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_csvl* 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.

```
import random
import numpy as np
import pandas as pd
from scipy.stats import multivariate_normal
import matplotlib.pyplot as plt
from sklearn import preprocessing, cluster, model_selection, metrics

synInputData = pd.read_csv('input.csv',header=None).values
synOutputData = pd.read_csv('output.csv',header=None).values
letorInputData = pd.read_csv('Querylevelnorm_X.csv',header=None).values
letorOutputData = pd.read_csv('Querylevelnorm_t.csv',header=None).values

# X_train, X_test, y_train, y_test = model_selection.train_test_split(synInputData, synOutputData, test_size=0.20, shuffle=False)
# X_validate, X_test, y_validate, y_test = model_selection.train_test_split(X_test, y_test, test_size=0.50, shuffle=False)

X_train, X_test, y_train, y_test = model_selection.train_test_split(letorInputData, letorOutputData, test_size=0.20, shuffle=False)
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 matrx, Φ shown belowis calculated from

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

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

$$\Phi = \begin{bmatrix} \phi_0(\mathrm{x}_1) & \phi_1(\mathrm{x}_1) & \phi_2(\mathrm{x}_1) & \dots & \phi_{M-1}(\mathrm{x}_1) \\ \phi_0(\mathrm{x}_2) & \phi_1(\mathrm{x}_2) & \phi_2(\mathrm{x}_2) & \dots & \phi_{M-1}(\mathrm{x}_2) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \phi_0(\mathrm{x}_N) & \phi_1(\mathrm{x}_N) & \phi_2(\mathrm{x}_N) & \dots & \phi_{M-1}(\mathrm{x}_N) \end{bmatrix}$$

```
def compute_design_matrix(X_train, k_clusters):
    N,D = X_train.shape
    kmeans = cluster.KMeans(k_clusters).fit(X_train)
    centers = kmeans.cluster_centers_
    centers = centers[:,np.newaxis,:]
    spreads = []
    covM = np.cov(X_train.T)
    for _ in range(0,k_clusters): spreads.append(covM)
    spreads = np.array(spreads)
    X = X_train[np.newaxis,:,:]

    basis_func_outputs = np.exp(np. sum(np.matmul(X - centers, ↩
        spreads) * (X - centers), axis=2) / (-2) ).T
    return np.insert(basis_func_outputs, 0, 1, axis=1)
```

Closed Form Solution

The closed form solution with the regularisation is calculated from

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

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

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

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

Stochastic Gradient Descent

```
def SGD(learningRate, l2_lamda, designMatrix, outValues, miniBatchSize, numEpochs,k_clusters):
    N=designMatrix.shape[0]
    weights = np.zeros([1, k_clusters+1])
    for epoch in range(numEpochs):
    for i in range(int(N / miniBatchSize)):
    lower_bound = i * miniBatchSize
    upper_bound = min((i+1)*miniBatchSize, N)
    Phi = designMatrix[lower_bound : upper_bound, :]
    t = outValues[lower_bound : upper_bound, :]
    E_D = np.matmul((np.matmul(Phi, weights.T)-t).T,Phi )
    E = (E_D + l2_lamda * weights) / miniBatchSize
    weights = weights - learningRate * E
    return weights.flatten()
```

Error - RMS

The RMS error has been calculated from the given formula,

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

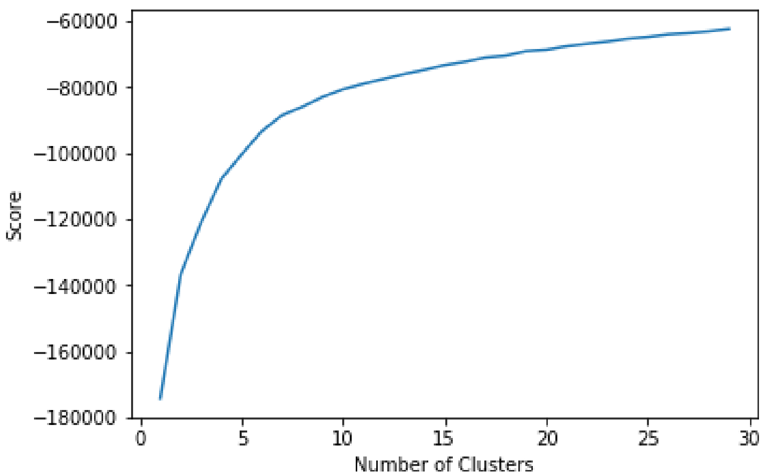
where $E(w)$ is defined as ...

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

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

```python
k_cluster = 10

l2_lamda = 0.9
```

```

designMatrix = compute_design_matrix(X_train,k_cluster)
weights_train0 = closed_form_solution(l2_lamda, designMatrix, y_train)
train_error0 = rms_error(weights_train0,designMatrix,y_train,l2_lamda)
print("Training error:",train_error0)

designMatrix2 = compute_design_matrix(X_validate,k_cluster)
train_error2 = rms_error(weights_train0,designMatrix2,y_validate,l2_lamda)
print("Test error (Validation set):",train_error2)


#Test error
designMatrix3 = compute_design_matrix(X_test,k_cluster)
train_error3 = rms_error(weights_train0,designMatrix3,y_test,l2_lamda)
print("Test error (test set):",train_error3)
```

Code Output

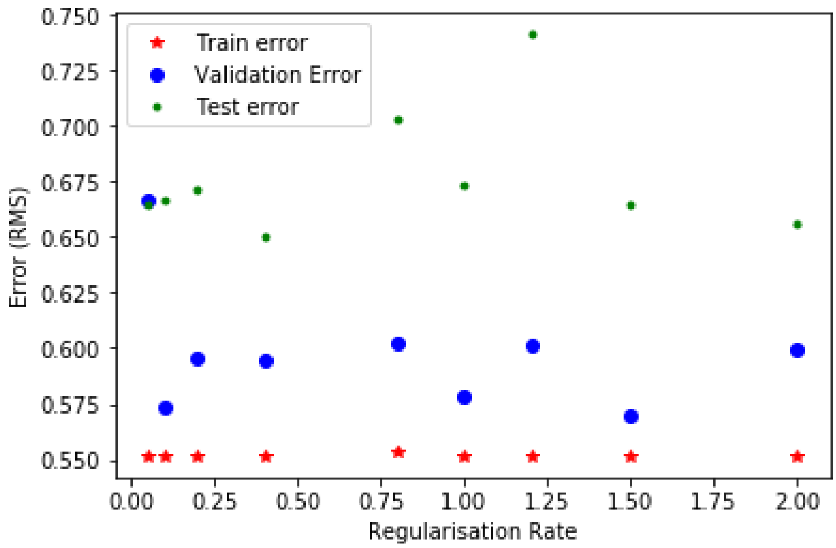
Listing 1: Code output.

```
From closed form solution.
Training error: [[ 0.54948601]]
Test error (Validation set): [[ 0.64268395]]
Test error (test set): [[ 0.68269302]]
From SGD.
Training error: 0.560374755594
Test error (Validation set): 0.556667218125
Test error (test set): 0.639586215758
```

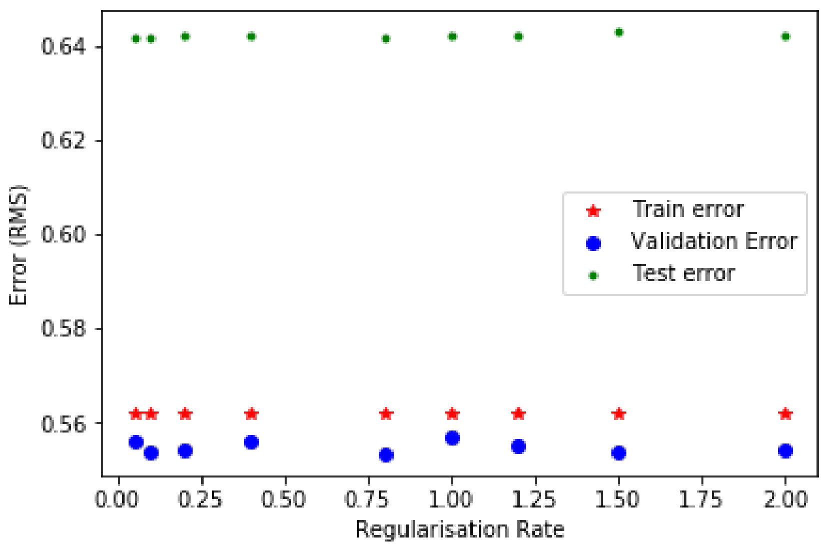
Results 2

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.

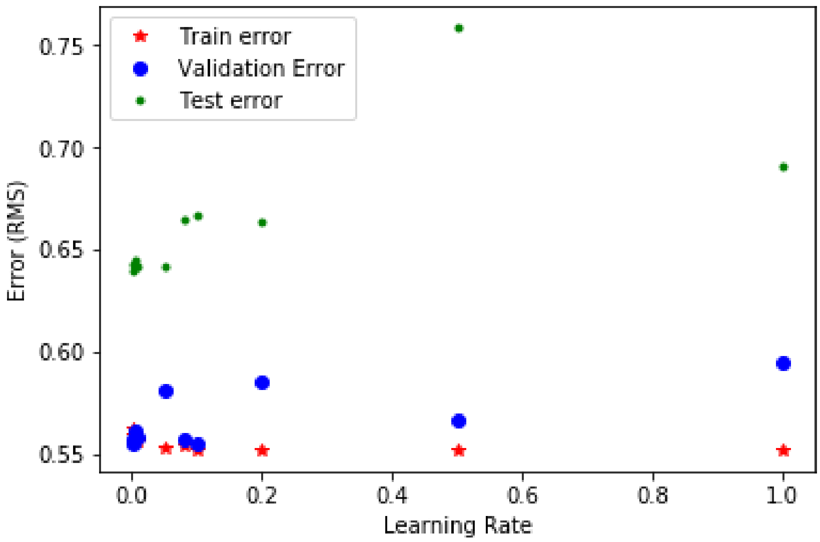
For Closed Form Solution



For Stochastic Gradient Descent

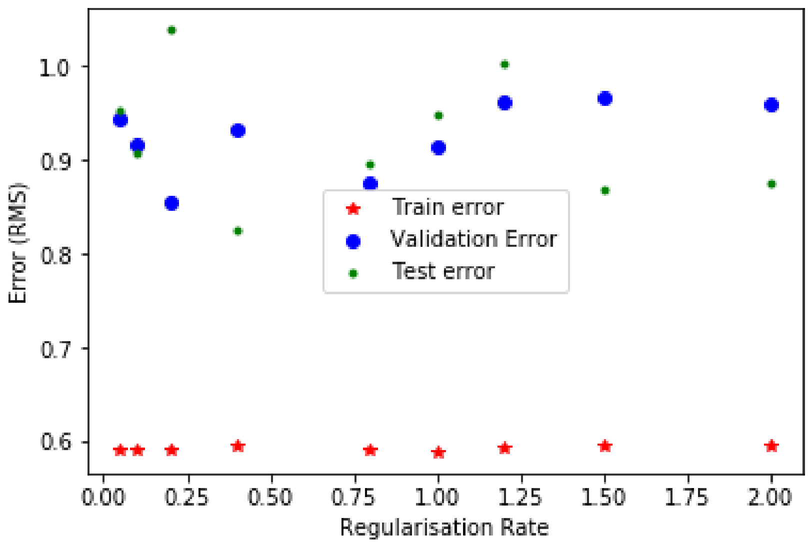


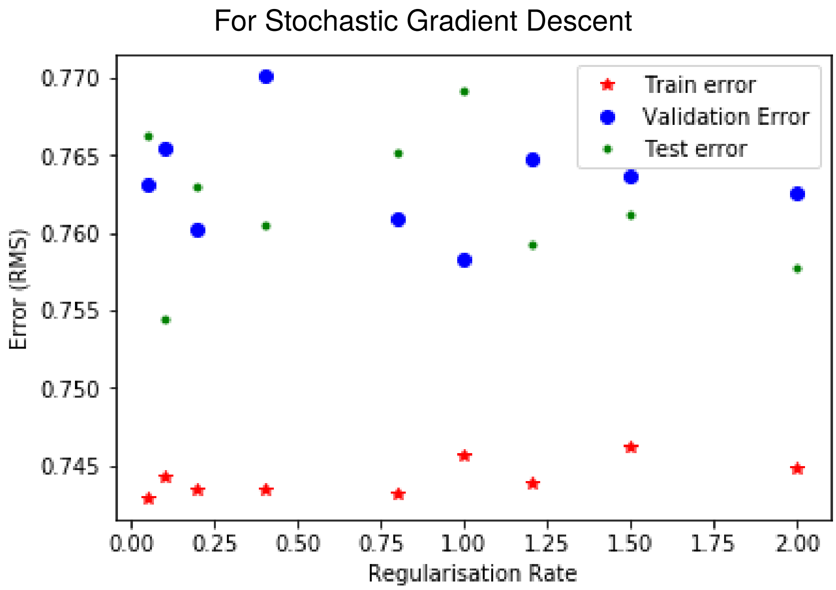
For Stochastic Gradient Descent

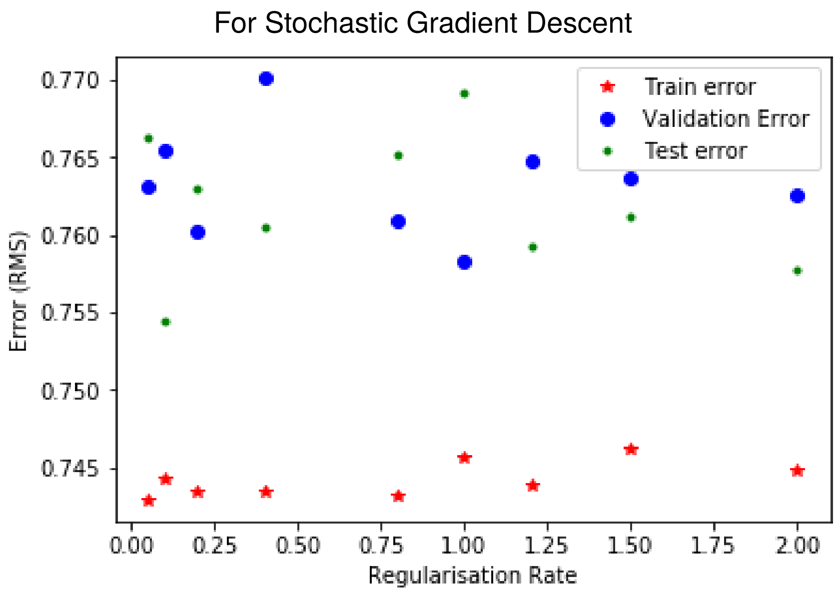


The below graphs are for the synthetic dataset.

For Closed Form Solution







- The stochastic graient 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.
- The learning time increases significantly with the increase in M, the number of basis functions.