CS 598: Practical Statistical Learning

Homework 1

**Netid: pushpit2**

Question 1 [50 Points] KNN

a) Write a function myknn(xtest, xtrain, ytrain, k) that fits a KNN model that predict a target point or multiple target points xtest. Here xtrain is the training dataset covariate value, ytrain is the training data outcome, and k is the number of nearest neighbors. Use the ℓ2 norm to evaluate the distance between two points. Please note that you cannot use any additional R package within this function.

Here I have implemented the KNN algorithm from scratch. The method **myknn** can be called from the outside with training & test data and value for k. It then internally uses l2NormAvgOutput method to find the k nearest neighbors for each of the data instance in test set and then return the mean of the Y values for those k nearest neighbors (When KNNs are used for classification, the mode of the Y values of K nearest neighbors are taken, but here we are using KNN for regression, hence taking mean of the Y values seems most accurate to me in order to do prediction in continuous space).

1. **def** l2NormAvgOutput(xTestInstance, xtrain, ytrain, k):
2. **if** k == 0:
3. **return** 0
4. dist = (xtrain - xTestInstance)\*\*2
5. dist = np.sum(dist, axis=1)
6. dist = np.sqrt(dist)
7. **return** np.mean(ytrain[np.argpartition(dist, k)[:k]])
9. **def** myknn(xtest, xtrain, ytrain, k):
10. **return** np.apply\_along\_axis(l2NormAvgOutput, 1, xtest, xtrain, ytrain, k)

b) Generate 1000 observations:

* Autoregressive covariance matrix:
  + (i,j)th entry equal to 0.5|i−j|

1. cov = np.full(shape=[numOfIndependentVar, numOfIndependentVar], fill\_value=1.0, dtype=float)
2. M,N = cov.shape
3. X, Y = np.ix\_(np.arange(M), np.arange(N))
4. out = cov \* (0.5 \*\* abs(X - Y))

* Mean vector (defaulted to vector given in the homework, but made it a param to the method, please see the full method at the end of this section): mean=np.array([1, 2, 3, 4, 5])
* Setting the seed: np.random.seed(1)
* X (independent variables generation):
  + Here I have used scipy.stats.multivariate\_normal library
* np.random.seed(1)
* rv = multivariate\_normal.rvs(mean=mean, cov=out, size=totalNumOfObservations)
* Y generation:
* mu, sigma = 0, 1 # mean and standard deviation
* epsilon = np.random.normal(mu, sigma, 1000)
* y = rv[:, 0] + rv[:, 1] + ((rv[:, 2] - 2.5) \*\* 2) + epsilon
* Complete method:

1. **def** generateObservations(numOfIndependentVar=5, mean=np.array([1, 2, 3, 4, 5]), totalNumOfObservations=1000):
2. cov = np.full(shape=[numOfIndependentVar, numOfIndependentVar], fill\_value=1.0, dtype=float)
3. M,N = cov.shape
4. X, Y = np.ix\_(np.arange(M), np.arange(N))
5. out = cov \* (0.5 \*\* abs(X - Y))
6. #Setting seed as 1
7. np.random.seed(1)
8. rv = multivariate\_normal.rvs(mean=mean, cov=out, size=totalNumOfObservations)
9. mu, sigma = 0, 1 # mean and standard deviation
10. epsilon = np.random.normal(mu, sigma, 1000)
11. y = rv[:, 0] + rv[:, 1] + ((rv[:, 2] - 2.5) \*\* 2) + epsilon
13. # Printing the first 3 observations
14. **print**(rv[0:3, :], y[0:3])
15. **return** rv, y