2 Programming

2.1

The CSV or XLS file contains a dataset for regression. There are 7750 samples with 25 features (described in the doc file). This data is for the purpose of bias correction of next-day maximum and minimum air temperatures forecast of the LDAPS model operated by the Korea Meteorological Administration over Seoul, South Korea. This data consists of summer data from 2013 to 2017. The input data is largely composed of the LDAPS model's next-day forecast data, in-situ maximum and minimum temperatures of present-day, and geographic auxiliary variables. There are two outputs (i.e., next-day maximum and minimum air temperatures) in this data. Hindcast validation was conducted for the period from 2015 to 2017. You need to delete the first two attributes (station and date) and use attributes 3-23 to predict attributes 24 and 25. Randomly split the data into two parts, one contains 80% of the samples and the other contains 20% of the samples. Use the first part as training data and train a linear regression model and make prediction on the second part. Report the training error and testing error in terms of RMSE. Repeat the splitting, training, and testing for 10 times. Use a loop and print the RMSEs in each trial.

2.2.

The classification data file contains the iris dataset. This is perhaps the best known database to be found in the pattern recognition literature. Fisher's paper is a classic in the field and is referenced frequently to this day. (See Duda Hart, for example.) The data set contains 3 classes of 50 instances each, where each class refers to a type of iris plant. One class is linearly separable from the other 2; the latter are NOT linearly separable from each other. There are fiver attributes: 1. sepal length in cm; 2. sepal width in cm; 3. petal length in cm; 4. petal width in cm; 5. class: – Iris Setosa – Iris Versicolour – Iris Virginica. You need to use the first attributes to predict the last attribute, namely, classifying the data into three classes. Randomly split the data into two parts, one contains 80% of the samples and the other contains 20% of the samples. Use the first part as training data and train a linear model and make classification on the second part. Report the training error and testing error in terms of classification error rate (number of miss-classified samples divided by number of all samples) Repeat the splitting, training, and testing for 10 times. Use a loop and print the classification errors in each trial. Note that you need to write the codes of learning the parameters by yourself. Do not use the classification or regression packages of Sklearn. You can use their tools to shuffle the data randomly for splitting.

Q 2.1 Air Temperatures Forecast

2.1.1 Introduction

In this problem, there are 7750 samples with 25 features for the purpose of bias correction of next-day maximum and minimum air temperatures forecast of the LDAPS model operated by the Korea Meteorological Administration over Seoul, South Korea. This data consists of summer data from 2013 to 2017.

The target is **to delete the first two attributes (station and date), and use attributes 3-23 to predict attributes 24 and 25** (i.e., next-day maximum and minimum air temperatures).

2.1.2 Main Theory

In this problem, the program applies the **linear regression model with multiple outputs** and uses the **closed form solution** for w to approach the prediction.

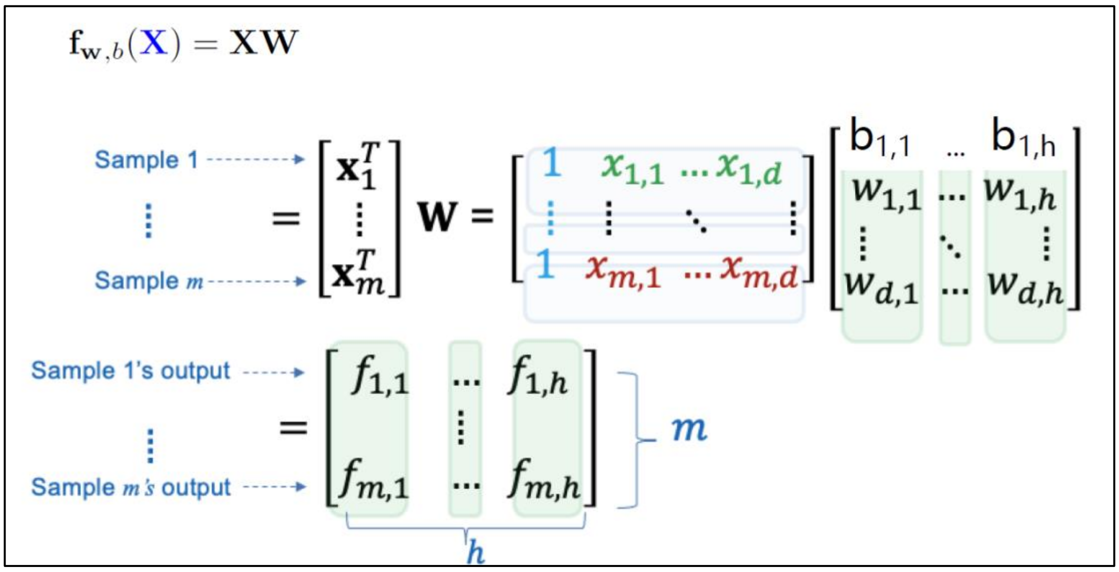

$f_{w,b}(X) = XW$
Sample 1
Sample m
$\begin{bmatrix} x_1^T \\ \vdots \\ x_m^T \end{bmatrix} W = \begin{bmatrix} 1 & x_{1,1} & \cdots & x_{1,d} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{m,1} & \cdots & x_{m,d} \end{bmatrix} \begin{bmatrix} b_{1,1} & \dots & b_{1,h} \\ w_{1,1} & \cdots & w_{1,h} \\ \vdots & \ddots & \vdots \\ w_{d,1} & \cdots & w_{d,h} \end{bmatrix}$
Sample 1's output
Sample m's output
$= \begin{bmatrix} f_{1,1} & \cdots & f_{1,h} \\ \vdots & \vdots & \vdots \\ f_{m,1} & \cdots & f_{m,h} \end{bmatrix}$
m
h

Figure 1: Theory

If $\mathbf{X}^\top \mathbf{X}$ is invertible, then

$$\boxed{\begin{array}{ll} \text{Learning:} & \hat{\mathbf{w}} = \left(\mathbf{X}^\top \mathbf{X}\right)^{-1} \mathbf{X}^\top \mathbf{y} \\ \text{Prediction:} & f_{\mathbf{w},b}\left(\mathbf{X}_{\text{new}}\right) = \mathbf{X}_{\text{new}}\, \hat{\mathbf{w}} \end{array}}$$

Figure 2: Analytical solution

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{(\hat{y}_i - y_i)^2}{n}}$$

$\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_n$ are predicted values

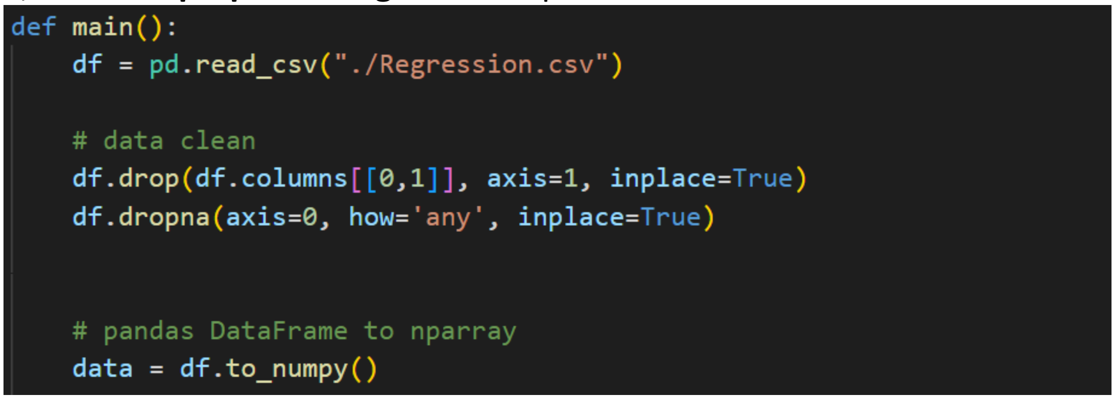
$y_1, y_2, \ldots, y_n$ are observed values

n is the number of observations

Figure 3: RMSE Formula

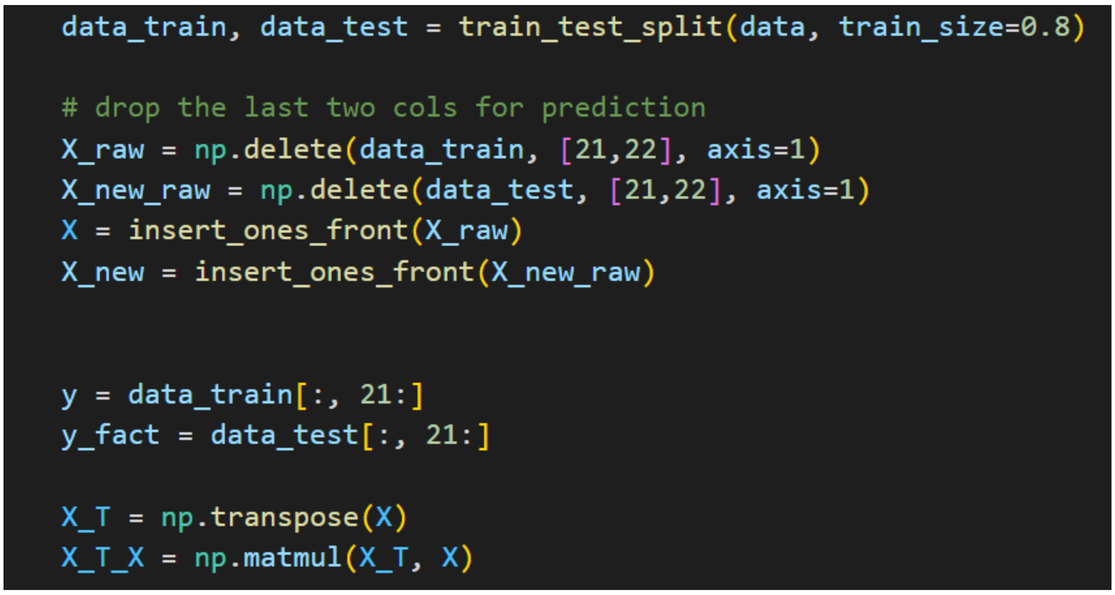
2.1.3 Implementation

First, do some **preprocessing** for the input data:



```
def main():
    df = pd.read_csv("./Regression.csv")

    # data clean
    df.drop(df.columns[[0,1]], axis=1, inplace=True)
    df.dropna(axis=0, how='any', inplace=True)


    # pandas DataFrame to nparray
    data = df.to_numpy()
```

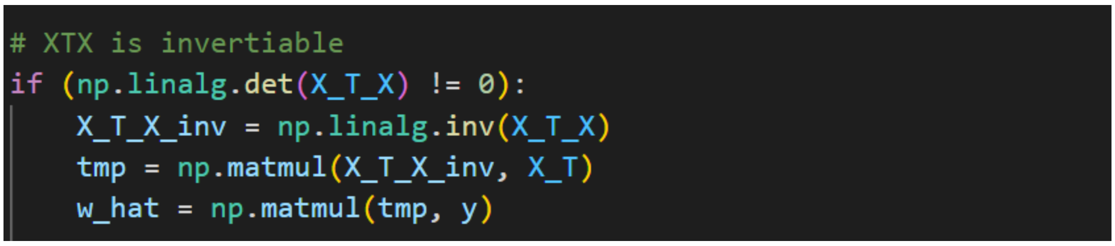
Figure 4: data clean



```
    data_train, data_test = train_test_split(data, train_size=0.8)

    # drop the last two cols for prediction
    X_raw = np.delete(data_train, [21,22], axis=1)
    X_new_raw = np.delete(data_test, [21,22], axis=1)
    X = insert_ones_front(X_raw)
    X_new = insert_ones_front(X_new_raw)


    y = data_train[:, 21:]
    y_fact = data_test[:, 21:]

    X_T = np.transpose(X)
    X_T_X = np.matmul(X_T, X)
```

Figure 5: data processing

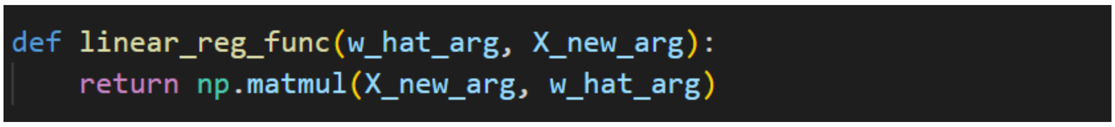
Second, calculate **w_hat** based on the theoretical background.



```
# XTX is invertiable
if (np.linalg.det(X_T_X) != 0):
    X_T_X_inv = np.linalg.inv(X_T_X)
    tmp = np.matmul(X_T_X_inv, X_T)
    w_hat = np.matmul(tmp, y)
```

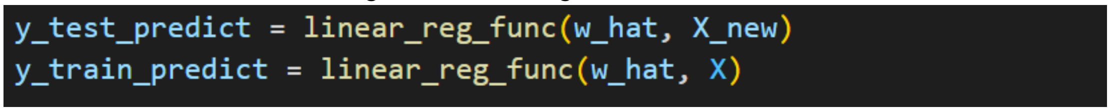
Figure 6: calc for w

Third, **fit the linear regression model** and do the prediction:



```
def linear_reg_func(w_hat_arg, X_new_arg):
    return np.matmul(X_new_arg, w_hat_arg)
```

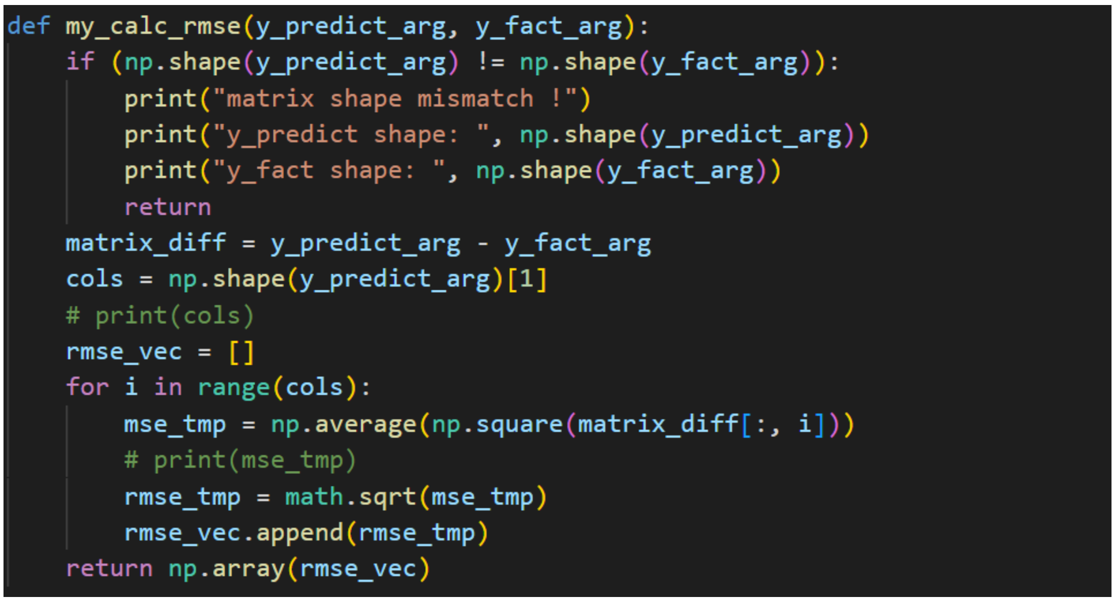
Figure 7: linear regression function



```
y_test_predict = linear_reg_func(w_hat, X_new)
y_train_predict = linear_reg_func(w_hat, X)
```

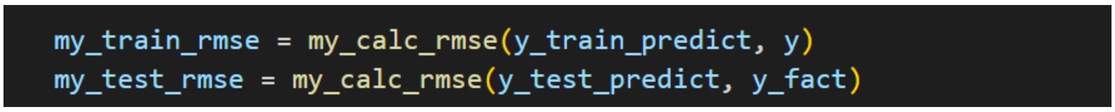
Figure 8: do prediction

Next, calculate **RMSE**:



```
def my_calc_rmse(y_predict_arg, y_fact_arg):
    if (np.shape(y_predict_arg) != np.shape(y_fact_arg)):
        print("matrix shape mismatch !")
        print("y_predict shape: ", np.shape(y_predict_arg))
        print("y_fact shape: ", np.shape(y_fact_arg))
        return
    matrix_diff = y_predict_arg - y_fact_arg
    cols = np.shape(y_predict_arg)[1]
    # print(cols)
    rmse_vec = []
    for i in range(cols):
        mse_tmp = np.average(np.square(matrix_diff[:, i]))
        # print(mse_tmp)
        rmse_tmp = math.sqrt(mse_tmp)
        rmse_vec.append(rmse_tmp)
    return np.array(rmse_vec)
```

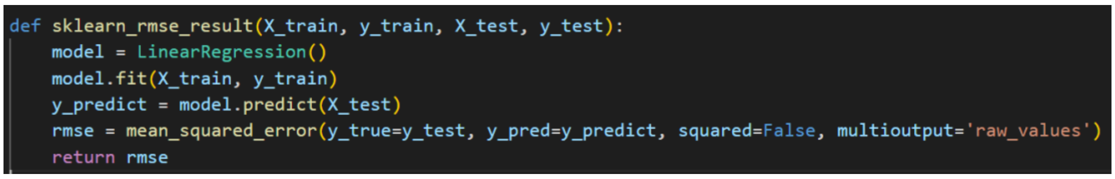
Figure 9: calc for RMSE



```
my_train_rmse = my_calc_rmse(y_train_predict, y)
my_test_rmse = my_calc_rmse(y_test_predict, y_fact)
```

Figure 10: calc for RMSE

Finally, **output the RMSEs under uniform average** and compare them with those calculated by the sklearn module:

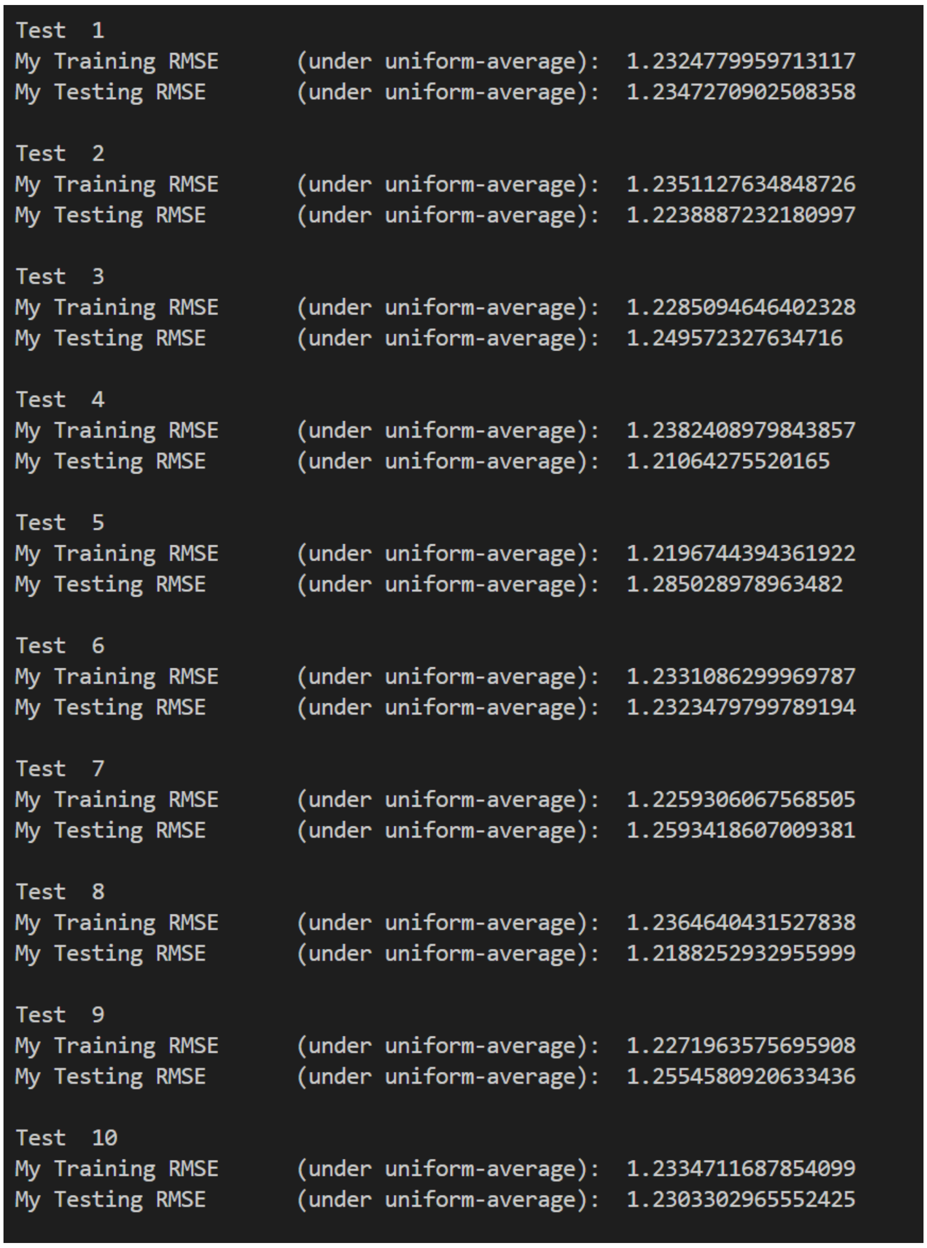


```
def sklearn_rmse_result(X_train, y_train, X_test, y_test):
    model = LinearRegression()
    model.fit(X_train, y_train)
    y_predict = model.predict(X_test)
    rmse = mean_squared_error(y_true=y_test, y_pred=y_predict, squared=False, multioutput='raw_values')
    return rmse
```

Figure 11: Sklearn Module

2.1.4 Result & Analysis

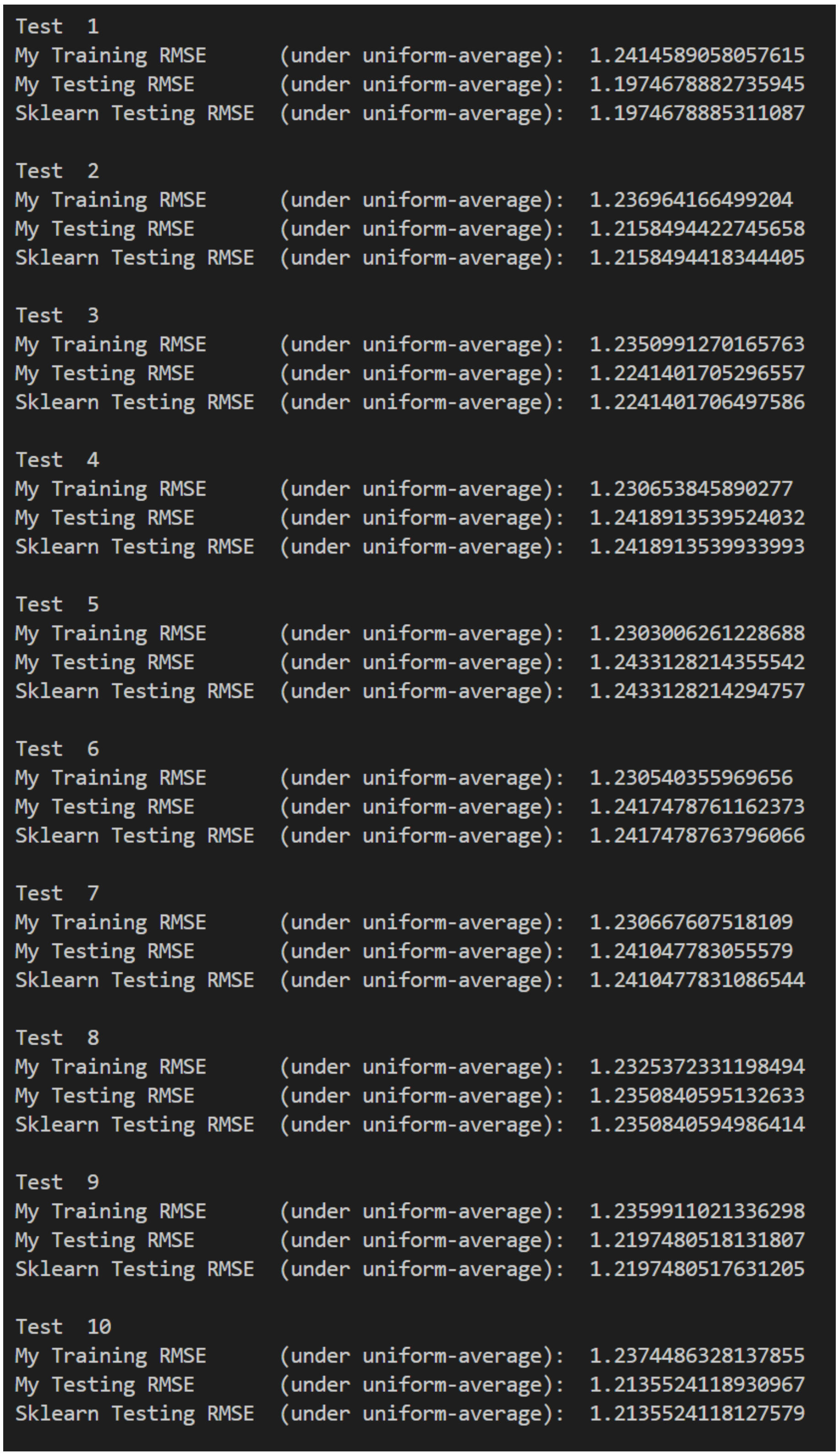
The test results and training results are given as below:



```
Test  1
My Training RMSE      (under uniform-average):  1.2324779959713117
My Testing RMSE       (under uniform-average):  1.2347270902508358

Test  2
My Training RMSE      (under uniform-average):  1.2351127634848726
My Testing RMSE       (under uniform-average):  1.2238887232180997

Test  3
My Training RMSE      (under uniform-average):  1.2285094646402328
My Testing RMSE       (under uniform-average):  1.249572327634716

Test  4
My Training RMSE      (under uniform-average):  1.2382408979843857
My Testing RMSE       (under uniform-average):  1.21064275520165

Test  5
My Training RMSE      (under uniform-average):  1.2196744394361922
My Testing RMSE       (under uniform-average):  1.285028978963482

Test  6
My Training RMSE      (under uniform-average):  1.2331086299969787
My Testing RMSE       (under uniform-average):  1.2323479799789194

Test  7
My Training RMSE      (under uniform-average):  1.2259306067568505
My Testing RMSE       (under uniform-average):  1.2593418607009381

Test  8
My Training RMSE      (under uniform-average):  1.2364640431527838
My Testing RMSE       (under uniform-average):  1.2188252932955999

Test  9
My Training RMSE      (under uniform-average):  1.2271963575695908
My Testing RMSE       (under uniform-average):  1.2554580920633436

Test  10
My Training RMSE      (under uniform-average):  1.2334711687854099
My Testing RMSE       (under uniform-average):  1.2303302965552425
```

Figure 12: Program1- Result

As illustrated in the figure2 above, since the program have multi-output, the program calculates the final RMSE **under uniform average**.



```
Test  1
My Training RMSE      (under uniform-average):  1.2414589058057615
My Testing RMSE       (under uniform-average):  1.1974678882735945
Sklearn Testing RMSE  (under uniform-average):  1.1974678885311087

Test  2
My Training RMSE      (under uniform-average):  1.236964166499204
My Testing RMSE       (under uniform-average):  1.2158494422745658
Sklearn Testing RMSE  (under uniform-average):  1.2158494418344405

Test  3
My Training RMSE      (under uniform-average):  1.2350991270165763
My Testing RMSE       (under uniform-average):  1.2241401705296557
Sklearn Testing RMSE  (under uniform-average):  1.2241401706497586

Test  4
My Training RMSE      (under uniform-average):  1.230653845890277
My Testing RMSE       (under uniform-average):  1.2418913539524032
Sklearn Testing RMSE  (under uniform-average):  1.2418913539933993

Test  5
My Training RMSE      (under uniform-average):  1.2303006261228688
My Testing RMSE       (under uniform-average):  1.2433128214355542
Sklearn Testing RMSE  (under uniform-average):  1.2433128214294757

Test  6
My Training RMSE      (under uniform-average):  1.230540355969656
My Testing RMSE       (under uniform-average):  1.2417478761162373
Sklearn Testing RMSE  (under uniform-average):  1.2417478763796066

Test  7
My Training RMSE      (under uniform-average):  1.230667607518109
My Testing RMSE       (under uniform-average):  1.241047783055579
Sklearn Testing RMSE  (under uniform-average):  1.2410477831086544

Test  8
My Training RMSE      (under uniform-average):  1.2325372331198494
My Testing RMSE       (under uniform-average):  1.2350840595132633
Sklearn Testing RMSE  (under uniform-average):  1.2350840594986414

Test  9
My Training RMSE      (under uniform-average):  1.2359911021336298
My Testing RMSE       (under uniform-average):  1.2197480518131807
Sklearn Testing RMSE  (under uniform-average):  1.2197480517631205

Test  10
My Training RMSE      (under uniform-average):  1.2374486328137855
My Testing RMSE       (under uniform-average):  1.2135524118930967
Sklearn Testing RMSE  (under uniform-average):  1.2135524118127579
```

Figure 13: Comparing Sklearn

Comparing the RMSE with the Sklearn, the results are nearly the same. It indicates that the **prediction is reliable**.

Q 2.2 Iris Classification

2.2.1 Introduction

In this problem, the data contains the **iris dataset**, which contains 3 classes of 50 instances each, where each class refers to a type of iris plant. One class is linearly separable from the other 2; the latter are NOT linearly separable from each other. There are five attributes: 1. sepal length in cm; 2. sepal width in cm; 3. petal length in cm; 4. petal width in cm; 5. Class.

The goal is to **use the first four attributes to predict the last attribute**, namely, classifying the data into three classes.

2.2.2 Main Theory

This is a classification problem, and TWO classes are NOT linearly separatable. Hence, we adopt **polynomial regression** to separate these data.

Linear model with basis expansion $\phi(\mathbf{x})$

$$\begin{aligned} f_{\mathbf{w},b}(\mathbf{x}) &= w_0 + \sum_{i=1}^{d} w_i x_i + \sum_{i=1}^{d}\sum_{j=1}^{d} w_{ij} x_i x_j + \sum_{i=1}^{d}\sum_{j=1}^{d}\sum_{k=1}^{d} w_{ijk} x_i x_j x_k + \dots \\ &= \phi(\mathbf{x})^\top \mathbf{w}, \end{aligned}$$

where

$$\begin{aligned} \phi(\mathbf{x}) &= [1, x_1, \dots, x_d, \dots, x_i x_j, \dots, x_i x_j x_k, \dots]^\top, \\ \mathbf{w} &= [w_0, w_1, \dots, w_d, \dots, w_{ij}, \dots, w_{ijk}, \dots]^\top. \end{aligned}$$

Figure 14: Polynomial expansion

$$\mathbf{P}(\mathbf{X}) = [\phi(\mathbf{x}_1)^\top; \dots; \phi(\mathbf{x}_m)^\top] \in \mathbb{R}^{m \times |\mathbf{w}|}.$$

Figure 15: Expression Form

Ridge regression with basis expansion $\mathbf{P}(\mathbf{X})$:

$$\begin{aligned} &\text{Learning:} && \hat{\mathbf{w}} = (\mathbf{P}^\top \mathbf{P} + \lambda \mathbf{I})^{-1} \mathbf{P}^\top \mathbf{y} \\ &\text{Prediction:} && f_{\mathbf{w},b}(\mathbf{P}(\mathbf{X}_{new})) = \mathbf{P}_{new} \hat{\mathbf{w}} \end{aligned}$$

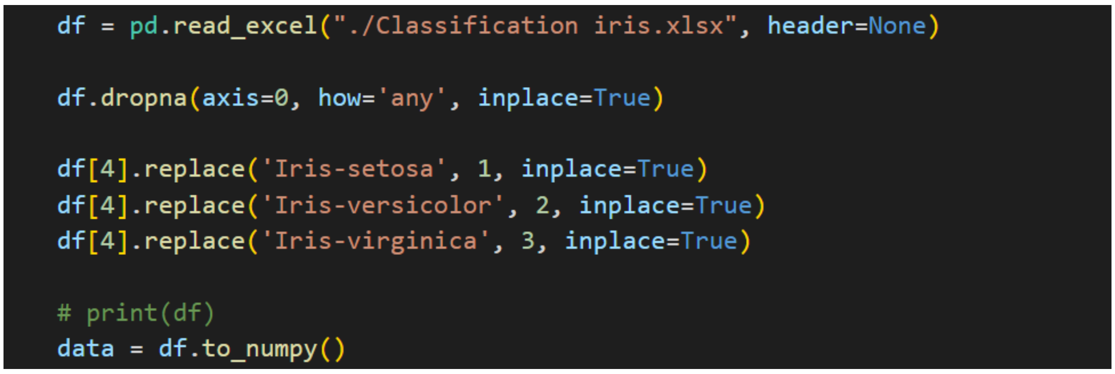
Figure 16: Ridge regression

- Multi-Category Prediction: $f_{\mathbf{w},b}(\mathbf{P}(\mathbf{X}_{new})) = \mathbf{argmax}_{i=1,\dots,C}(\mathbf{P}_{new}\hat{\mathbf{w}})$

Figure 17: muti-category prediction

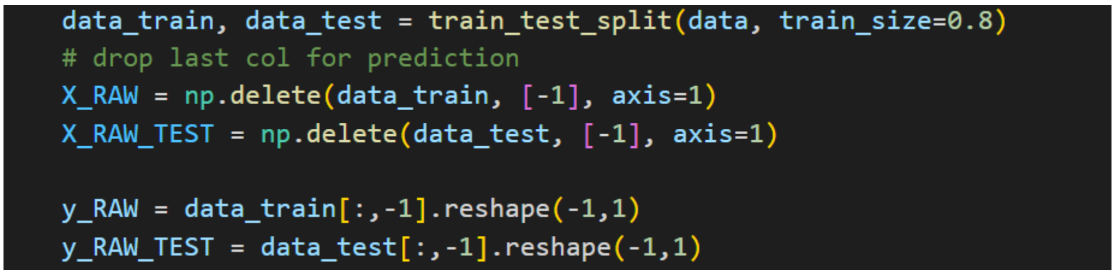
2.2.3 Implementation

First, do the data preprocessing:



```python
df = pd.read_excel("./Classification iris.xlsx", header=None)

df.dropna(axis=0, how='any', inplace=True)

df[4].replace('Iris-setosa', 1, inplace=True)
df[4].replace('Iris-versicolor', 2, inplace=True)
df[4].replace('Iris-virginica', 3, inplace=True)

# print(df)
data = df.to_numpy()
```

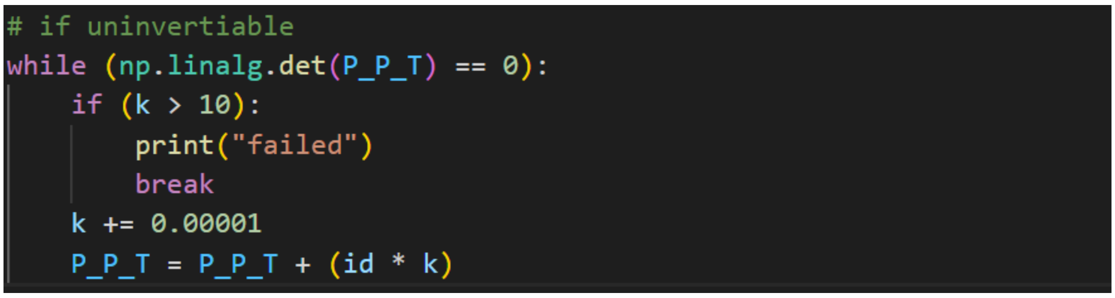
Figure 18: data preprocessing



```python
data_train, data_test = train_test_split(data, train_size=0.8)
# drop last col for prediction
X_RAW = np.delete(data_train, [-1], axis=1)
X_RAW_TEST = np.delete(data_test, [-1], axis=1)

y_RAW = data_train[:,-1].reshape(-1,1)
y_RAW_TEST = data_test[:,-1].reshape(-1,1)
```

Figure 19: Split and regularize data

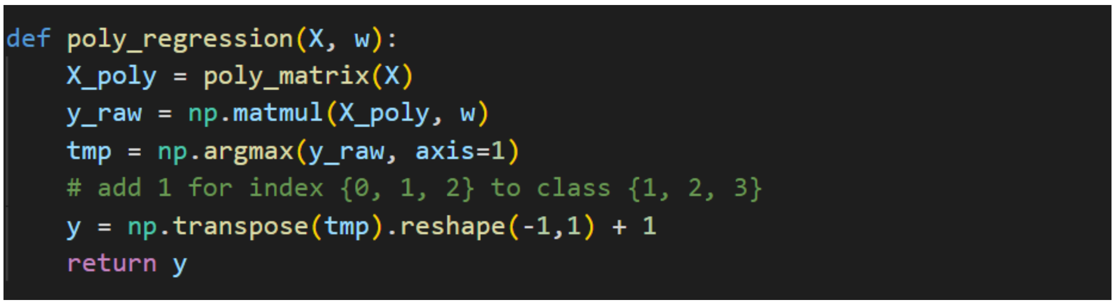
Second, apply ridge regression if P multiply by P transpose is not invertible:



```python
# if uninvertiable
while (np.linalg.det(P_P_T) == 0):
    if (k > 10):
        print("failed")
        break
    k += 0.00001
    P_P_T = P_P_T + (id * k)
```

Figure 20: use ridge if not invertible

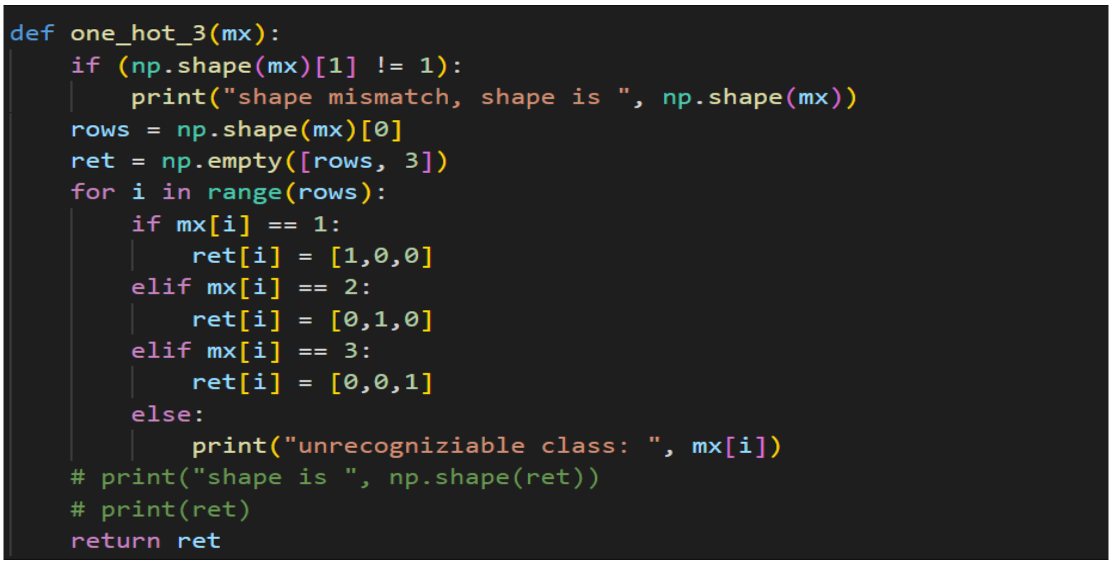
Third, change X to the polynomial basis expansion matrix:



```python
def poly_regression(X, w):
    X_poly = poly_matrix(X)
    y_raw = np.matmul(X_poly, w)
    tmp = np.argmax(y_raw, axis=1)
    # add 1 for index {0, 1, 2} to class {1, 2, 3}
    y = np.transpose(tmp).reshape(-1,1) + 1
    return y
```

Figure 21: basis expansion

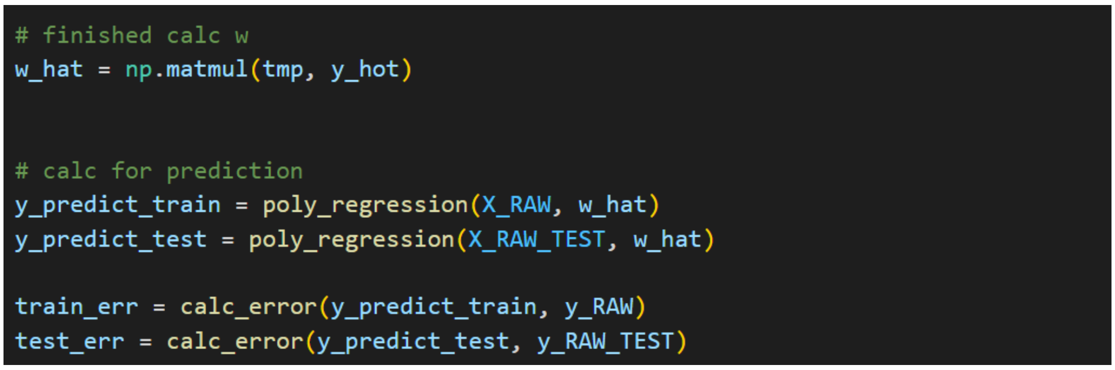
Next, change y to one-hot matrix:



```
def one_hot_3(mx):
    if (np.shape(mx)[1] != 1):
        print("shape mismatch, shape is ", np.shape(mx))
    rows = np.shape(mx)[0]
    ret = np.empty([rows, 3])
    for i in range(rows):
        if mx[i] == 1:
            ret[i] = [1,0,0]
        elif mx[i] == 2:
            ret[i] = [0,1,0]
        elif mx[i] == 3:
            ret[i] = [0,0,1]
        else:
            print("unrecognizable class: ", mx[i])
    # print("shape is ", np.shape(ret))
    # print(ret)
    return ret
```

Figure 22: Change y to one-hot matrix

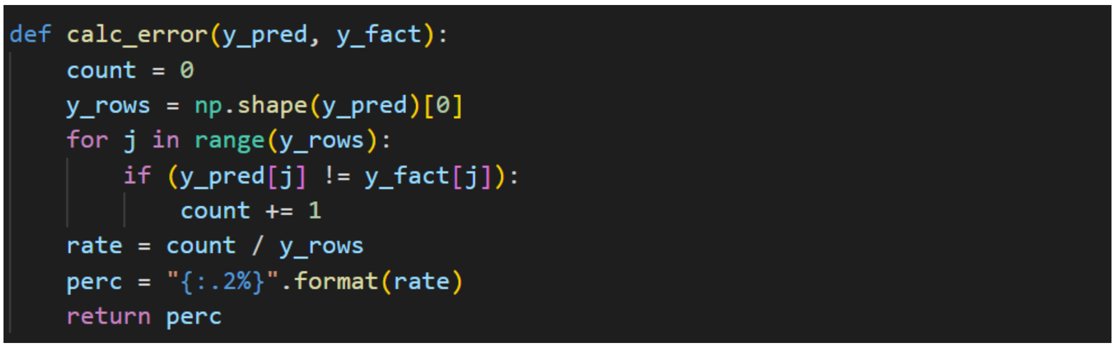
Then, calculate w_hat and do the prediction:



```
# finished calc w
w_hat = np.matmul(tmp, y_hot)


# calc for prediction
y_predict_train = poly_regression(X_RAW, w_hat)
y_predict_test = poly_regression(X_RAW_TEST, w_hat)

train_err = calc_error(y_predict_train, y_RAW)
test_err = calc_error(y_predict_test, y_RAW_TEST)
```

Figure 23: do prediction

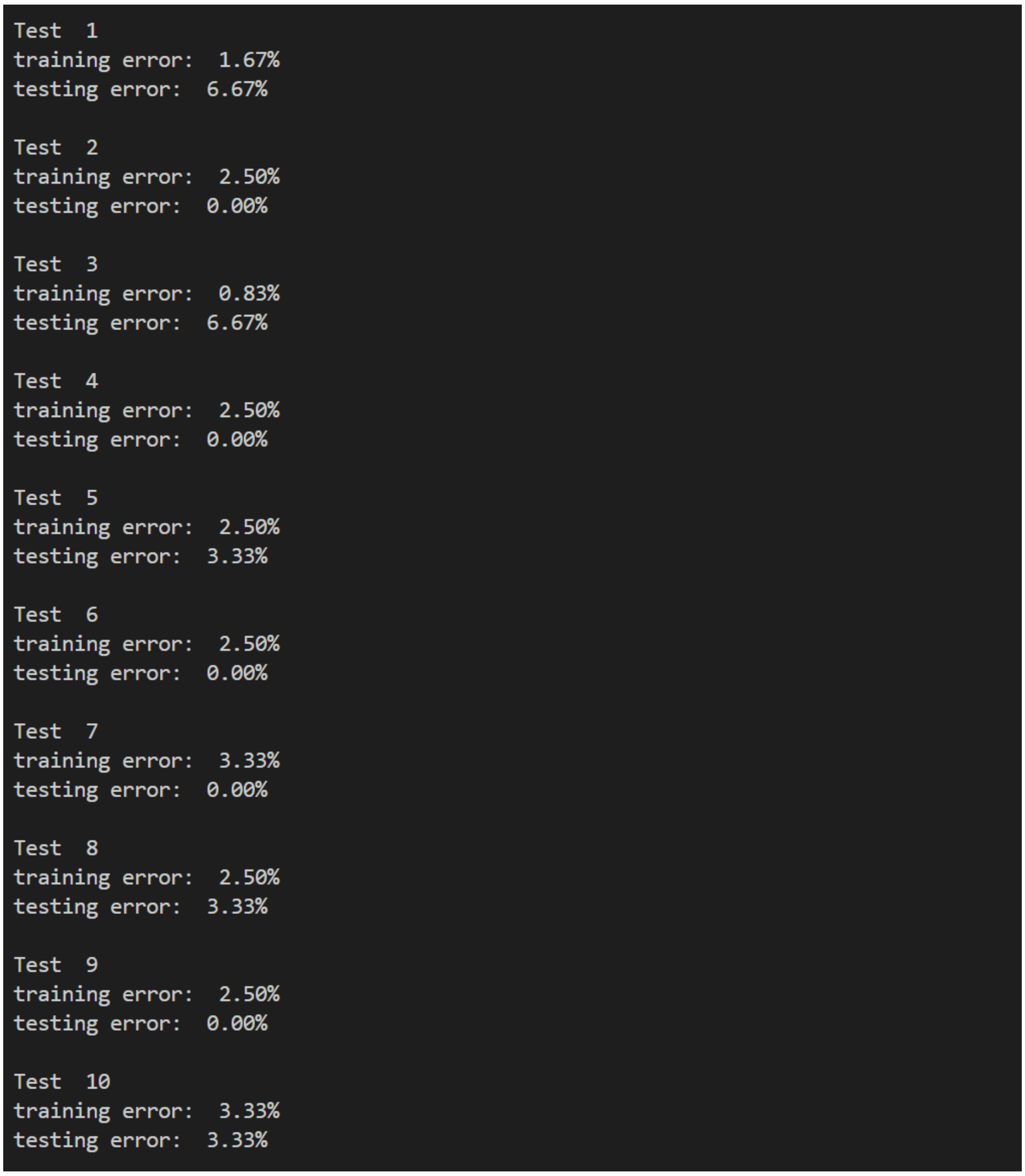
Finally, calculate and output error in terms of classification error rate (number of miss-classified samples divided by number of all samples)



```
def calc_error(y_pred, y_fact):
    count = 0
    y_rows = np.shape(y_pred)[0]
    for j in range(y_rows):
        if (y_pred[j] != y_fact[j]):
            count += 1
    rate = count / y_rows
    perc = "{:.2%}".format(rate)
    return perc
```

Figure 24: calc for error

2.2.4 Result & Analysis



```
Test  1
training error:  1.67%
testing error:  6.67%

Test  2
training error:  2.50%
testing error:  0.00%

Test  3
training error:  0.83%
testing error:  6.67%

Test  4
training error:  2.50%
testing error:  0.00%

Test  5
training error:  2.50%
testing error:  3.33%

Test  6
training error:  2.50%
testing error:  0.00%

Test  7
training error:  3.33%
testing error:  0.00%

Test  8
training error:  2.50%
testing error:  3.33%

Test  9
training error:  2.50%
testing error:  0.00%

Test  10
training error:  3.33%
testing error:  3.33%
```

Figure 25: error output

Note that since the sample only have 30 observations for the testing prediction, **testing error below and around 10%**, which means that the prediction only missed around 3 data out of 30, **should be acceptable**.