CA_Assignment3

June 6, 2018

1 Cognitive Algorithms - Assignment 4 (30 points)

Cognitive Algorithms
Summer term 2018
Technische Universität Berlin
Fachgebiet Maschinelles Lernen

Due on June 13, 2018 10am via ISIS

After completing all tasks, run the whole notebook so that the content of each cell is properly displayed. Make sure that the code was ran and the entire output (e.g. figures) is printed. Print the notebook as a PDF file and again make sure that all lines are readable - use line breaks in the Python Code '' if necessary. Points will be deducted, if code or content is not readable!

Upload the PDF file that contains a copy of your notebook on ISIS.

Group: Group08
Members: - Chen, Yang - Liu, Huiran - Smejkal, Karel - Tian, Qihang - Arat, Emrecan

2 Part 1: Theory (3 points)

A) (2 points) Explain briefly the goal of classification and regression. What is the difference between both tasks?

[Your answer for A) here]

The goal of classification is generalisation, that is, correct categorisation or prediction of new data; The goal of regression is to find out a model, which can generalise our data well and describe ~~the relationships among features of data~~. The difference between both tasks is that the labels in classification are finite. However, there exist infinite labels in regression.

B) Which statement is true? - [ ] Classification is a supervised learning task, regression is an unsupervised learning task.
- [ ] Classification is an unsupervised learning task, regression is a supervised learning task.
- [X] Classification and regression are both supervised learning tasks.
- [ ] Classification and regression are both unsupervised learning tasks.

2.1 # Part 2: Programming (27 points)

Note that part 2 of this assignment consists of two tasks.

2.1.1 Task 1: Ordinary Least Squares (9 points)

In this assignment you will implement a linear regression and predict two dimensional hand positions from electromyographic (EMG) recordings obtained with high-density electrode arrays on the lower arm. Download the data set `myo_data.mat` from the ISIS web site, if not done yet.

```
In [1]: import pylab as pl
        import scipy as sp
        import numpy as np
        from numpy.linalg import inv
        from scipy.interpolate import interp1d
        from scipy.io import loadmat
        %matplotlib inline
```

```
In [2]: def load_myo_data(fname):
            ''' Loads EMG data from <fname>
            '''
            # load the data
            data = loadmat(fname)
            # extract data and hand positions
            X = data['training_data']
            X = sp.log(X)
            Y = data['training_labels']
            #Split data into training and test data
            X_train = X[:, :5000]
            X_test = X[:, 5000:]
            Y_train = Y[:, :5000]
            Y_test = Y[:, 5000:]
            return X_train,Y_train,X_test, Y_test

        def train_ols(X_train, Y_train, llambda = 0):
            ''' Trains ordinary least squares (ols) regression
            Input:       X_train   -  DxN array of N data points with D features
                         Y         -  D2xN array of length N with D2 multiple labels
                         llabmda   -  Regularization parameter
            Output:      W         -  DxD2 array, linear mapping used to estimate labels
                                      with sp.dot(W.T, X)
            '''
            #your code here
            return (inv(X_train.dot(X_train.T)+llambda*np.identity(X_train.shape[0]))
                    .dot(X_train).dot(Y_train.T))

        def apply_ols(W, X_test):
            ''' Applys ordinary least squares (ols) regression
            Input:       X_test     -  DxN array of N data points with D features
                         W         -  DxD2 array, linear mapping used to estimate labels
                                      trained with train_ols
            Output:     Y_test     -  D2xN array
            '''
```

```
    #your code here
    return W.T.dot(X_test)

def predict_handposition():
    X_train,Y_train,X_test, Y_test = load_myo_data('myo_data.mat')
    # compute weight vector with linear regression
    W = train_ols(X_train, Y_train)
    # predict hand positions
    Y_hat_train = apply_ols(W, X_train)
    Y_hat_test = apply_ols(W, X_test)

    pl.figure(figsize=(8,6))
    pl.subplot(2,2,1)
    pl.plot(Y_train[0,:1000],Y_train[1,:1000],'.k',label = 'true')
    pl.plot(Y_hat_train[0,:1000],Y_hat_train[1,:1000],'.r', label = 'predicted')
    pl.title('Training Data')
    pl.xlabel('x position')
    pl.ylabel('y position')
    pl.legend(loc = 0)

    pl.subplot(2,2,2)
    pl.plot(Y_test[0,:1000],Y_test[1,:1000],'.k')
    pl.plot(Y_hat_test[0,:1000],Y_hat_test[1,:1000],'.r')
    pl.title('Test Data')
    pl.xlabel('x position')
    pl.ylabel('y position')

    pl.subplot(2,2,3)
    pl.plot(Y_train[1,:600], 'k', label = 'true')
    pl.plot(Y_hat_train[1,:600], 'r--', label = 'predicted')
    pl.xlabel('Time')
    pl.ylabel('y position')
    pl.legend(loc = 0)

    pl.subplot(2,2,4)
    pl.plot(Y_test[1,:600],'k')
    pl.plot(Y_hat_test[1,:600], 'r--')
    pl.xlabel('Time')
    pl.ylabel('y position')

def test_assignment4():
    ##Example without noise
    x_train = sp.array([[ 0,  0,  1 , 1],[ 0,  1,  0, 1]])
    y_train = sp.array([[0, 1, 1, 2]])
    w_est = train_ols(x_train, y_train)
    w_est_ridge = train_ols(x_train, y_train, llambda = 1)
    assert(sp.all(w_est.T == [[1, 1]]))
    assert(sp.all(w_est_ridge.T == [[.75, .75]]))
    y_est = apply_ols(w_est,x_train)
    assert(sp.all(y_train == y_est))
    print 'No-noise-case tests passed'

    ##Example with noise
    #Data generation
    w_true = 4
    X_train = sp.arange(10)
    X_train = X_train[None,:]
    Y_train = w_true * X_train + sp.random.normal(0,2,X_train.shape)
    #Regression
    w_est = train_ols(X_train, Y_train)
    Y_est = apply_ols(w_est,X_train)
    #Plot result
    pl.figure()
    pl.plot(X_train.T, Y_train.T, '+', label = 'Train Data')
    pl.plot(X_train.T, Y_est.T, label = 'Estimated regression')
    pl.xlabel('x')
    pl.ylabel('y')
    pl.legend(loc = 'lower right')
```

** A) (5 points)** Implement ordinary least squares regression (OLS) with an optional ridge parameter by completing the function stubs `ols_train` and `ols_apply`. In `ols_train`, you estimate a linear mapping W,

$$W = (X_{\text{train}} X_{\text{train}}^\top + \lambda I)^{-1} X_{\text{train}} Y_{\text{train}}^\top$$

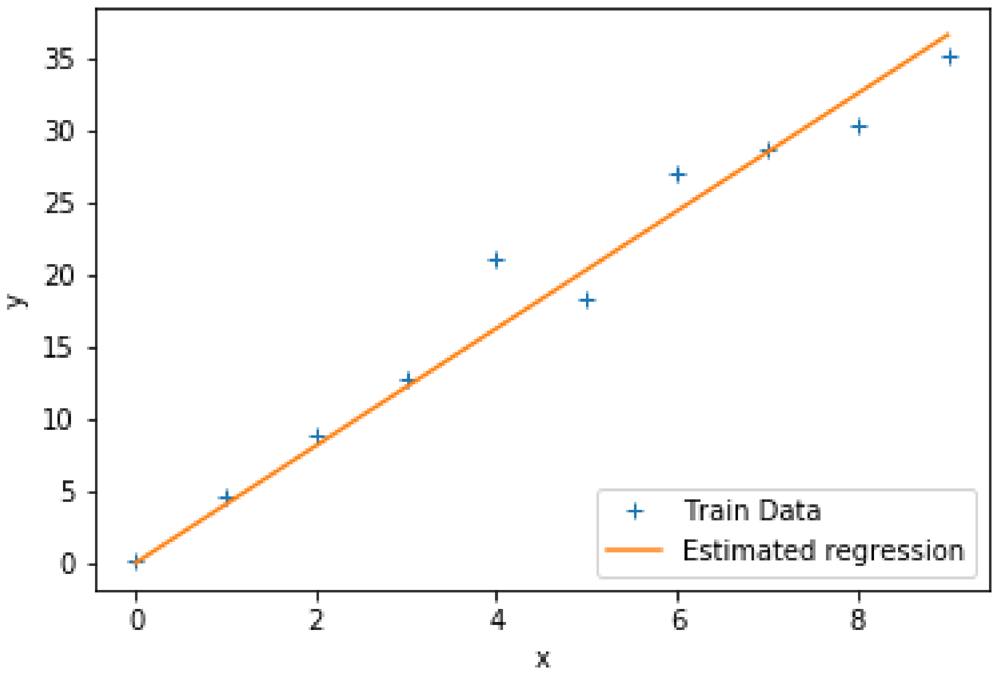
that optimally predicts the training labels from the training data, $X_{\text{train}} \in \mathbb{R}^{D_X \times N_{tr}}$, $Y_{\text{train}} \in \mathbb{R}^{D_Y \times N_{tr}}$. Here, $\lambda \in \mathbb{R}$ is the (optional) Ridge regularization parameter.
The function `ols_apply` than uses the weight vector to predict the (unknown) hand positions of new test data $X_{\text{test}} \in \mathbb{R}^{D_X \times N_{te}}$

$$Y_{\text{test}} = W^\top X_{\text{test}}$$

The function `test_assignment4` helps you to debug your code.

```
In [219]: test_assignment4()
```

```
No-noise-case tests passed
```



B) (1 point) The data set `myo_data.mat` consists of preprocessed EMG data X and 2-dimensional stimulus labels Y. Labels are x/y positions of the hand during different hand movements. The function `load_myo_data` loads the data and splits it into train and test data. Familiarize yourself with the data by answering the following questions:
How many time points N_{tr} does the train set contain? How many time points N_{te} does the test set contain? At each time point, at how many electrodes D_X was the EMG collected?

```
In [220]: X_train,Y_train,X_test, Y_test = load_myo_data("./myo_data.mat")
          print(X_train.shape[1])
          print(X_test.shape[1])
          print(X_test.shape[0])
```

```
5000
5255
192
```

N_{tr} = **[5000]**
N_{te} = **[5255]**
D_X = **[192]**

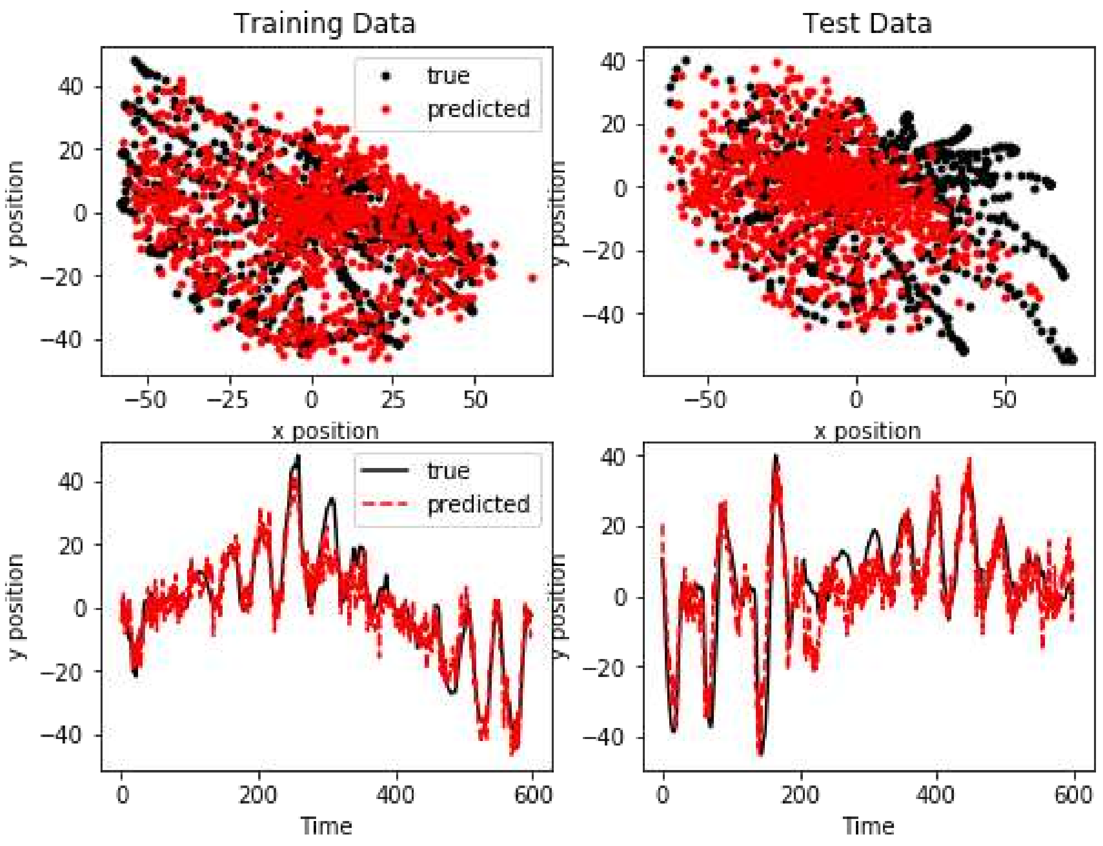
C) (1 points) Predict two dimensional hand positions by calling the function `predict_handpositions`. It plots, for the train and the test data, the true hand position versus the estimated hand position. Do you notice a performance difference between train and test data set? Is this a surprising result?

[Your answer for C) here]

5.1

The result of test data looks like a little bit far away from the truth, especially some details at the lower right corner. The possible reason for this difference is overfitting.

```
In [3]: predict_handposition()
```

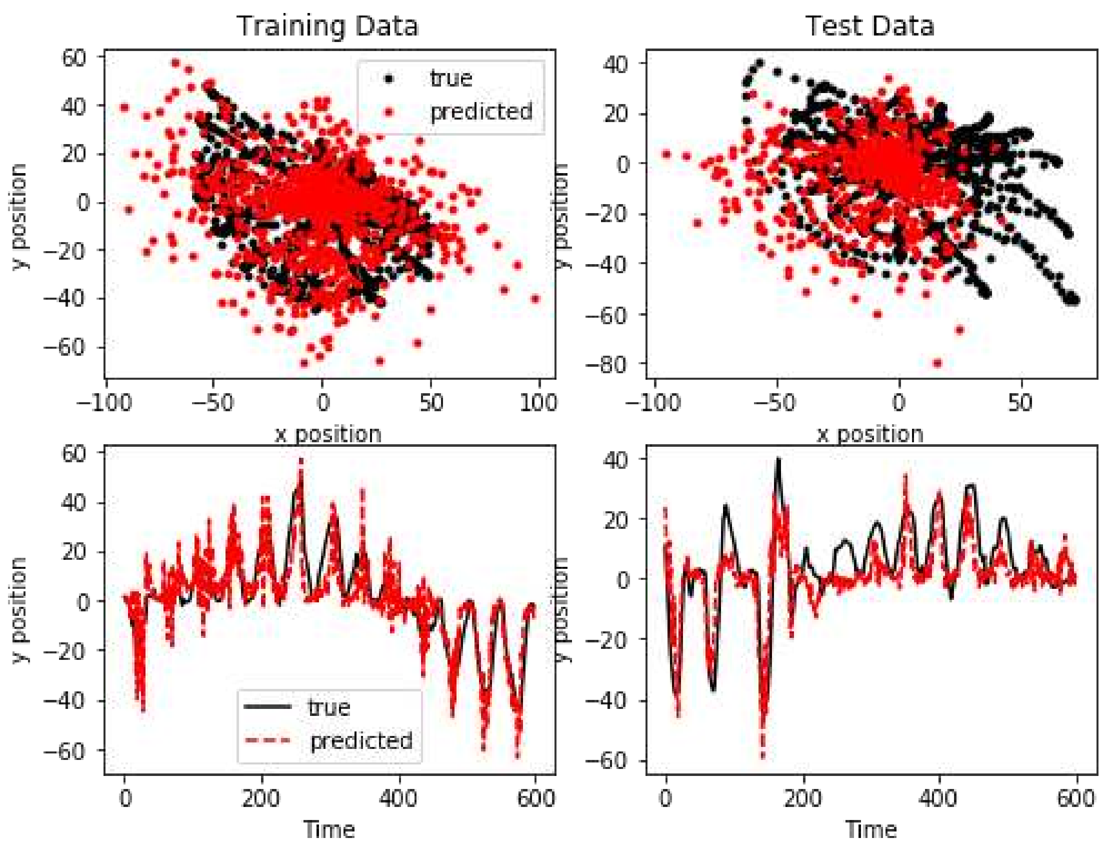


D) (1 points) In the previous tasks, we have used the logarithmized muscle activiations to predict the hand positions. Comment the line where we logarithmize the EMG features in the function `load_myo_data` and call `predict_handpositions` again. Do you notice a performance difference compared to the logarithmized version? Why?

[Your answer for D) here]

It seems that the true handpositions are overestimated, that is, all peaks in red line are higher than the true activations. This difference is introduced by the assumed relationship between hand position and muscle activation. Actually, hand position is a non-linear function of muscle activation, hence, before we apply linear regression, we have to transform muscle activations into a non-linear space, here is the logarithm space.

```
In [222]: predict_handposition()
```



E) (1 points) If we cannot predict the labels Y perfectly by a linear regression on X, does this imply that the relationship between X and Y is non-linear? Explain your decision.

[Your answer for E) here]

No. In general linear regression problem $Y = W^T X + \epsilon$, except the linear term of inputs X, there still exists noise which can also lead to not perfect prediction. It is just not to say the relationship between X and Y are non-linear.

2.1.2 Task 2: Polynomial Regression (18 points)

In task 1 you implemented linear regression. However, you will see in this task, that aboves code can be generalized to polynomial regression.

A) (9 points) Write a function `test_polynomial_regression` which generates toy data and visualizes the results from a polynomial regression. The goal is to create two plots as in the Figure below (Note that your figure will look slightly different, because the data is generated randomly.) To do so, first create toy data from a sine function as follows:

$$x_i \in \{0, 1, 2, \ldots, 10\}, y_i = \sin(x_i) + \epsilon_i, \quad \epsilon_i \sim \mathcal{N}(0, 0.5)$$

where $\mathcal{N}$(mean, standard deviation) denotes the Gaussian distribution and $i \in \{1, 2, \ldots, 11\}$ is an index. Then implement polynomial regression, which models the relationship between y and x as

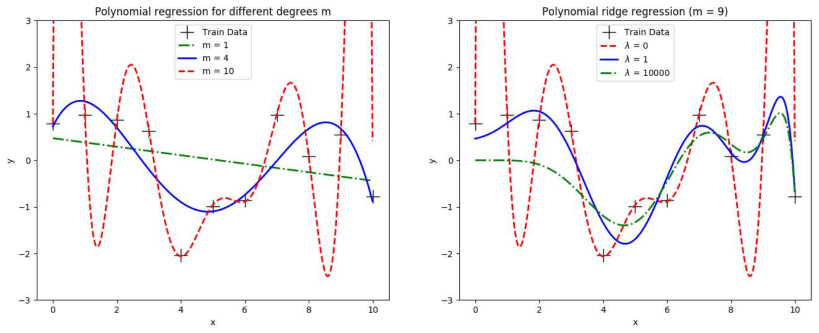


Figure_1

an mth order polynomial, i.e. $\hat{y} = w_0 + w_1 x + w_2 x^2 + \ldots + w_m x^m$. The parameters $w_0, w_1, \ldots, w_m \in \mathbb{R}$ are estimated by Ridge Regression.

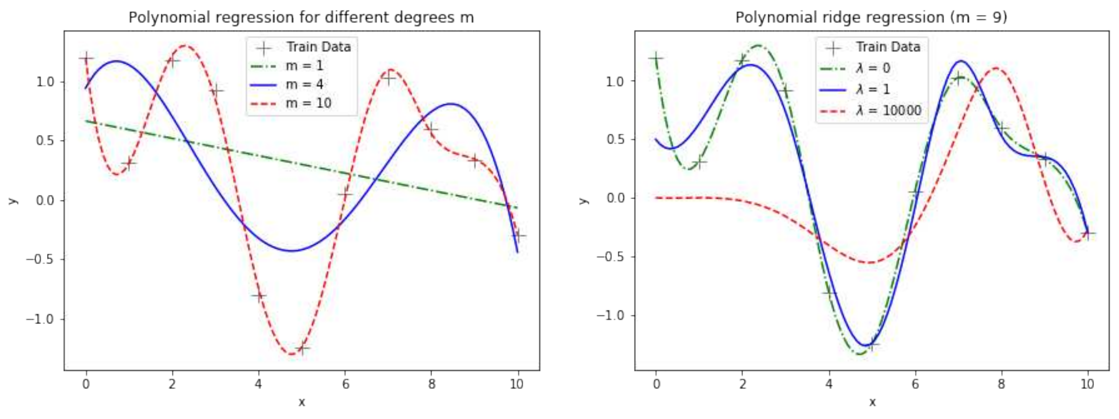
Hint: You can use your functions `ols_train` and `ols_apply`, if you build an appropriate data matrix (for loops are allowed to do so).

Apply and visualize polynomial ridge regression for different parameters.

```
In [4]: #your code here
        def polynomial_data(m):
            return np.power(np.arange(11)[:,np.newaxis],np.arange(m+1))

        def test_polynomial_regression():
            np.random.seed(20180530) #to make sure the result could be repeatly represented
            x = np.arange(11)
            toy = np.sin(x)+np.random.normal(0,0.5,11)
            X_1 = polynomial_data(1).T
            X_4 = polynomial_data(4).T
            X_10 = polynomial_data(10).T

            W_1 = train_ols(X_1,toy[np.newaxis,:],llambda = 0)
            Y_1 = apply_ols(W_1,X_1)

            W_4 = train_ols(X_4,toy[np.newaxis,:],llambda = 0)
            Y_4 = apply_ols(W_4,X_4)
            Y_4 = interp1d(x, Y_4, kind='cubic')

            W_10 = train_ols(X_10,toy[np.newaxis,:],llambda = 0)
            Y_10 = apply_ols(W_10,X_10)
            Y_10 = interp1d(x, Y_10, kind='cubic')

            f, axs = pl.subplots(1,2,figsize=(15,5))
            axs[0].plot(x, toy,'+',c='gray',markersize=12,label="Train Data")
            axs[0].plot(x, Y_1[0], '-.', c='g', linewidth=1.5, label="m = 1")

            x_new = np.linspace(0, 10, num=100, endpoint=True)
            axs[0].plot(x_new, Y_4(x_new)[0], c='b', linewidth=1.5, label="m = 4")
```

```
    axs[0].plot(x_new, Y_10(x_new)[0], '--', c='r', linewidth=1.5, label="m = 10")

    axs[0].set_xlabel('x')
    axs[0].set_ylabel('y')
    axs[0].set_title('Polynomial regression for different degrees m')
    axs[0].legend(loc=9)

    X_9 = polynomial_data(9).T
    W_0 = train_ols(X_9,toy[np.newaxis,:],llambda = 0)
    Y_0 = apply_ols(W_0,X_9)
    Y_0 = interp1d(x, Y_0, kind='cubic')

    W_1 = train_ols(X_9,toy[np.newaxis,:],llambda = 1)
    Y_1 = apply_ols(W_1,X_9)
    Y_1 = interp1d(x, Y_1, kind='cubic')

    W_10000 = train_ols(X_9,toy[np.newaxis,:],llambda = 10000)
    Y_10000 = apply_ols(W_10000,X_9)
    Y_10000 = interp1d(x, Y_10000, kind='cubic')

    axs[1].plot(x, toy,'+',c='gray',markersize=12,label="Train Data")
    x_new = np.linspace(0, 10, num=100, endpoint=True)
    axs[1].plot(x_new, Y_0(x_new)[0], '-.', c='g', linewidth=1.5, label=r'$\lambda$ = 0'
    axs[1].plot(x_new, Y_1(x_new)[0], c='b', linewidth=1.5, label=r'$\lambda$ = 1')
    axs[1].plot(x_new, Y_10000(x_new)[0], '--', c='r', linewidth=1.5, label=r'$\lambda$

    axs[1].set_xlabel('x')
    axs[1].set_ylabel('y')
    axs[1].set_title('Polynomial ridge regression (m = 9)')
    axs[1].legend(loc=9)
    pl.show()
```

```
In [5]: test_polynomial_regression()
```



B) (9 points) Run your code of task A) multiple times. - (4 points) What do you observe for the different values of the parameters m and λ? Explain this behaviour. - (2 points) Decide for each of the two figures which values of the parameters yield the best fit. - (3 points) Do you expect those parameters to perform good on all possible data sets? Explain your decision.

[Your answers for B) here]

For the different values of the parameter m, it describes trade-offs between bias and variance of w estimation. If m is small, it leads to a biased but low variance estimation. However, if the order of polynomial is higher, that is, m is too large, this introduces cooverfitting problem, even if the variance of esitmation becomes smaller.

For the parameter λ, when λ is zero, the result of ridge regression is the same with what ordinary linear squares produces, that is, there is no shrinkages on coefficents w. As λ increasing, the norm of coefficients vector w becomes smaller and smaller. This means that some components in vector w is reduced. By making use of this way, some higher order polynomial could be discarded, since their coefficients asymptotically approach zero, so that overfitting could be avoided. If λ is too la 10.1 most coefficients are nearby zero, the result is just a line.

Given three different values of m, m produces a better fit without loss generalisation. However, for different λ values, adding a little penality to the coefficients introduces some bias, but at the same time the variance is reduced.

No. How to choose a better set of parameters m and λ depends on the problem we want to solve. For instance, if relationships between given data are linear, $m = 1$ and $\lambda = 0$ are wise options.

Index of comments

5.1 P2, T1, C) Performance on test data is worse, especially in the region of the data, that was not part of the training. -1

10.1 P2, T2, B) Best fit for m=4, or m = 9 with lambda = 1. We do not want to fit each data point (overfit), but rather the sine function! -2