Machine Learning

TECHNISCHE UNIVERSITÄT DARMSTADT

Summer Semester 2017, Homework 3 (70 points + 10 bonus)
Prof. Dr. J. Peters, M. Ewerton, S. Parisi
Wang Yujue 2573991, Zhi Rong 2806891

Due Date: Wednesday, 28 June 2017 (before the lecture)

Problem 3.1 Linear Regression [31 Points + 5 Bonus]

a) Polynomial Features [10 Points]

$$y = \boldsymbol{\phi}(\boldsymbol{x})^T \boldsymbol{\theta} \tag{1}$$

$$\boldsymbol{\phi}(\boldsymbol{x}) = \left(1, x, x^2, \ldots, x^k\right)^T \tag{2}$$

$$\boldsymbol{\Phi} = \boldsymbol{\Phi}(\boldsymbol{X}) = \begin{bmatrix} \boldsymbol{\phi}(x_1)^T \\ \vdots \\ \boldsymbol{\phi}(x_n)^T \end{bmatrix} \tag{3}$$

$$\boldsymbol{\theta} = \left(\boldsymbol{\Phi}^T \boldsymbol{\Phi} + \lambda \boldsymbol{I}\right) \boldsymbol{\Phi}^T \boldsymbol{y} \tag{4}$$

When the model complexity is 12th degree polynomial, I achieve the best RMSE. Yes it will change if we evaluate our model on the training data, it will be smaller and not changing very much after 12th degree polynomial. It is because that we train the model according to this training set, if we again only evaluate our result on this training set, it does no make many sense.

And note that, originally up to 21st degree polynomial gives result in Figure 2 when the polynomial is very high order. Python gives numerical error when computing inverting. This leads to unreasonable high RMSE. So instead I plot the RMSE up to 17th degree polynomial, which is more reasonable.

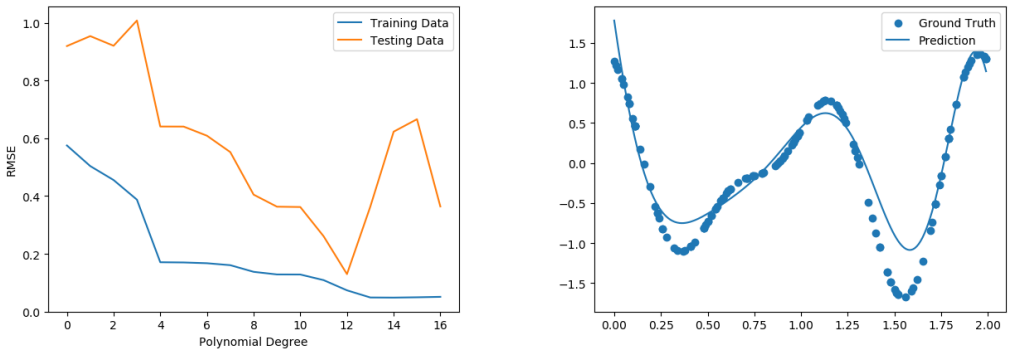

Figure 1: RMSE and model prediction over the ground truth

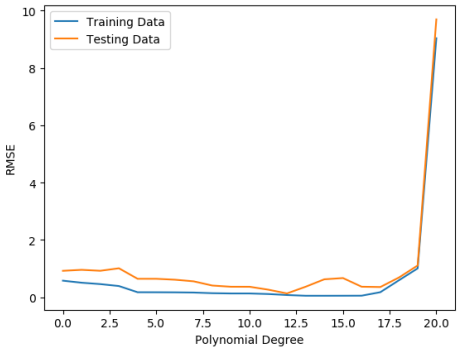

Figure 2: RMSE

b) Gaussian Features [4 Points]

Note that, different Normalization method leads to slightly different style of this Gaussian feature plot. Here I use three different feature scaling methods, which are Rescaling, Standardization and Scaling to Unit Length respectively.

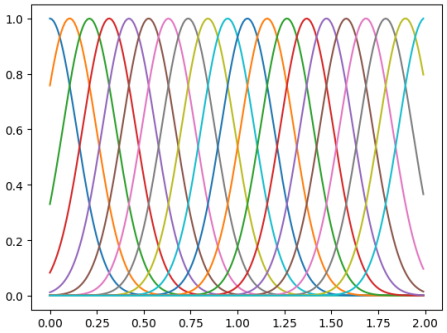

Figure 3: Normalization by Rescaling

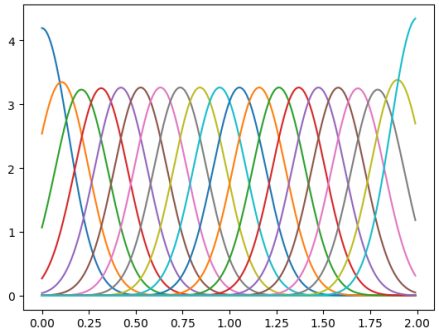

Figure 4: Normalization by Standardization

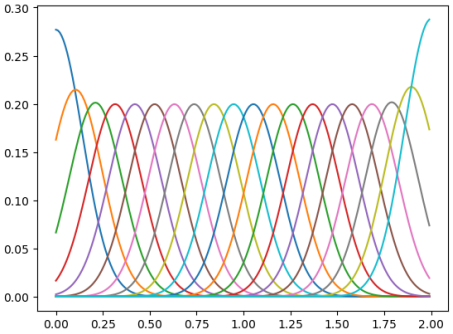

Figure 5: Normalization by Scaling to Unit Length

c) Gaussian Features, Continued [4 Points]

40 basis functions have the best performance and the RMSE is 0.0165.

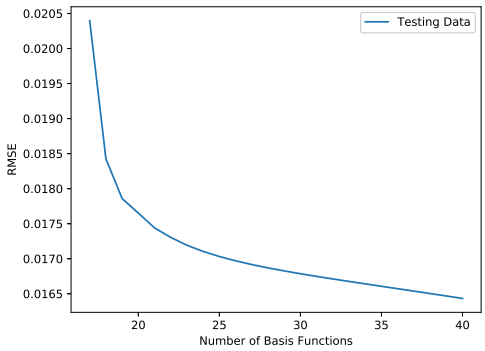

Figure 6: RMSE versus basis functions

d) Bayesian Linear Regression [10 Points]

Using Bayesian linear regression, the mean and the standard deviation of the predictive distribution learned using the first [10, 12, 16, 20, 50, 150] data points are shown in Figure 7 and Figure 8

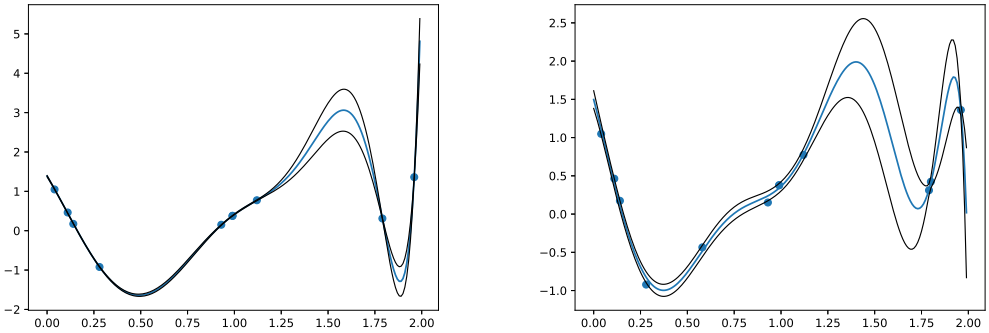

Figure 7: Mean and standard deviation of the predictive distribution

e) Bayesian Linear Regression, Continued [3 Points]

Try to get more datasets, it is quite straight forward. The more training data we have the less uncertainty the prediction. However, it is normally not easy to get large number of data and to label them is also very expensive. Remove the outliers of datasets can also reduce uncertainty but it is hard to decide previously which should be considered as outliers especially when we only have a small number of datasets. And use higher polynomial can increase the prediction but can also end up in over fitting.

Name, Vorname: _____ Matrikelnummer: ⌴⌴⌴⌴⌴⌴⌴

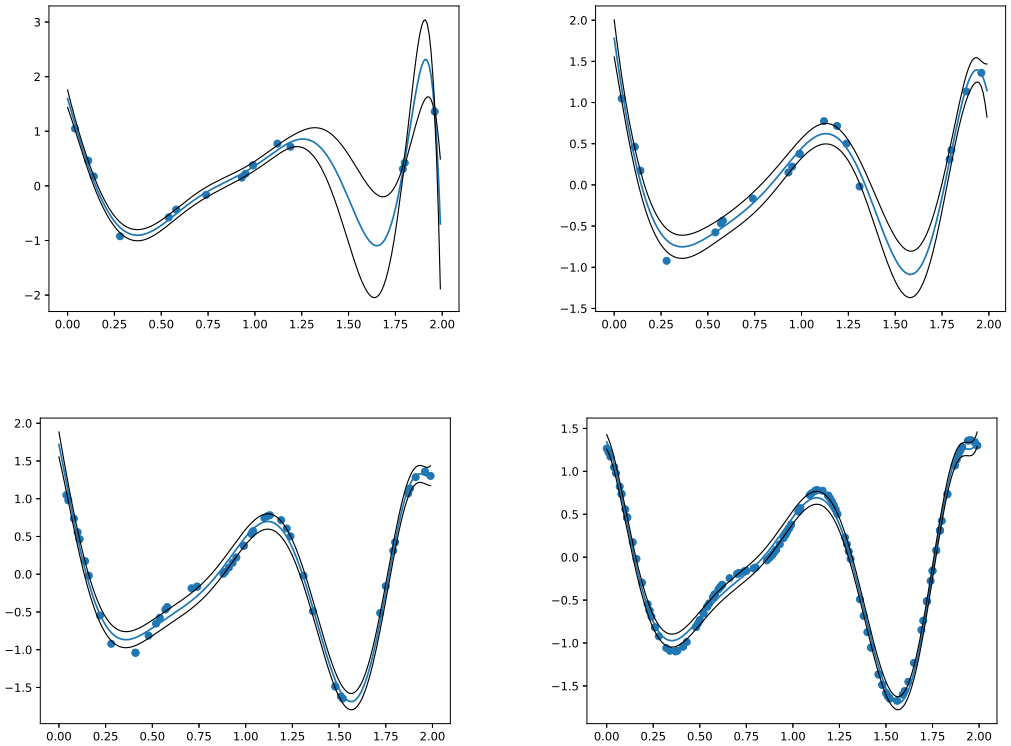

Figure 8: Mean and standard deviation of the predictive distribution

f) Cross Validation [5 Bonus Points]

We use the test set error to evaluate how well does the model generalize. However, the problem of test set error is that it is likely to be an optimistic estimate of generalization error. I.e. our extra parameter (degree of polynomial) is fit to test set. So we need to introduce cross validation set to decide this extra parameter. So that to prevent overfitting. Normally, we can separate our dataset into 60% training set 20% cross validation set 20% test set.

Problem 3.2 Linear Classification [16 Points]

a) Discriminative and Generative Models [4 Points]

Discriminative models, as opposed to generative models, do not allow one to generate samples from the joint distribution of x and y. However, for tasks such as classification and regression that do not require the joint distribution, discriminative models can yield superior performance. On the other hand, generative models are typically more flexible than discriminative models in expressing dependencies in complex learning tasks. In addition, most discriminative models are inherently supervised and cannot easily be extended to unsupervised learning. Application specific details ultimately dictate the suitability of selecting a discriminative versus generative model.

Examples of discriminative models:
Linear regression
Logistic regression
Support vector machines
Neural networks

Generative models contrast with discriminative models, in that a generative model is a full probabilistic model of all variables, whereas a discriminative model provides a model only for the target variable(s) conditional on the observed variables.

Examples of generative models:
Gaussian Mixture model

b) Linear Discriminant Analysis [12 Points]

29 samples are misclassified.

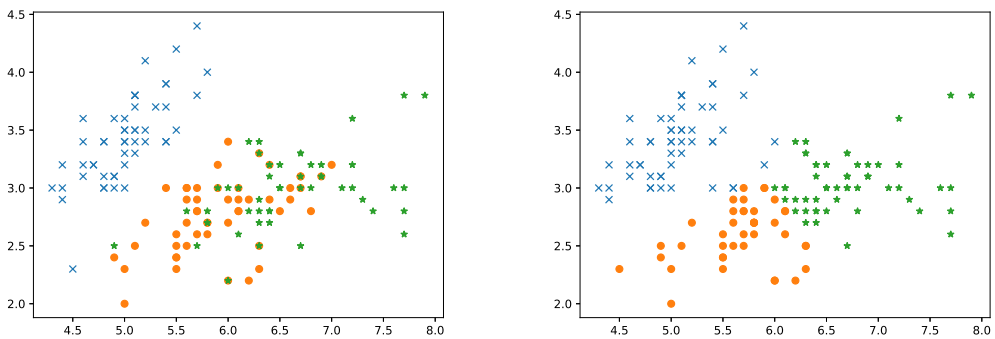

Figure 9: Ground Truth and Samples Classified after LDA Classifier

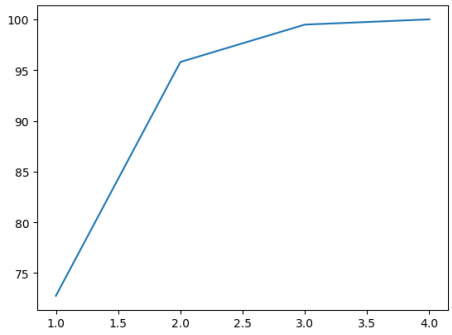

Figure 10: Percentage of the Cumulative Variance Explained

Problem 3.3 Principal Component Analysis [23 Points + 5 Bonus]

In this exercise, you will use the dataset `iris.txt`. It contains data from three kind of Iris flowers ('Setosa', 'Versicolour' and 'Virginica') with 4 attributes: sepal length, sepal width, petal length, and petal width. Each row contains a sample while the last attribute is the label (0 means that the sample comes from a 'Setosa' plant, 1 from a 'Versicolour' and 2 from 'Virginica'). (You are allowed to use built-in functions for computing the mean, the covariance, eigenvalues and eigenvectors.)

a) Data Normalization [3 Points]

Normalizing the data is a common practice in machine learning. Normalize the provided dataset such that it has zero mean and unit variance per dimension. Why is normalizing important? Attach a snippet of your code.

Normalization generally avoid duplicate and redundant data. Which leads to faster data processing.

For example, When features differ by orders of magnitude, first performing feature scaling can make gradient descent converge much more quickly.

b) Principal Component Analysis [8 Points]

Apply PCA on your normalized dataset and generate a plot showing the proportion (percentage) of the cumulative variance explained. How many components do you need in order to explain at least 95% of the dataset variance? Attach a snippet of your code.

From the plot in Figure 10, we find that 2 components needed to explain at least 95% of the dataset variance.

c) Low Dimensional Space [6 Points]

Using as many components as needed to explain 95% of the dataset variance, generate a scatter plot of the lower-dimensional projection of the data. Use different colors or symbols for data points from different classes. What do you observe? Attach a snippet of your code.

Figure 11 is the data projection of the principal components analysis.

Two principal components are enough to distinguish the three different classes. PCA has successfully separated out different classes.

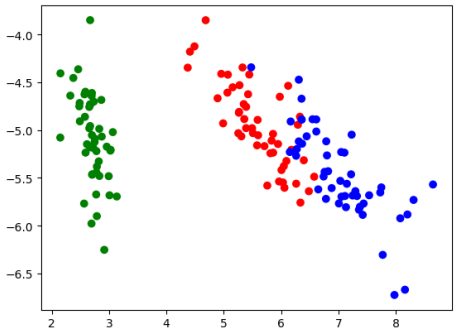

Figure 11: Lower Dimensional Projection of the Data

d) Projection to the Original Space [6 Points]

Reconstruct the original dataset by using different number of principal components. Using the normalized root mean square error (NRMSE) as a metric, fill the table below (error per input versus the amount of principal components used).

N. of components	x_1	x_2	x_3	x_4
1	0.1040	0.1609	0.0384	0.0831
2	0.0640	0.0170	0.0379	0.0807
3	0.0086	0.0032	0.0343	0.0238
4	0.0000	0.0000	0.0000	0.0000

Attach a snippet of your code. (Remember that in the first step you normalized the data.)

e) Kernel PCA [5 Bonus Points]

Throughout this class we have seen that PCA is an easy and efficient way to reduce the dimensionality of some data. However, it is able to detect only linear dependences among data points. A more sophisticated extension to PCA, Kernel PCA, is able to overcome this limitation. This question asks you to deepen this topic by conducting some research by yourself: explain what Kernel PCA is, how it works and what are its main limitations. Be as concise (but clear) as possible.

The general task of kernel methods is to find and study general types of relations (for example principal components) in datasets. Kernel PCA enable them to operate in a high-dimensional, implicit feature space without ever computing the coordinates of the data in that space. Using a kernel, the originally linear operations of PCA are performed in a reproducing kernel Hilbert space. It finds the eigenvectors of the covariance of a kernel projection of the dataset. Advantage: it is able to capture non-linear correlation between input variables. Disadvantage: it can be much more computationally expensive.

Machine Learning - Homework 3

Name, Vorname: _____ Matrikelnummer: ␣␣␣␣␣␣␣␣

Code:

Problem 3.1

```
import numpy as np
import matplotlib.pyplot as plt

# a)
def get_polynomial_feature_matrix(points, num_polynomials):
    num_points = len(points)
    feature_matrix = np.zeros((num_points,num_polynomials))
    for i in range(num_points):
        for j in range(num_polynomials):
            feature_matrix[i,j] = points[i] ** j
    return feature_matrix

def ridge_linear_regression(data, feature_matrix, ridge_coeff):
    labels = data[:,1]
    X = feature_matrix
    num_features = np.shape(X)[1]
    preInv = (X.T).dot(X) + np.eye(num_features) * ridge_coeff
    theta = np.linalg.inv(preInv).dot(X.T).dot(labels)
    return theta

def calc_rmse(true, prediction):
    num_points = np.shape(true)[0]
    err = true-prediction
    rmse = np.sqrt((err.T).dot(err)/num_points)
    return rmse

def predict_polynomial(points, theta):
    num_polynomials = np.size(theta)
    X = get_polynomial_feature_matrix(points, num_polynomials)
    prediction = X.dot(theta)
    return prediction

data_all = np.loadtxt('dataSets/linRegData.txt')
data_train = data_all[:20,:]
data_eval = data_all[20:,:]
points_train = data_train[:,0]
points_eval = data_eval[:,0]
true_train = data_train[:,1]
true_eval = data_eval[:,1]

max_polynomial = 21
ridge_coef = 10**(-6)

train_err = []
eval_err = []
for num_features in range(1,max_polynomial+1):
    feature_matrix = get_polynomial_feature_matrix(points_train, num_features)
    theta = ridge_linear_regression(data_train, feature_matrix, ridge_coef)
    predictions_train = predict_polynomial(points_train,theta)
```

```
    predictions_eval = predict_polynomial(points_eval,theta)
    train_err.append(calc_rmse(true_train, predictions_train))
    eval_err.append(calc_rmse(true_eval, predictions_eval))

num_poly = np.argmin(eval_err)
print(num_poly)
best_feature_matrix = get_polynomial_feature_matrix(points_train, np.argmin(eval_err))
theta_best = ridge_linear_regression(data_train,best_feature_matrix,ridge_coef)
model_input = np.arange(0,2,0.01)
best_model = predict_polynomial(model_input,theta_best)

plt.figure()
h_all_data = plt.scatter(data_all[:,0],data_all[:,1],label="Ground␣Truth")
h_model, = plt.plot(model_input,best_model,label="Prediction")
plt.legend(handles = [h_all_data, h_model])
plt.show()

plt.figure()
h_train_err, = plt.plot(train_err, label="Training␣Data")
h_eval_err, = plt.plot(eval_err, label="Testing␣Data")
plt.legend(handles = [h_train_err,h_eval_err])
plt.xlabel("Polynomial␣Degree")
plt.ylabel("RMSE")
plt.show()

# b)
def eval_gaus(x, mu, sig2):
    exp = np.exp(-(x - mu)**2 / (2*sig2))
    return exp

def get_gaussian_feature_matrix(points, num_centers):
    sig2 = 0.02
    dist = 2.0/(num_centers-1)
    mu = np.arange(0,2.001,dist)
    feature_matrix = np.zeros( (len(points),len(mu)) )
    for i in range(len(mu)):
        feature_matrix[:,i] = eval_gaus(points, mu[i], sig2)
    return feature_matrix

def predict_rbf(points, theta):
    num_centers = np.size(theta)
    X = get_gaussian_feature_matrix(points, num_centers)
    prediction = X.dot(theta)
    return prediction

numfeature = 20
feature_matrix = get_gaussian_feature_matrix(points_train,numfeature)
theta = ridge_linear_regression(data_train,feature_matrix,ridge_coef)
features = feature_matrix = get_gaussian_feature_matrix(model_input,numfeature)

# feature normalization and plot
plt.figure()
```

```
# three different normalization method leads slightly different style of this gaussian feature
for i in range(numfeature):
    # Rescaling
    features[:,i] = (features[:,i] - np.min(features[:,i])) / (np.max(features[:,i]) - np.min(f
    plt.plot(model_input, features[:, i])
plt.figure()
for i in range(numfeature):
    # Scaling to unit length
    features[:, i] = features[:, i] / np.linalg.norm(features[:, i])
    plt.plot(model_input, features[:, i])
plt.figure()
for i in range(numfeature):
    # Standardization
    features[:, i] = (features[:, i] - np.min(features[:, i])) / np.std(features[:, i])
    plt.plot(model_input, features[:, i])

plt.show()

# c)
train_err = []
eval_err = []
min_num_centers = 17
max_num_centers = 40
for num_cent in range(min_num_centers,max_num_centers+1):
    feature_matrix = get_gaussian_feature_matrix(points_train, num_cent)
    theta = ridge_linear_regression(data_train, feature_matrix, ridge_coef)
    predictions_train = predict_rbf(points_train,theta)
    predictions_eval = predict_rbf(points_eval,theta)
    train_err.append(calc_rmse(true_train, predictions_train))
    eval_err.append(calc_rmse(true_eval, predictions_eval))

plt.figure()
h_eval_err, = plt.plot(np.arange(min_num_centers,max_num_centers+1),eval_err, label="Testing␣Da
plt.legend(handles = [h_eval_err])
plt.xlabel("Number␣of␣Basis␣Functions")
plt.ylabel("RMSE")
plt.show()

# d)
pol_rank = 12
num_train_samples = [10, 12, 16, 20, 50, 150]
model = predict_rbf(model_input, theta)

for i in range(len(num_train_samples)):
    # calculate model predictive mean
    data_train = data_all[:num_train_samples[i],:]
    points_train = data_train[:,0]
    true_train = data_train[:,1]
    feature_matrix = get_polynomial_feature_matrix(points_train, pol_rank)
    theta = ridge_linear_regression(data_train, feature_matrix, ridge_coef)
    predictions_train = predict_polynomial(points_train,theta)
```

```
    centered = true_train - predictions_train
    sig2model = (centered.T).dot(centered)/len(true_train)
    A = feature_matrix
    lambdaI = np.eye(np.shape(A)[1]) * ridge_coef
    inv = np.linalg.inv( (A.T).dot(A) + lambdaI )
    sig = []
    model_input = np.arange(0,2,0.01)
    for j in range(len(model_input)):
        x = get_polynomial_feature_matrix([model_input[j]],pol_rank).T
        sig.append( np.sqrt( (sig2model + sig2model * (x.T).dot(inv).dot(x))[0,0] ) )

    plt.figure()
    h_train_data = plt.scatter(data_train[:, 0], data_train[:, 1], label="train data")
    h_model, = plt.plot(model_input,model,label='Mean')
    h_std, = plt.plot(model_input,model+sig,color='black',linewidth=1.0, label = "Standard D
    plt.plot(model_input,model-sig,color='black',linewidth=1.0)
    plt.show()
```

Problem 3.2

```
import numpy as np
import matplotlib.pyplot as plt

# load and separate data
data_all = np.loadtxt('dataSets/ldaData.txt')
data_C1 = data_all[:50,:]
data_C2 = data_all[50:100,:]
data_C3 = data_all[100:,:]

ones1 = np.ones((np.shape(data_C1)[0],1))
ones2 = np.ones((np.shape(data_C2)[0],1))
ones3 = np.ones((np.shape(data_C3)[0],1))
zeros1 = np.zeros((np.shape(data_C1)[0],1))
zeros2 = np.zeros((np.shape(data_C2)[0],1))
zeros3 = np.zeros((np.shape(data_C3)[0],1))
points_C1_1 = np.concatenate( [ones1,data_C1], 1 )
points_C2_1 = np.concatenate( [ones2,data_C2], 1 )
points_C3_1 = np.concatenate( [ones3,data_C3], 1 )
points_all_1 = np.concatenate( [points_C1_1,points_C2_1,points_C3_1], 0 )
labels_C1 = np.concatenate( [ones1,zeros1,zeros1], 1 )
labels_C2 = np.concatenate( [zeros2,ones2,zeros2], 1 )
labels_C3 = np.concatenate( [zeros3,zeros3,ones3], 1 )
labels_all = np.concatenate( [labels_C1,labels_C2,labels_C3], 0 )

# using least squares
X = points_all_1
T = labels_all
W = np.linalg.inv(X.T.dot(X)).dot(X.T).dot(T)

Predictions = np.zeros(np.shape(labels_all))
for i in range(np.shape(Predictions)[0]):
    Predictions[i,:] = W.T.dot(X[i,:].T)
predictions = np.argmax(Predictions,1)+1
```

```
points_pred_C1 = data_all[predictions == 1,:]
points_pred_C2 = data_all[predictions == 2,:]
points_pred_C3 = data_all[predictions == 3,:]

truth = np.argmax(labels_all,1)+1
Error = np.zeros(np.shape(truth))
Error[truth != predictions] = 1
num_misclassified = np.sum(Error)
print(num_misclassified)

plt.figure()
h_C1 = plt.plot(points_C1_1[:,1],points_C1_1[:,2], 'x')
h_C2 = plt.plot(points_C2_1[:,1],points_C2_1[:,2], 'o')
h_C3 = plt.plot(points_C3_1[:,1],points_C3_1[:,2], '*')
plt.show()

plt.figure()
h_C1p = plt.plot(points_pred_C1[:,0],points_pred_C1[:,1], 'x')
h_C2p = plt.plot(points_pred_C2[:,0],points_pred_C2[:,1], 'o')
h_C3p = plt.plot(points_pred_C3[:,0],points_pred_C3[:,1], '*')
plt.show()
```

Problem 3.3

```
import numpy as np
import matplotlib.pyplot as plt

data_dim = 5

# read data and normalize
def read_dataset(path):
    data = []
    txt = open(path)
    for line in txt:
        a,b,c,d,e = map(float,line.strip().split(","))
        data.append((a,b,c,d,e))
    return np.asarray(data)

def normalize(data):
    return (data - data.mean(0)) / np.std(data,0)

# calculate eigenvalues & eigenvectors and visualize them
def eigendecomposition(train):
    vals,vecs = np.linalg.eig(np.cov(train.T))
    sort_idx = np.argsort(vals)[::-1]
    vals_sort = vals[sort_idx]
    vecs_sort = vecs[:,sort_idx]
    return vals_sort,vecs_sort

def propotion(vals):
    cum_var = np.cumsum(vals)
    var_pro = (cum_var / cum_var[-1]) * 100
    return var_pro
```

```
# low dimensional representation
def representation(train,vecs,dim):
    cord = np.zeros((len(train),dim))
    for i,x in enumerate(train):
        cord[i,:] = vecs[:,:dim].T.dot(x)
    return cord


data = read_dataset('dataSets/iris.txt')
train = data[:,:4]

label = data[:,-1]
N = len(label)

norm_train = normalize(train)
vals,vecs = eigendecomposition(norm_train)

var_pro = propotion(vals)
plt.plot(range(1,5),var_pro)
plt.show()

x_cord = representation(train,vecs,dim=2)
color = ['r','g','b']
plt.scatter(x_cord[:,0],x_cord[:,1],c=[color[int(i)] for i in label])
plt.show()

# projection to original space
for i in range(1,5):
    x_cordn = representation(train,vecs,dim = i)
    x_proj= np.zeros((N,4))
    for n, x in enumerate(x_cordn):
        x_proj[n,:] = vecs[:,:i].dot(x)
    error = [np.sqrt(sum((x_proj[:,m] - train[:,m])**2)/N) for m in range(4)]
    print(error)
```