practical_exercise_7 , Methods 3, 2021, autumn semester

Solution, Lau Møller Andersen

November 18, 2021

Exercises and objectives

1) Estimate bias and variance based on a true underlying function

2) Fitting training data and applying it to test sets with and without regularization

For each question and sub-question, please indicate one of the three following answers:
i. I understood what was required of me
ii. I understood what was required of me, but I did not know how to fulfil the requirement
iii. I did not understand what was required of me

EXERCISE 1 - Estimate bias and variance based on a true underlying function

We can express regression as $y = f(x) + \epsilon$ with $E[\epsilon] = 0$ and $var(\epsilon) = \sigma^2$ (E means expected value)

For a given point: x_0, we can decompose the expected prediction error , $E[(y_0 - \hat{f}(x_0))^2]$ into three parts - **bias**, **variance** and **irreducible error** (the first two together are the **reducible error**):

The expected prediction error is, which we also call the **Mean Squared Error**:
$E[(y_0 - \hat{f}(x_0))^2] = bias(\hat{f}(x_0))^2 + var(\hat{f}(x_0)) + \sigma^2$

where **bias** is;

$bias(\hat{f}(x_0)) = E[\hat{f}(x_0)] - f(x_0)$

1) Create a function, $f(x)$ that squares its input. This is our **true** function
   i. generate data, y, based on an input range of [0, 6] with a spacing of 0.1. Call this x
   ii. add normally distributed noise to y with $\sigma = 5$ (set a seed to 7 `np.random.seed(7)`) to y and call it y_{noisy}
   iii. plot the true function and the generated points

2) Fit a linear regression using `LinearRegression` from `sklearn.linear_model` based on y_{noisy} and x (see code chunk below associated with Exercise 1.2)
   i. plot the fitted line (see the `.intercept_` and `.coef_` attributes of the `regressor` object) on top of the plot (from 1.1.iii)
   ii. now run the code chunk below associated with Exercise 1.2.ii - what does X_quadratic amount to?
   iii. do a quadratic and a fifth order fit as well and plot them (on top of the plot from 1.2.i)
3) Simulate 100 samples, each with sample size `len(x)` with $\sigma = 5$ normally distributed noise added on top of the true function
   i. do linear, quadratic and fifth-order fits for each of the 100 samples
   ii create a **new** figure, `plt.figure`, and plot the linear and the quadratic fits (colour them appropriately); highlight the true value for $x_0 = 3$. From the graphics alone, judge which fit has

the highest bias and which has the highest variance

ii. create a **new** figure, `plt.figure`, and plot the quadratic and the fifth-order fits (colour them appropriately); highlight the true value for $x_0 = 3$. From the graphics alone, judge which fit has the highest bias and which has the highest variance

iii. estimate the **bias** and **variance** at x_0 for the linear, the quadratic and the fifth-order fits (the expected value $E[\hat{f}(x_0)] - f(x_0)$ is found by taking the mean of all the simulated, $\hat{f}(x_0)$, differences)

iv. show how the **squared bias** and the **variance** is related to the complexity of the fitted models

v. simulate **epsilon**: `epsilon = np.random.normal(scale=5, size=100)`. Based on your simulated values of **bias, variance and epsilon**, what is the **Mean Squared Error** for each of the three fits? Which fit is better according to this measure?

```
# Exercise 1.2
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit() ## what goes in here?
```

```
# Exercise 1.2.ii
from sklearn.preprocessing import PolynomialFeatures
quadratic = PolynomialFeatures(degree=2)
X_quadratic = quadratic.fit_transform(x.reshape(-1, 1))
regressor = LinearRegression()
regressor.fit() # what goes in here?
y_quadratic_hat # calculate this
```

SOLUTION 1.1

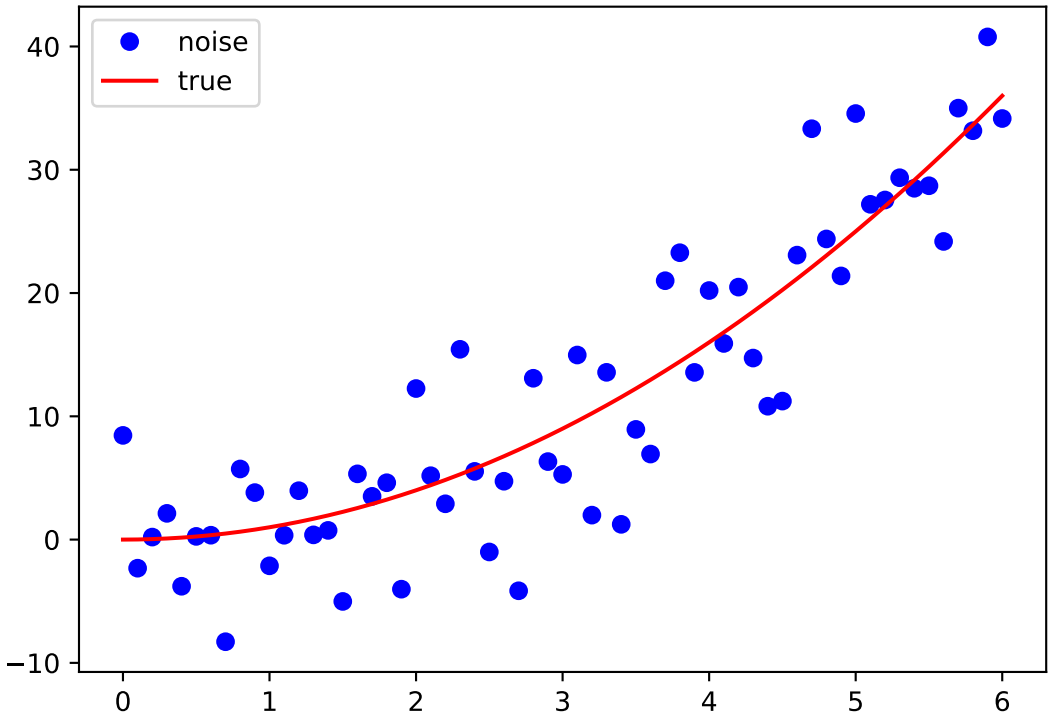
```
import numpy as np
import matplotlib.pyplot as plt
def square(x):
    return(x**2)

x = np.arange(0, 6.1, 0.1)
y_true = square(x)

np.random.seed(7) ## set a seed for replication

y_noise = y_true + np.random.normal(scale=5, size=len(y_true)) ## add normal noise

plt.figure()
plt.plot(x, y_noise, 'bo')
plt.plot(x, y_true, 'r-')
plt.legend(['noise', 'true'])
plt.show()
```



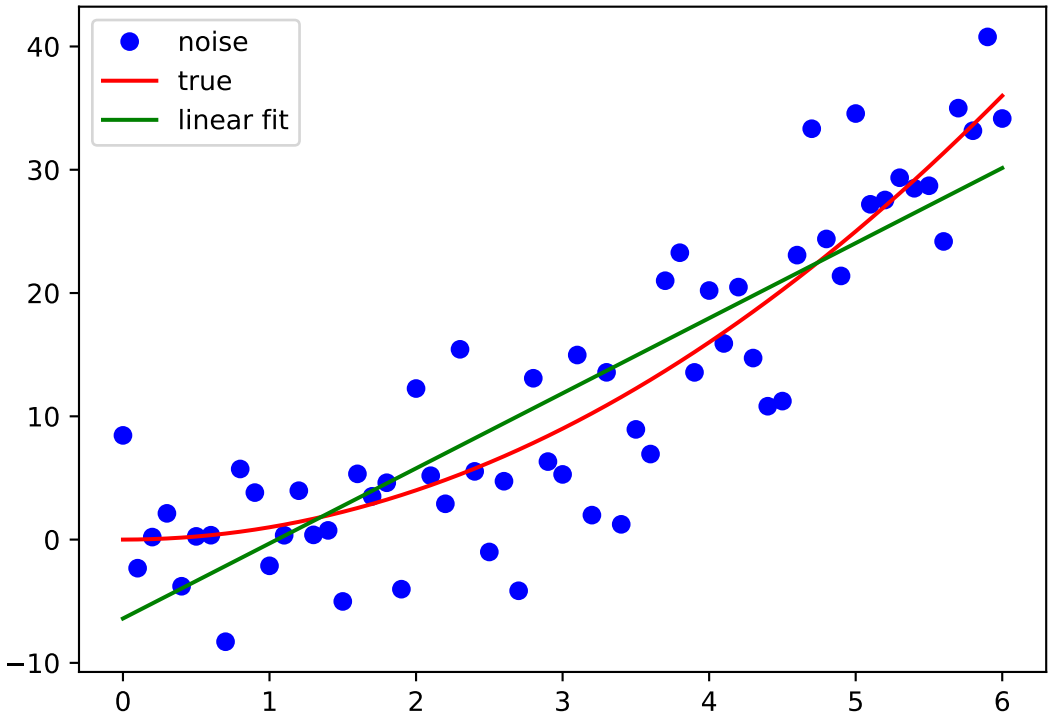
SOLUTION 1.2

```
# Exercise 2.2
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(x.reshape(-1, 1), y_noise) ## the reshaping is necessary, see error you get if you don't
```

```
## LinearRegression()
```

```
beta0 = regressor.intercept_
beta1 = regressor.coef_[0]
y_linear_hat = beta0 + beta1 * x ## fitted line

plt.figure()
plt.plot(x, y_noise, 'bo')
plt.plot(x, y_true, 'r-')
plt.plot(x, y_linear_hat, 'g-')
plt.legend(['noise', 'true', 'linear fit'])
plt.show()
```



SOLUTION 1.2.ii

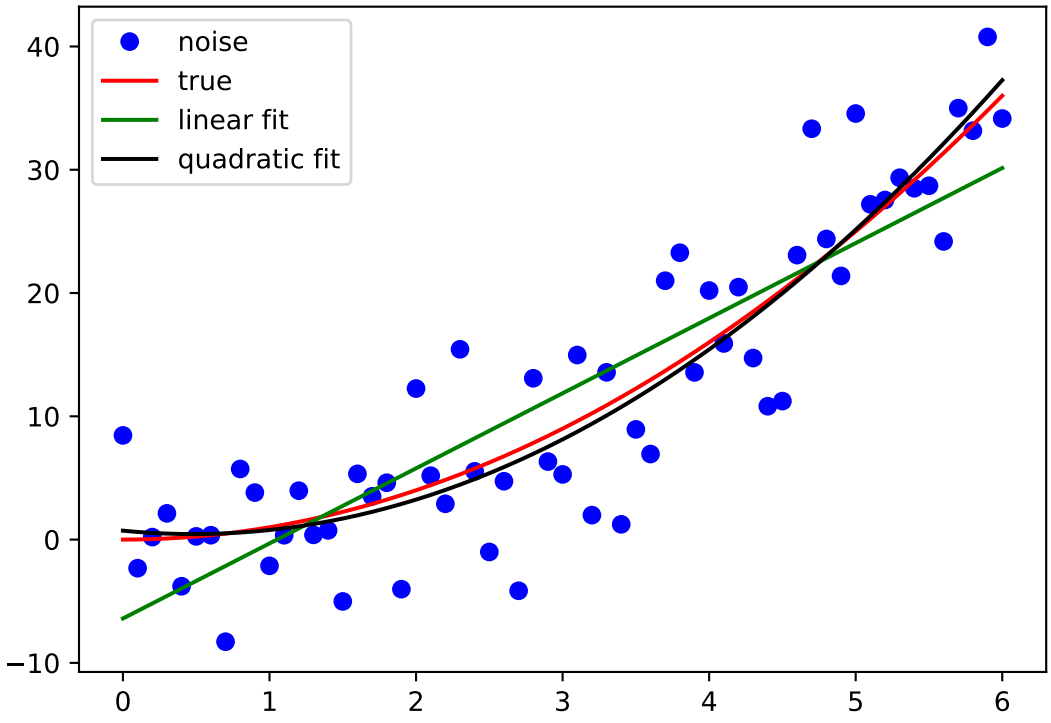
```
# Exercise 2.2.ii
from sklearn.preprocessing import PolynomialFeatures
quadratic = PolynomialFeatures(degree=2)
X_quadratic = quadratic.fit_transform(x.reshape(-1, 1)) # transform the X, such that you get a third co
regressor = LinearRegression()
regressor.fit(X_quadratic, y_noise)
```

```
## LinearRegression()
```

```
y_quadratic_hat = regressor.intercept_ + regressor.coef_[1] * x + regressor.coef_[2] * x**2

plt.figure()
plt.plot(x, y_noise, 'bo')
plt.plot(x, y_true, 'r-')
plt.plot(x, y_linear_hat, 'g-')
plt.plot(x, y_quadratic_hat, 'k-')
plt.legend(['noise', 'true', 'linear fit', 'quadratic fit'])
plt.show()

# Exercise 2.2.ii
```

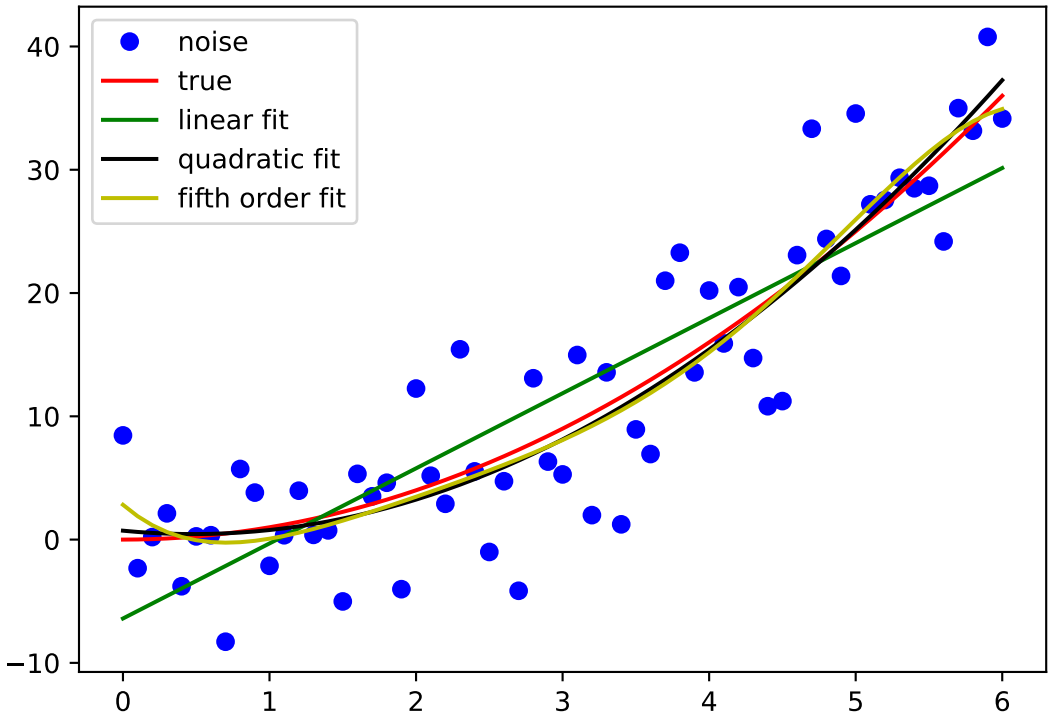



```
fifth = PolynomialFeatures(degree=5)
X_fifth = fifth.fit_transform(x.reshape(-1, 1))
regressor = LinearRegression()
regressor.fit(X_fifth, y_noise)
```

```
## LinearRegression()
```

```
y_fifth_hat = regressor.intercept_ + regressor.coef_[1] * x + regressor.coef_[2] * x**2 + regressor.coe

plt.figure()
plt.plot(x, y_noise, 'bo')
plt.plot(x, y_true, 'r-')
plt.plot(x, y_linear_hat, 'g-')
plt.plot(x, y_quadratic_hat, 'k-')
plt.plot(x, y_fifth_hat, 'y-')
plt.legend(['noise', 'true', 'linear fit', 'quadratic fit', 'fifth order fit'])
plt.show()
```

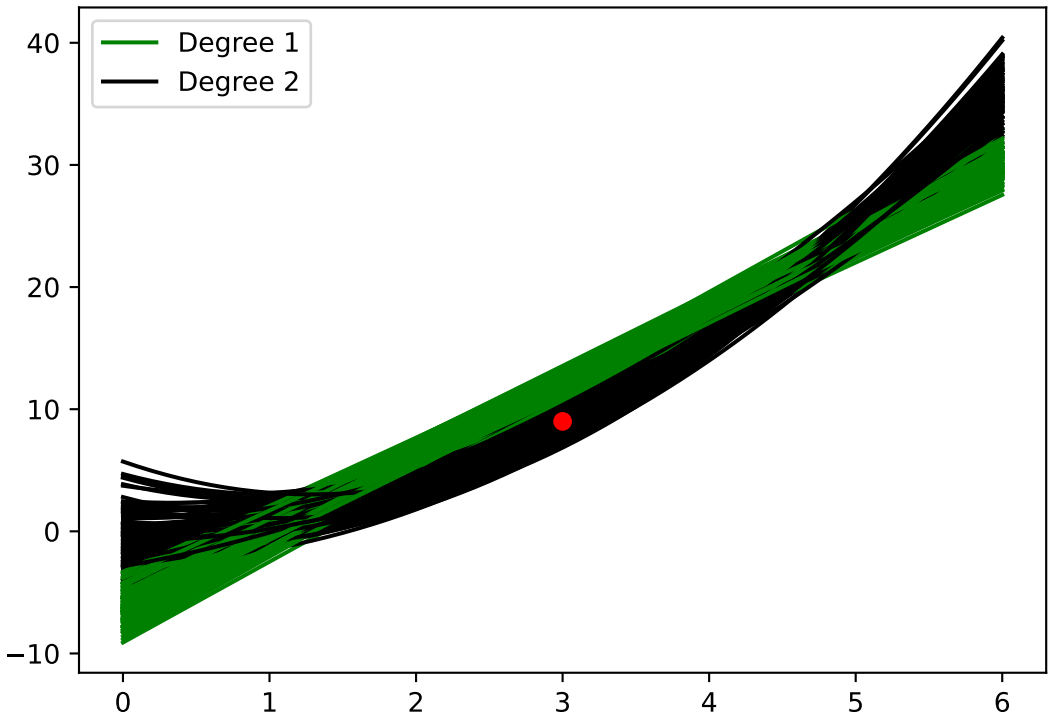


SOLUTION 1.3

```
def simulate_fit_and_plot(x, n_samples, degrees, sigma, x_highlight):
    predictions = list() # these are the predictions I'll get out for x_0
    for degree in degrees:
        predictions.append(list()) # preparing a list for each polynomial reg
    np.random.seed(7)
    plt.figure()
    colour_strings = ['g-', 'k-', 'k-', 'k-', 'y-']
    for _ in range(n_samples):
        y_true = x**2 # our true function
        # draw a new sample from the population with noise on to
        y_sample = y_true + np.random.normal(scale=sigma, size=len(x))
        # loop through your polynomial fits
        for degree_index, degree in enumerate(degrees):
            poly = PolynomialFeatures(degree=degree)
            X = poly.fit_transform(x.reshape(-1, 1))
            regressor = LinearRegression(fit_intercept=False)
            regressor.fit(X, y_sample)
            y_hat = np.zeros(len(x))
            for number in range(degree+1):
                # build y_hat, adding the contribution of each coefficient one
                # at a time
                y_hat += regressor.coef_[number] * x**number
            ## find the prediction at x_0 and append it to the appropriate list
            predictions[degree_index].append(y_hat[x == x_highlight][0])
            plt.plot(x, y_hat, colour_strings[degree-1]) # plot the drawn sample
```

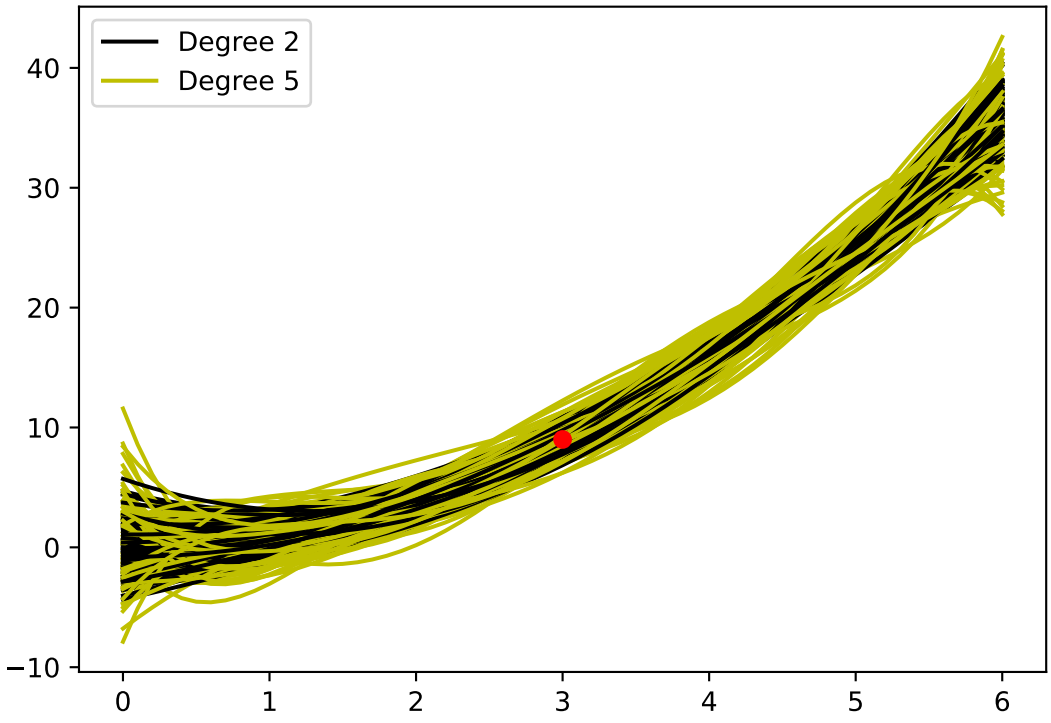
```
    plt.plot(x_highlight, x_highlight**2, 'ro') # plot the true point
    plt.legend(['Degree ' + str(degrees[0]), 'Degree ' + str(degrees[1])])
    plt.show()
    return predictions

# I do not run all three together, since the plot may be messy
predictions_1_and_2 = simulate_fit_and_plot(x, 100, [1, 2], 5, 3)
```




```
predictions_2_and_5 = simulate_fit_and_plot(x, 100, [2, 5], 5, 3)

# Using the formula above
```




```
bias_1 = np.mean(predictions_1_and_2[0]) - y_true[x == 3]
bias_2 = np.mean(predictions_1_and_2[1]) - y_true[x == 3]
bias_5 = np.mean(predictions_2_and_5[1]) - y_true[x == 3]

# Using the formula above
variance_1 = np.var(predictions_1_and_2[0])
variance_2 = np.var(predictions_1_and_2[1])
variance_5 = np.var(predictions_2_and_5[1])

epsilon = np.random.normal(scale=5, size=100)

MSE_1 = bias_1**2 + variance_1 + np.var(epsilon)
MSE_2 = bias_2**2 + variance_2 + np.var(epsilon)
MSE_5 = bias_5**2 + variance_5 + np.var(epsilon)
```

EXERCISE 2: Fitting training data and applying it to test sets with and without regularization

All references to pages are made to this book: Raschka, S., 2015. Python Machine Learning. Packt Publishing Ltd.

1) Import the housing dataset using the upper chunk of code from p. 280
   i. and define the correlation matrix `cm` as done on p. 284

   ii. based on this matrix, do you expect collinearity can be an issue if we run multiple linear regression

by fitting MEDV on LSTAT, INDUS, NOX and RM?

2) Fit MEDV on LSTAT, INDUS, NOX and RM (standardize all five variables by using `StandardScaler.fit_transform`, (`from sklearn.preprocessing import StandardScaler`) by doing multiple linear regression using `LinearRegressionGD` as defined on pp. 285-286
   i. how much does the solution improve in terms of the cost function if you go through 40 iterations instead of the default of 20 iterations?

   ii. how does the residual sum of squares based on the analytic solution (Ordinary Least Squares) compare to the cost after 40 iterations?
   iii. Bonus question: how many iterations do you need before the Ordinary Least Squares and the Gradient Descent solutions result in numerically identical residual sums of squares?

3) Build your own cross-validator function. This function should randomly split the data into k equally sized folds (see figure p. 176) (see the code chunk associated with exercise 2.3). It should also return the Mean Squared Error for each of the folds
   i. Cross-validate the fits of your model from Exercise 2.2. Run 11 folds and run 500 iterations for each fit

   ii. What is the mean of the mean squared errors over all 11 folds?

4) Now, we will do a Ridge Regression. Use `Ridge` (see code chunk associated with Exercise 2.4) to find the optimal `alpha` parameter (λ)
   i. Find the *MSE* (the mean of the *MSE's* associated each fold) associated with a reasonable range of `alpha` values (you need to find the minimum)

   ii. Plot the *MSE* as a function of `alpha` (λ). Make sure to include an *MSE* for `alpha=0` as well

   iii. Find the *MSE* for the optimal `alpha`, compare its *MSE* to that of the OLS regression
   iv. Do the same steps for Lasso Regression `Lasso` (2.4.i.-2.4.iii.)
   v. Describe the differences between these three models, (the optimal Lasso, the optimal Ridge and the OLS)

SOLUTION 2.1

```
import pandas as pd
import numpy as np
df = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/housing/housing.data', head
df.columns = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD',
              'TAX', 'PTRATIO', 'B', 'LSTAT', 'MEDV']
cols = ['LSTAT', 'INDUS', 'NOX', 'RM', 'MEDV']
cm = np.corrcoef(df[cols].values.T)
```

SOLUTION 2.2

```
class LinearRegressionGD(object):

    def __init__(self, eta=0.001, n_iter=20):
        self.eta = eta ## learning rate
        self.n_iter = n_iter

    def fit(self, X, y):
        self.w_ = np.zeros(1 + X.shape[1]) # out predictor variables
        self.cost_ = []

        for i in range(self.n_iter):
```

```
            output = self.net_input(X) # our linear predictor X@Beta
            errors = (y - output)
            self.w_[1:] += self.eta * X.T.dot(errors)
            self.w_[0] += self.eta * errors.sum()
            cost = (errors**2).sum() / 2.0
            self.cost_.append(cost)
        return self

    def net_input(self, X):
        return np.dot(X, self.w_[1:]) + self.w_[0]

    def predict(self, X):
        return self.net_input(X)

X = df[['LSTAT', 'INDUS', 'NOX', 'RM']].values
y = df['MEDV'].values
from sklearn.preprocessing import StandardScaler
X_std = StandardScaler().fit_transform(X)
## squeeze makes it have one dimension (removes all singleton dimensions)
y_std = np.squeeze(StandardScaler().fit_transform(y.reshape(-1, 1)))
LR = LinearRegressionGD()
LR.n_iter = 40
LR.fit(X_std, y_std)
```

```
## <__main__.LinearRegressionGD object at 0x7fdeccaa8f10>
```

```
cost_20 = LR.cost_[19]
cost_40 = LR.cost_[39]
print(cost_20 - cost_40) # improvement

## optimal
# OLS equation
```

```
## 0.005397815504494474
```

```
beta_hat = np.linalg.inv(X_std.T @ X_std) @ X_std.T @ y_std
y_hat = X_std @ beta_hat
RSS = np.sum((y_std - y_hat)**2)

print(cost_40 - RSS/2)
## found by trial and error - you could write a function that loops until it numerically becomes 0
```

```
## 2.0038510925246555e-05
```

```
LR.n_iter = 201
LR.fit(X_std, y_std)
```

```
## <__main__.LinearRegressionGD object at 0x7fdeccaa8f10>
```

```
print(LR.cost_[-1] - RSS/2)
```

```
## 0.0
```

```
def cross_validate(estimator, X, y, k): # estimator is the object created by initialising LinearRegress
    mses = list() # we want to return k mean squared errors
    fold_size = y.shape[0] // k # we do integer division to get a whole number of samples
    for fold in range(k): # loop through each of the folds
```

```
        X_train = ?
        y_train = ?
        X_test = ?
        y_test = ?

        # fit training data
        # predict on test data
        # calculate MSE

    return mses
```

SOLUTION 2.3

```
def cross_validate(estimator, X, y, k):
    mses = list()
    fold_size = y.shape[0] // k
    for fold in range(k):
        slice_begin = fold * fold_size
        slice_end   = (fold+1) * fold_size
        test_indices = np.arange(slice_begin, slice_end)
        X_train = np.delete(X, test_indices, axis=0) # removes the indices
        y_train = np.delete(y, test_indices)
        X_test = X[test_indices]
        y_test = y[test_indices]

        estimator.fit(X_train, y_train)
        y_test_hat = estimator.predict(X_test)
        mse = np.mean((y_test - y_test_hat)**2)
        mses.append(mse)

    return mses

LR = LinearRegressionGD(n_iter=500)
mses = cross_validate(LR, X_std, y_std, 11)
print(np.mean(mses))
```

```
## 0.4401522859127669
```

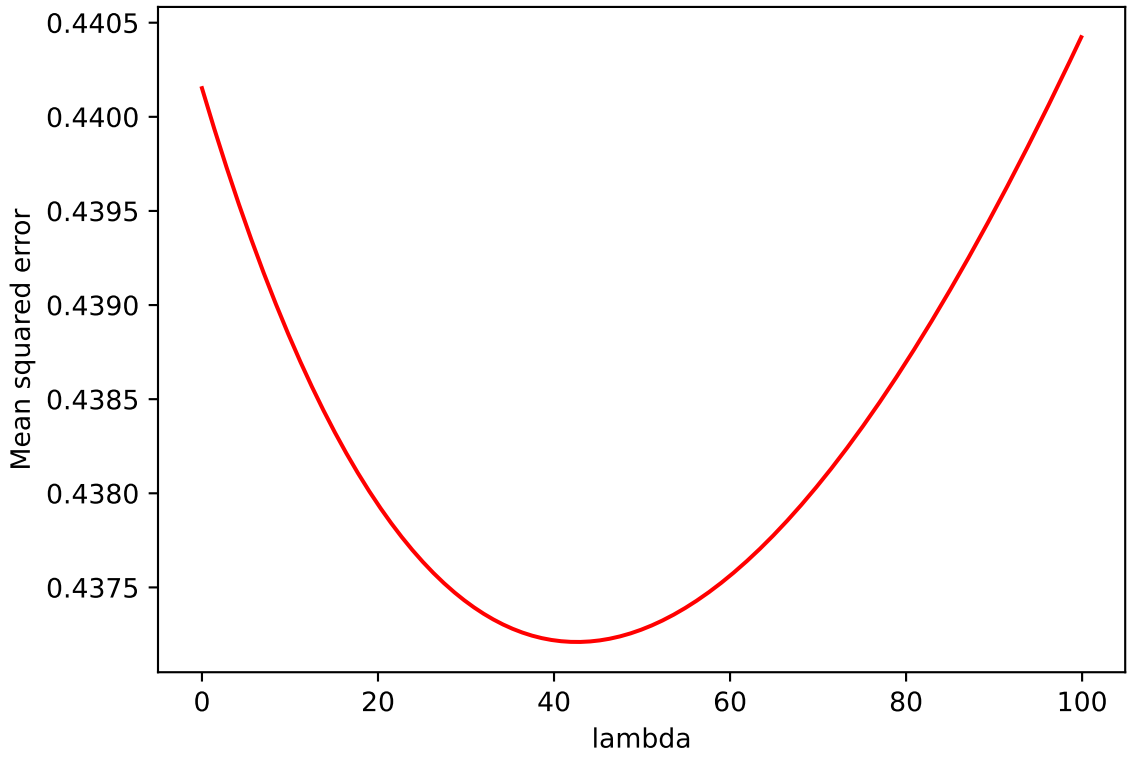
```
# Exercise 2.4
from sklearn.linear_model import Ridge, Lasso
RR = Ridge(alpha=?)
LassoR = Lasso(alpha)
```

SOLUTION 2.4

```
from sklearn.linear_model import Ridge
alpha_range = np.arange(0.1, 100, 0.1)
mse_list = [None] * len(alpha_range) # initialise with
for alpha_index, alpha in enumerate(alpha_range):
    RR = Ridge(alpha=alpha)
    mse_list[alpha_index] = np.mean(cross_validate(RR, X_std, y_std, 11))

plt.figure()
plot_alpha_range = np.concatenate((np.array([0]), alpha_range)) # include OLS
LR = LinearRegressionGD(n_iter=500)
```

```
# include OLS
plot_mse_list = mse_list.copy() # don't change the original
plot_mse_list.insert(0, np.mean(cross_validate(LR, X_std, y_std, 11)))
plt.plot(plot_alpha_range, plot_mse_list, 'r')
plt.xlabel('lambda')
plt.ylabel('Mean squared error')
plt.show()
```




```
print(np.min(mse_list))
```

```
## 0.43721008078749807
```

```
print(np.min(mse_list) -  np.mean(mses)) # mses is from OLS
```

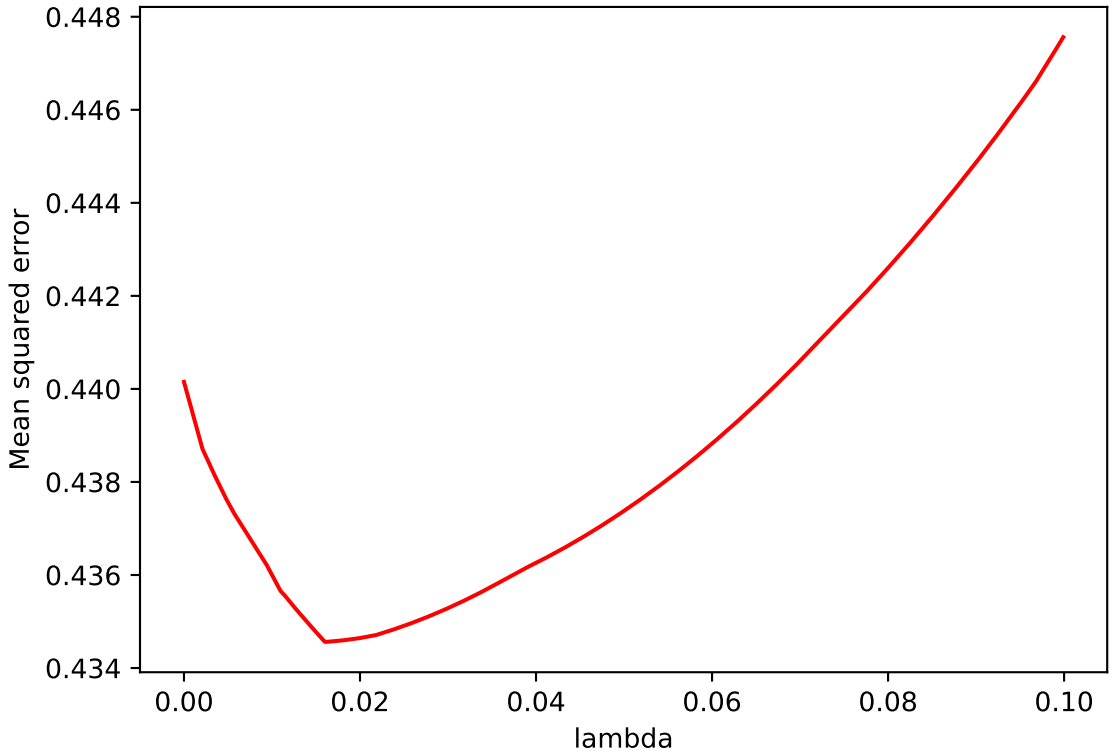
```
## -0.0029422051252688264
```

SOLUTION 2.5

```
from sklearn.linear_model import Lasso
alpha_range = np.arange(0.0001, 0.1, 0.0001)
mse_list = [None] * len(alpha_range)
for alpha_index, alpha in enumerate(alpha_range):
    LASSO = Lasso(alpha=alpha)
    mse_list[alpha_index] = np.mean(cross_validate(LASSO, X_std, y_std, 11))

plt.figure()
plot_alpha_range = np.concatenate((np.array([0]), alpha_range)) # include OLS
LR = LinearRegressionGD(n_iter=500)
```

```
# include OLS
plot_mse_list = mse_list.copy() # don't change the original
plot_mse_list.insert(0, np.mean(cross_validate(LR, X_std, y_std, 11)))
plt.plot(plot_alpha_range, plot_mse_list, 'r')
plt.xlabel('lambda')
plt.ylabel('Mean squared error')
plt.show()
```




```
print(np.min(mse_list))
```

```
## 0.4345599301054542
```

```
print(np.min(mse_list) -  np.mean(mses)) # mses is from OLS
```

```
## -0.00559235580731271
```