

PROJECT 1: PROBABILITY DISTRIBUTIONS AND BAYESIAN NETWORKS

Avinash Kommineni, 50248877

October 23, 2017

Introduction

The csv file is read and loaded into the workspace by the *read_csv* query of *pandas* library as a dataframe. Since there is no header and the values start right from row 0, the argument *header=None* is used. The given data is divided into training set, validation set and test set of sizes 80%, 10%, 10% respectively and stored into their respective X's and y's.

Listing 1: Calculating Mean Variance Standard deviation.

```
1 import random
2 import numpy as np
3 import pandas as pd
4 from scipy.stats import multivariate_normal
5 import matplotlib.pyplot as plt
6 from sklearn import preprocessing, cluster, model_selection, ←
   metrics
7
8 synInputData = pd.read_csv('input.csv',header=None).values
9 synOutputData = pd.read_csv('output.csv',header=None).values
10 letorInputData = pd.read_csv('Querylevelnorm_X.csv',header=None).←
   values
11 letorOutputData = pd.read_csv('Querylevelnorm_t.csv',header=None).←
   values
12
13 # X_train, X_test, y_train, y_test = model_selection.←
   train_test_split(synInputData, synOutputData, test_size=0.20, ←
   shuffle=False)
14 # X_validate, X_test, y_validate, y_test = model_selection.←
   train_test_split(X_test, y_test, test_size=0.50, shuffle=False)
15
16 X_train, X_test, y_train, y_test = model_selection.train_test_split←
```

```

    (letorInputData, letorOutputData, test_size=0.20, shuffle=False)
17 X_validate, X_test, y_validate, y_test = model_selection.↵
    train_test_split(X_test, y_test, test_size=0.50, shuffle=False)

```

Design Matrix

The design matrix, Φ shown below is calculated from

$$\phi_j(x) = \exp\left(-\frac{1}{2}(x - \mu_i)^T \Sigma_j^{-1}(x - \mu_i)\right)$$

and substituted accordingly in the below form. Use of vector methods help code this pretty easily.

$$\Phi = \begin{bmatrix} \phi_0(x_1) & \phi_1(x_1) & \phi_2(x_1) & \dots & \phi_{M-1}(x_1) \\ \phi_0(x_2) & \phi_1(x_2) & \phi_2(x_2) & \dots & \phi_{M-1}(x_2) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \phi_0(x_N) & \phi_1(x_N) & \phi_2(x_N) & \dots & \phi_{M-1}(x_N) \end{bmatrix}$$

Listing 2: Calculating covarianceMat and correlationMat matrices.

```

1 def compute_design_matrix(X_train, k_clusters):
2     N,D = X_train.shape
3     kmeans = cluster.KMeans(k_clusters).fit(X_train)
4     centers = kmeans.cluster_centers_
5     centers = centers[:,np.newaxis,:]
6     spreads = []
7     covM = np.cov(X_train.T)
8     for _ in range(0,k_clusters): spreads.append(covM)
9     spreads = np.array(spreads)
10    X = X_train[np.newaxis,:,:]
11
12    basis_func_outputs = np.exp(np.sum(np.matmul(X - centers, ↵
        spreads) * (X - centers), axis=2) / (-2) ).T
13    return np.insert(basis_func_outputs, 0, 1, axis=1)

```

Closed Form Solution

The closed form solution with the regularisation is calculated from

$$w_{ML} = (\lambda I + \Phi^T \Phi)^{-1} \Phi^T t$$

The equation has been implemented in a efficient way by the use of broadcasting.

Listing 3: Calculating logLikelihood value.

```
1 def closed_form_solution(l2_lamda, designMatrix, outValues):
2     weights = np.dot(np.dot(np.linalg.inv(np.dot(designMatrix.T,
3         designMatrix)), designMatrix.T), outValues)
4     return weights
```

As shown, it takes the design matrix, truth values and the regularisation factor λ .

Stochastic Gradient Descent

Listing 4: Calculating logLikelihood for multivariate gaussian.

```
1 def SGD(learningRate, l2_lamda, designMatrix, outValues, ↵
2     miniBatchSize, numEpochs, k_clusters):
3     N=designMatrix.shape[0]
4     weights = np.zeros([1, k_clusters+1])
5     for epoch in range(numEpochs):
6         for i in range(int(N / miniBatchSize)):
7             lower_bound = i * miniBatchSize
8             upper_bound = min((i+1)*miniBatchSize, N)
9             Phi = designMatrix[lower_bound : upper_bound, :]
10            t = outValues[lower_bound : upper_bound, :]
11            E_D = np.matmul((np.matmul(Phi, weights.T)-t).T, Phi )
12            E = (E_D + l2_lamda * weights) / miniBatchSize
13            weights = weights - learningRate * E
14            return weights.flatten()
```

Error - RMS

The RMS error has been calculated from the given formula,

$$E_{RMS} = \sqrt{2E(w^*)/N_v}$$

where $E(w)$ is defined as ...

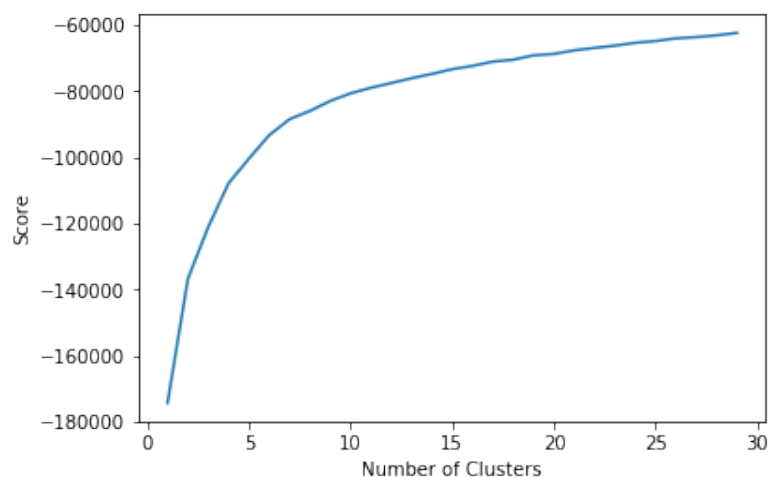
$$E(w) = E_D(w) + \frac{1}{2}\lambda E_W(w)$$

Listing 5: Calculating logLikelihood for multivariate gaussian.

```
1 def rms_error(weights, designMatrix, Y, l2_lamda):
2     y_calc = np.dot(designMatrix, weights)
3     E_D_W = 0.5*metrics.mean_squared_error(Y, y_calc)*designMatrix.shape[0]
4     E_W_W = 0.5*np.dot(weights.T, weights)
5     E_W = E_D_W + l2_lamda*E_W_W
6     train_error = np.sqrt(2*E_W/designMatrix.shape[0])
7     return train_error
```

Results 1

The hyper-parameter M needs to be set. I calculated it by plotting a graph of number of cluster vs scores.

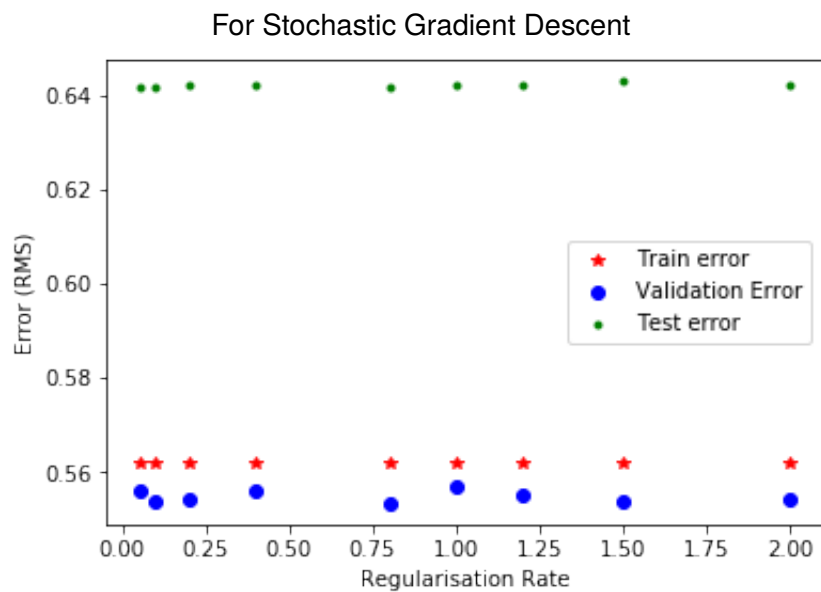
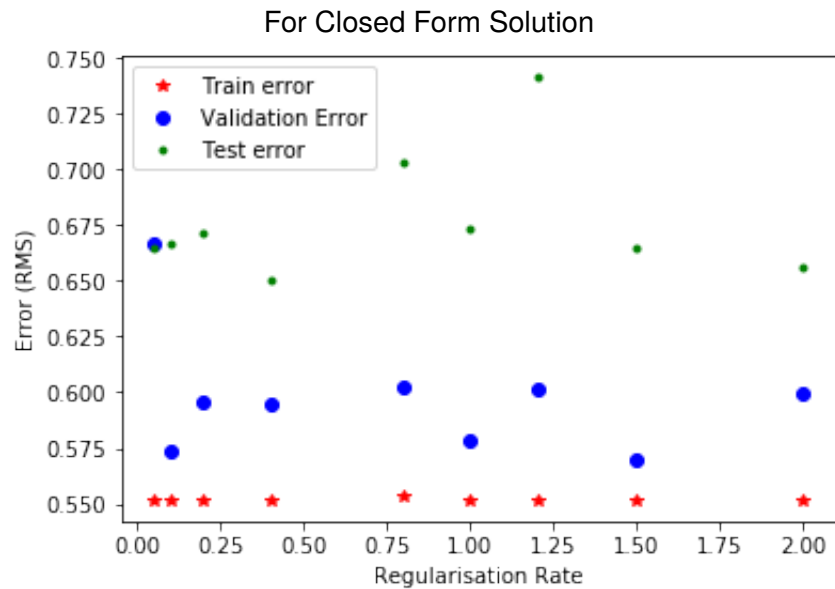


From the above graph, I took a safe value of 10 as the value of M.

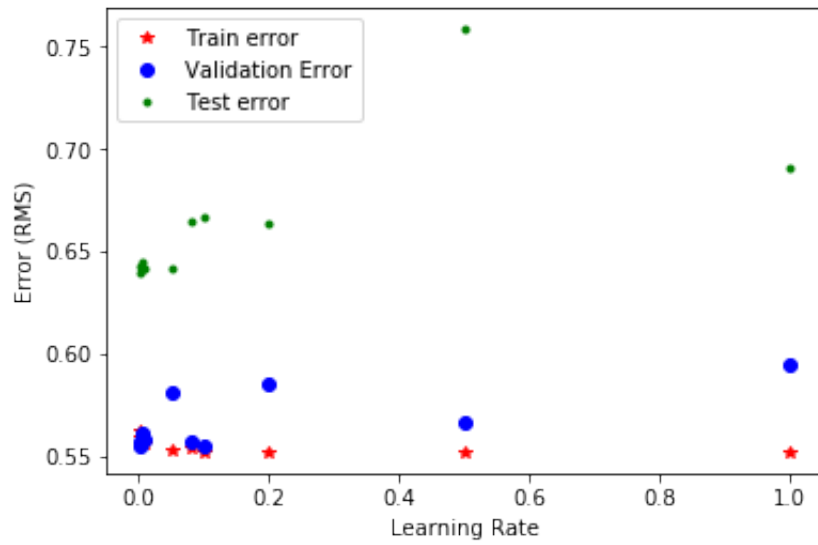
```
1 k_cluster = 10
2
3 l2_lamda = 0.9
4
5 designMatrix = compute_design_matrix(X_train,k_cluster)
6 weights_train0 = closed_form_solution(l2_lamda, designMatrix, ↵
    y_train)
7 train_error0 = rms_error(weights_train0,designMatrix,y_train,↵
    l2_lamda)
8 print("Training error:",train_error0)
9
10 designMatrix2 = compute_design_matrix(X_validate,k_cluster)
11 train_error2 = rms_error(weights_train0,designMatrix2,y_validate,↵
    l2_lamda)
12 print("Test error (Validation set):",train_error2)
13
14
15 #Test error
16 designMatrix3 = compute_design_matrix(X_test,k_cluster)
17 train_error3 = rms_error(weights_train0,designMatrix3,y_test,↵
    l2_lamda)
18 print("Test error (test set):",train_error3)
```

Results 2

The following plots are drawn to better understand and decide the variation between the 3 errors and hyper-parameters such as regularization λ and learning rate η .

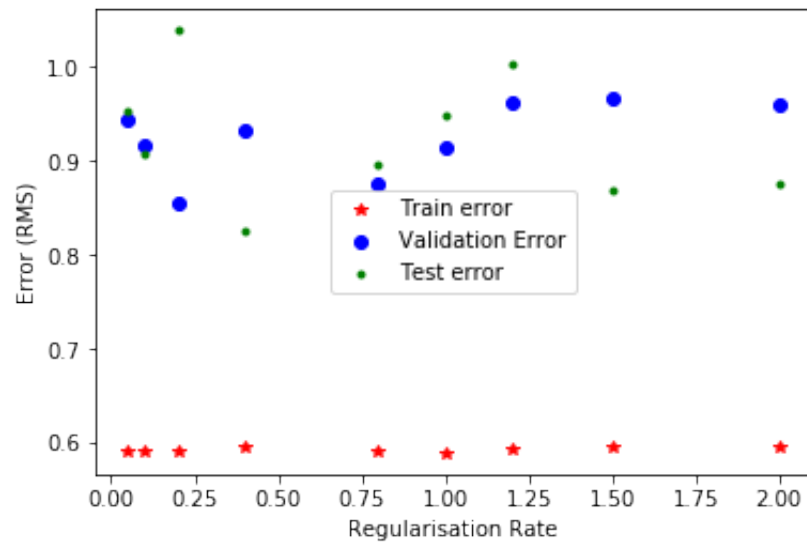


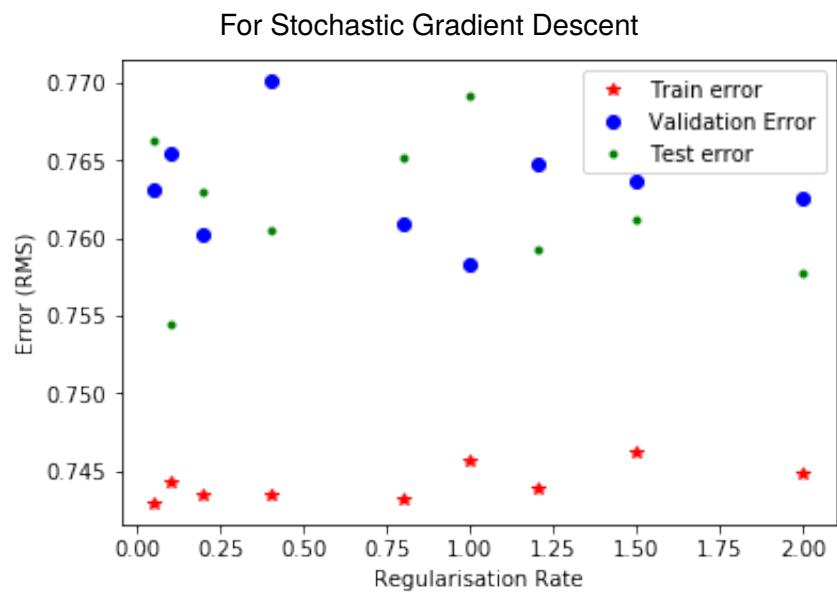
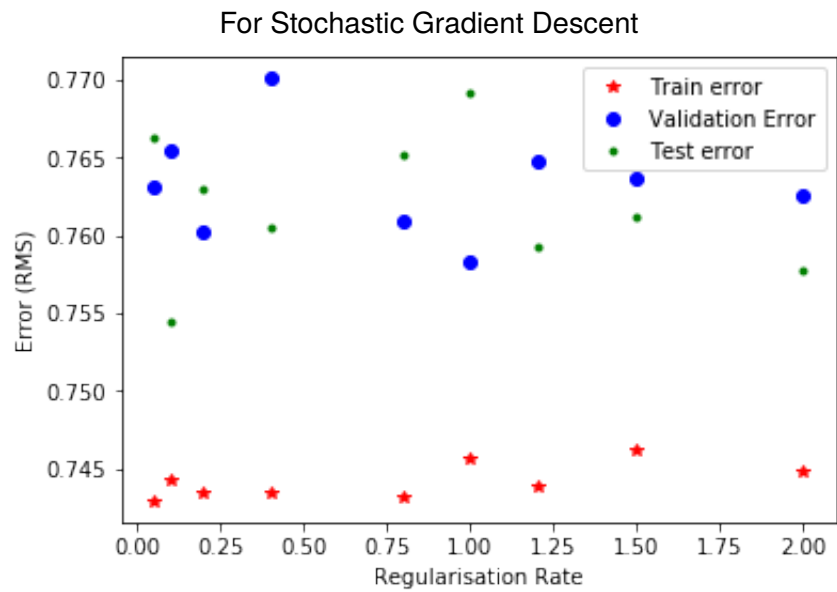
For Stochastic Gradient Descent



The below graphs are for the synthetic dataset.

For Closed Form Solution





- The stochastic gradient descent appears to have a better error rate than closed form solution even though test error is high when compared to train and validation set error but has a little consistency.