Assignment$_8$

Chaitra , Ganesh , Avik

July 5 2019

1 Problem 1

a) Let A be a symmetric matrix in R^{mxm}. Show that for all $x \in R^m$. Show that, if $f(x) = x^T Ax$, then: $\frac{\delta f}{\delta x} = 2Ax$
Proof:
Given that A is a symmetric matrix, it has a Lower Upper decomposition (LU). Thus we have, $A = LU \longrightarrow ①$
The product rule is defined as $\nabla xy = x(\nabla y) + y(\nabla x) \longrightarrow ②$
Furthermore we have, $\nabla Ax = A(\nabla x^T A = \nabla (A^T x)^T = (\nabla A^T x)^T = (A^T)^T = A) \longrightarrow ③$
With the definitions of ①, ② and ③,

$$\frac{\delta f}{\delta x} = x^T Ax$$
$$= x^T LUx (from ①)$$
$$= x^T L(\nabla Ux) + (\nabla x^T L)Ux (from ②)$$
$$= x^T LU + LUx$$
$$= x^T A + Ax$$
$$= x^T A^T + Ax (from ③)$$
$$= Ax + Ax$$

Therefore,

$$\frac{\delta f}{\delta x} = 2Ax$$

b) Consider the kernel ridge regression optmization problem (Lecture 8, Slide 39). Let $\alpha^* \in R^d$ the vector that minimizes the loss function. Show that: $\alpha^* = (K + \frac{1}{2C} I_n)^{-1} y$
Proof:
In order to get the optimal solution, one must set the gradient to zero. Using the representer theorem, we can write: (Lecture 8, Slide 39)

$$min_{\alpha \in R^n} \frac{1}{2} \alpha^T K\alpha + C||y - K\alpha||^2 \longrightarrow ①$$

We know that by the norm rule that, $\alpha\alpha^T = ||\alpha||^2 \longrightarrow ②$
Derive ① with respect to α,

$$= \frac{1}{2}||\alpha||^2 K + C||y - K\alpha||^2$$

$$= \frac{1}{2}.2||\alpha||K + C2||y - K\alpha||(-K)$$

$$= K\alpha - 2CK(y - K\alpha)$$

Now Divide by 2C,

$$= \frac{K\alpha}{2C} - K(y - K\alpha)$$

Set the gradient to zero,

$$= \frac{K\alpha}{2C} - Ky + K^2\alpha = 0$$

Take the common term $K\alpha$ out, you also get an Identity matrix $1 \longleftrightarrow I$,

$$0 = K\alpha(\frac{1}{2C}I_n + K) - Ky$$

Rearrange,

$$\alpha = \frac{Ky}{(\frac{1}{2C}I_n + K)K}$$

Therefore,

$$\alpha^* = (K + \frac{1}{2C}I_n)^{-1}y$$

2 Problem 2

We proofed the following theorem (Lecture 8, Slide 99).
$w_{RR} = (XX^T + \frac{1}{2C}I)^{-1}Xy$ The traditional linear regression has as solution $w_R = (XX^T)^{-1}Xy$. The matrix $X \in R^{nxd}$ is commonly not invertible. For example, if our problem has more features than entries the traditional linear regression is not defined since (XX^T) is singular. Ridge regression solve this problem adding $\frac{1}{2C}I$. For which values of C, $(XX^T + \frac{1}{2C}I)$ is singular and therefore would have no solution? (Tip: consider the eingenvalues of XX^T)
Proof:
We know that $|A - \lambda I| = 0 \longrightarrow ①$ where λ is the eigenvalue of the matrix.
Given that, $w_{RR} = (XX^T + \frac{1}{2C}I)^{-1}Xy \longrightarrow ②$ Now for $(XX^T + \frac{1}{2C}I)$ to be singular its determinant must be zero.
By representing $\longrightarrow ②$ in $\longrightarrow ①$ we get,

$$-\lambda I = \frac{1}{2C}I$$

$$C = -\frac{1}{2\lambda}$$

Therefore, for any values of λ other than zero, we get C values for which $(XX^T + \frac{1}{2C}I)$ is singular and thus will have no solution.

3 Problem 3

In this exercise we will code the LOOCV trick for the Ridge Regression. Our aim is to predict the a metro interstate traffic volume given features about the time and weather conditions. You can download the dataset from UC Irvine machine learning repository1 . ATTENTION: In this dataset you have categorical variables. You need to preprocess those variables (it was discussed in the last exercise session).
a) Devide your dataset in a training set (95 %) / test set (5 %) . Use the training set to fit your model finding the best value for C. Use the LOOCV trick.

```
# coding: utf-8

# In[1]:


import pandas as pd
import numpy as np
import datetime
import os
import math
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt


# In[2]:


def getData(filePathWithName):

    return pd.read_csv(filePathWithName)


# In[3]:


def getPreprocessedData(df):

    columnNameUniqueValueCountMap = {}

    for element in df.dtypes.iteritems():

        if(element[0] != 'date_time'):
            if element[1].name == 'object':
```

```
                columnNameUniqueValueCountMap[element[0]] = df[element[0]].uniqu
            else:
                columnNameUniqueValueCountMap[element[0]] = 0
        else:
            timeList = []
            for index in range(0,24):
                timeList.append(index)
            columnNameUniqueValueCountMap['time'] = timeList

    preprocessedDataList = []

    columnNameList = []

    for key,value in columnNameUniqueValueCountMap.items():

        if(isinstance(value,np.ndarray)):

            for columnName in value:
                columnNameList.append(key+'_'+columnName)
        elif isinstance(value,list):
            for columnName in value:
                columnNameList.append(key+'_'+str(columnName))
        else:
            columnNameList.append(key)

    for row in df.iterrows():
        listData = []
        for item in row[1].iteritems():

            if item[0] in columnNameUniqueValueCountMap
and isinstance(columnNameUniqueValueCountMap[item[0]],np.ndarray)
and len(columnNameUniqueValueCountMap[item[0]]) > 0:

                uniqueValues = columnNameUniqueValueCountMap[item[0]]
                for value in uniqueValues:

                    if value == item[1]:
                        listData.append(1)
                    else:
                        listData.append(0)
            elif item[0] in columnNameUniqueValueCountMap
and not isinstance(columnNameUniqueValueCountMap[item[0]],np.ndarray):

                listData.append(item[1])
            else:
                uniqueValues = columnNameUniqueValueCountMap['time']
```

```
                for value in uniqueValues:
                    dateTimeObj = datetime.datetime.strptime(item[1], '%Y-%m-%d 
                    if value == dateTimeObj.hour:
                        listData.append(1)
                    else:
                        listData.append(0)

        preprocessedDataList.append(listData)

    preprocessedDataFrame = pd.DataFrame(preprocessedDataList)
    preprocessedDataFrame.columns = columnNameList
    return preprocessedDataFrame


# In[4]:


def performRegression(dataFrame):

    c = [0.1, 0.01, 0.05, 0.02, 1]
    bestWRR = []
    RMSE = []
    trainFrame = dataFrame.iloc[:,17:90]

    responseFrame = dataFrame.iloc[:,-1]
    trainFrame = trainFrame.iloc[:,:-1]
    xxT = np.dot(trainFrame.transpose(), trainFrame)
    for i in c:
        wRR = 0.0
        cIdentity = (1/(2*i))*np.identity(trainFrame.shape[1])
        inverse = np.linalg.inv(xxT + cIdentity)
        xY = (np.dot(trainFrame.transpose(),responseFrame))
        wRR = np.dot(inverse,xY)
        bestWRR.append(wRR)
        A = xxT + cIdentity
        AInv = np.linalg.inv(A)
        sumRmse = 0.0
        for valIndex in range(len(dataFrame)):
            numerator = (np.dot(dataFrame.iloc[valIndex,17:89],wRR)) - dataFrame
            denominator = (1-(np.dot(np.dot(dataFrame.iloc[valIndex,17:89],AInv)
            sumRmse += math.pow((numerator/denominator),2)
        RMSE.append(math.sqrt(sumRmse/len(dataFrame)))
    print("RMSE",RMSE)
    return bestWRR
```

```
# In[5]:


def predict(wRR, testSet):

    return np.dot(testSet.iloc[:,17:89],wRR)


# In[6]:


if os.path.exists('./data/preprocessedData.csv'):
    preprocessedData = getData("./data/preprocessedData.csv")
else:
    df = getData("./data/Metro_Interstate_Traffic_Volume.csv")
    preprocessedData = getPreprocessedData(df)
    preprocessedData.to_csv('./data/preprocessedData.csv', sep=',')


# In[7]:


preprocessedData.head()


# In[8]:


trainingSet, testSet = train_test_split(preprocessedData, test_size=0.05, shuffl
wRR = performRegression(trainingSet)


# In[9]:


predictionVector = predict(wRR[0], testSet)
print(predictionVector)
d1 = (predictionVector - testSet.iloc[:,-1])
d2 = np.power(d1,2)
plt.boxplot(d1)
plt.show()
plt.boxplot(d2)
plt.show()
print(predictionVector)
```

Best Value of C: 0.01

```
RMSE [947.0064084140643, 954.4462545830494, 947.4365968981006, 949.6713294019498, 947.0137122978796]
```

Figure 1: RMSE values

b) The most popular method to check the quality of a regression algorithm is the deviation between the predicted value and actual value . Make two boxplot and compare.

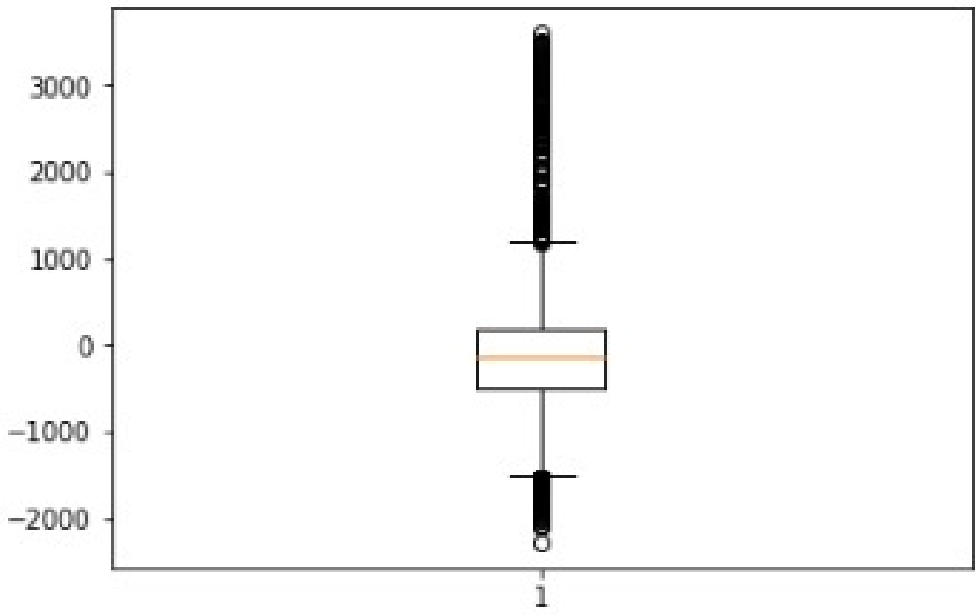

Figure 2: Boxplot d1

c) Perform an analysis of the boxplots of item (b). Write a little text describing what you understood.
Answer: The boxplot of d1 and d2 reveals a lot of outliers in the distribution. Mean line is centered towards zero in case of d1 but is left skewed in case of d2. The distribution is more spread towards the positive deviation from the mean than towards the left side.

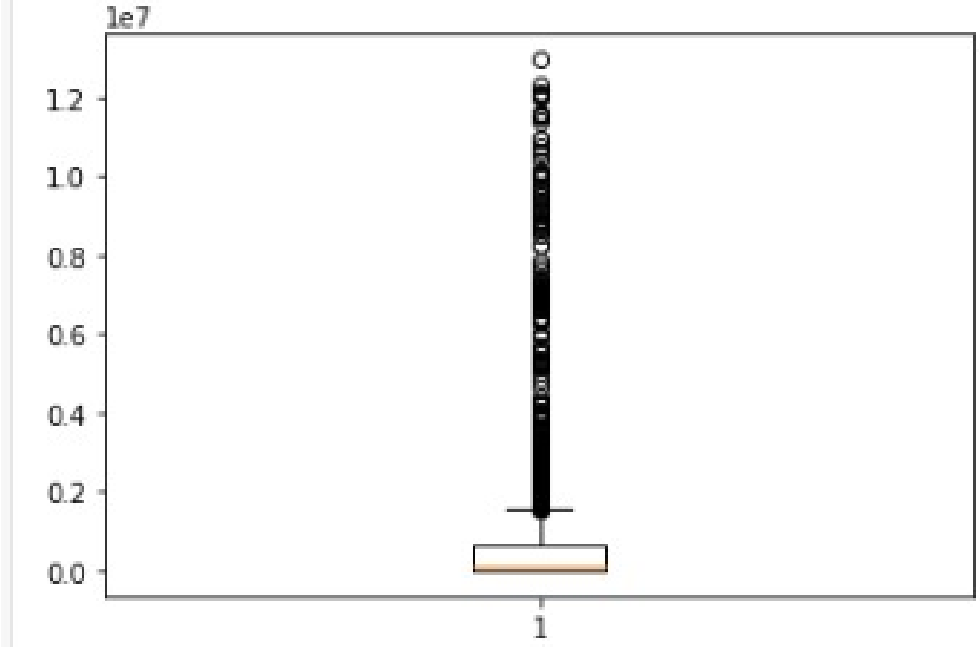

Figure 3: Boxplot d2