X}\mathbf{w} = \mathbf{X}^T \mathbf{t}$, so the optimal solution is identical. The $\frac{1}{N}$ factor cancels out when finding the minimum, which means both formulations give the same optimal weights.

6 Appendix

6.1 Exercise 3 code:

```
    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("\nMean of house prices on training set: $%.2f (in $1000's)" %
   mean_price)

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

t_pred_mean = numpy.full_like(t_test, mean_price)

rmse_value = rmse(t_test, t_pred_mean)
print("RMSE on test set using mean model: %.4f" % rmse_value)

# (c) visualization of results
plt.figure(figsize=(8, 6))
plt.scatter(t_test, t_pred_mean, alpha=0.5)
plt.xlabel('True House Prices ($1000s)')
plt.ylabel('Predicted House Prices ($1000s)')
```

```
plt.title('Mean Model: True vs Predicted House Prices')
plt.plot([t_test.min(), t_test.max()], [t_test.min(), t_test.max()],
         'r--', lw=2, label='Perfect Prediction')
plt.legend()
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.savefig('scatter_plot.png', dpi=300)
plt.show()
```

6.2 Exercise 2:

```
    import numpy
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])

#RMSE function
def rmse(t, tp):
    return numpy.sqrt(numpy.mean((t - tp)**2))

print("PART (b):")

# (b) fit linear regression using only the first feature
model_single = linreg.LinearRegression()
model_single.fit(X_train[:,0], t_train)

#weights printout and interpretation
print("\nWeights for single-feature model (CRIM only):")
print("  w0 (intercept): %.4f" % model_single.w[0, 0])
```

```
print("  w1 (CRIM coefficient): %.4f" % model_single.w[1, 0])
print("\nInterpretation:")
print("  - w0 = %.4f: The baseline house price when CRIM = 0" %
   model_single.w[0, 0])
print("  - w1 = %.4f: For each unit increase in crime rate," %
   model_single.w[1, 0])
print("    the house price decreases by $%.2f (in $1000s)" % abs(
   model_single.w[1, 0]))


print("PART (c):")

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

print("\nWeights for all-features model:")
feature_names = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE',
                 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT']
print("  w0 (intercept): %.4f" % model_all.w[0, 0])
for i, name in enumerate(feature_names):
    print("  w%d (%s): %.4f" % (i+1, name, model_all.w[i+1, 0]))

print("PART (d): Evaluation of Results")

# (d) evaluation of results

t_pred_single = model_single.predict(X_test[:,0])
rmse_single = rmse(t_test, t_pred_single)
t_pred_all = model_all.predict(X_test)
rmse_all = rmse(t_test, t_pred_all)

print("\nRMSE on test set:")
print("  Single feature model (CRIM only): %.4f" % rmse_single)
print("  All features model: %.4f" % rmse_all)
print("\nImprovement: %.4f (%.1f%% reduction in error)" %
      (rmse_single - rmse_all, 100 * (rmse_single - rmse_all) /
         rmse_single))

fig, axes = plt.subplots(1, 2, figsize=(14, 6))
```

```
axes[0].scatter(t_test, t_pred_single, alpha=0.5)
axes[0].plot([t_test.min(), t_test.max()], [t_test.min(), t_test.max()],
             'r--', lw=2, label='Perfect Prediction')
axes[0].set_xlabel('True House Prices ($1000s)')
axes[0].set_ylabel('Predicted House Prices ($1000s)')
axes[0].set_title('Single Feature Model (CRIM)\nRMSE = %.4f' %
   rmse_single)
axes[0].legend()
axes[0].grid(True, alpha=0.3)

axes[1].scatter(t_test, t_pred_all, alpha=0.5)
axes[1].plot([t_test.min(), t_test.max()], [t_test.min(), t_test.max()],
             'r--', lw=2, label='Perfect Prediction')
axes[1].set_xlabel('True House Prices ($1000s)')
axes[1].set_ylabel('Predicted House Prices ($1000s)')
axes[1].set_title('All Features Model\nRMSE = %.4f' % rmse_all)
axes[1].legend()
axes[1].grid(True, alpha=0.3)

plt.tight_layout()
plt.savefig('exercise4_scatter_plots.png', dpi=300)
plt.show()
```

6.3 linreg code:

```
    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):
        self.w = None
```

```
    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)

        n_samples = X.shape[0]
        X_augmented = numpy.concatenate([numpy.ones((n_samples, 1)), X],
           axis=1)

        self.w = numpy.linalg.inv(X_augmented.T @ X_augmented) @
           X_augmented.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]
        """

        if X.ndim == 1:
            X = X.reshape(-1, 1)

        n_samples = X.shape[0]
        X_augmented = numpy.concatenate([numpy.ones((n_samples, 1)), X],
           axis=1)

        predictions = X_augmented @ self.w
```

```
        return predictions
```