Name: Deep Jahan Grewal
Roll No.: 201364124
Assignment – 1

K Nearest Neighbor Classifier

Introduction:
K Nearest Neighbor classifier is used to classify certain test data into a class based on training data. If majority K nearest neighbors belong to a certain class, the test data set is classified into that class. The nearest neighbors are decided according to a distance formula which may vary for dataset to dataset.
One of the most common distance formula is Euclidean Distance formula where data can be considered as a point in N-dimentional space (N is the number of components of a data which is used to classify it) and distance between two points is calculated as square root of sum of squares of difference between components of data.

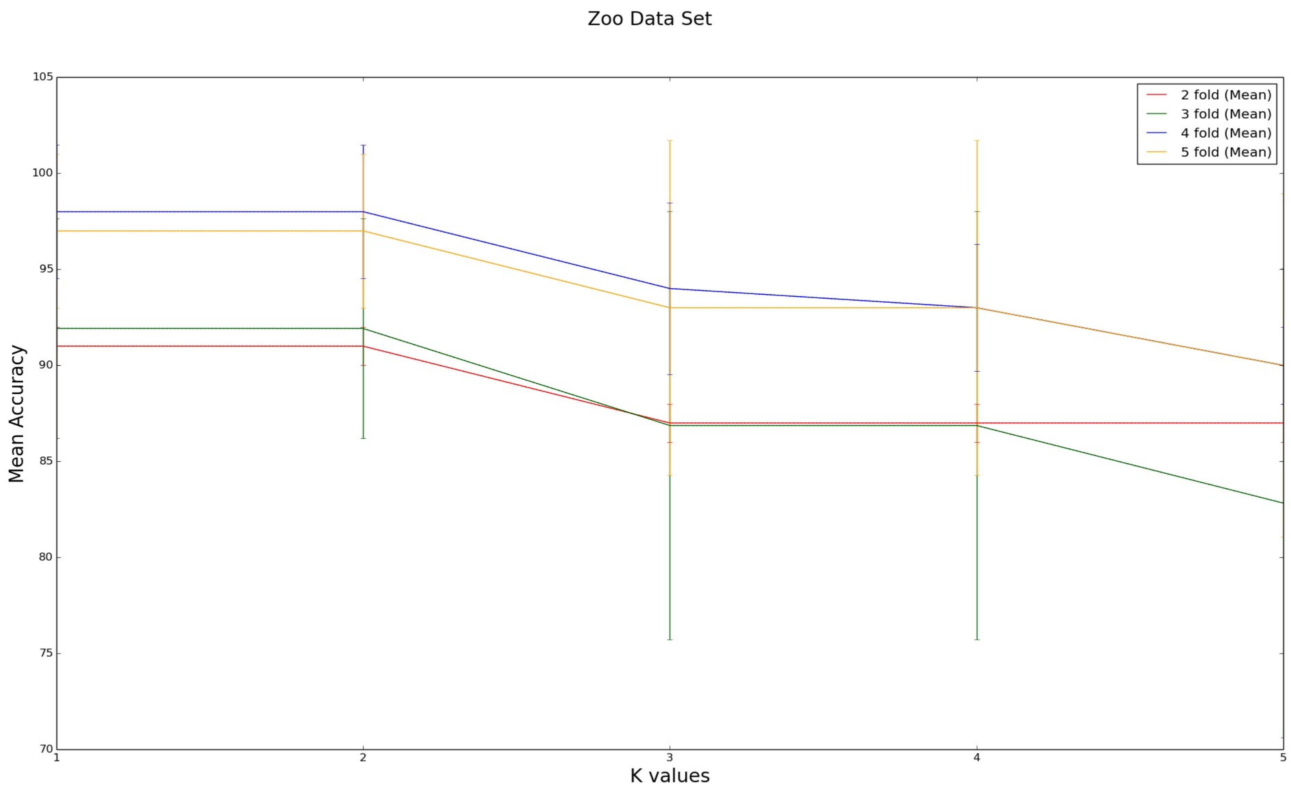
Code:
Python is used to write the required code. The code is as follows.

```
import math
import random
import matplotlib.pyplot as plt

#Load data from a file into lists
def loadDataset(file,ignore):
	f=open(file,"r")
	lin=f.readline().split()
	data=[]
	while(lin):
		ele=[]
		for i in xrange(len(lin)-1):
			if i not in ignore:
				ele.append(float(lin[i]))
		ele.append(lin[-1])
		data.append(ele)
		lin=f.readline().split()
	f.close()
	return data

#Divide the data randomly into 'folds' parts
def divideDataset(data,folds):
	random.shuffle(data)
	length=len(data)
	bound = int(length/folds)
	dataSet=[]
	for i in xrange(folds):
		dataSet.append(data[bound*i:bound*(i+1)])
	return dataSet

#Euclidean Distance Formula used
def euclideanDistance(point1, point2, length):
	distance=0
	for i in xrange(length):
        distance+=pow(point1[i]-point2[i],2)
    return math.sqrt(distance)

#Return k nearest neighbors of 'test' data
def nearestNeighbors(trainSet,test,k):
    neighbors=[]
    length=len(test)-1
    def getkey(item):
        return item[1]
    for i in xrange(len(trainSet)):
        neighbors.append( (trainSet[i],euclideanDistance(trainSet[i],test,length)) )
    neighbors.sort(key=getkey)
    k_neighbors=[]
    for i in neighbors[0:k]:
        k_neighbors.append(i[0])
    return k_neighbors

#Classify a test data into majority class of k neighbors.
#If a tie occurs, the nearest neighbor which is involved in tie is chosed and its class is used to
classify the given test data.
def classify(neighbors):
    classes={}
    maxclass=[]
    maxx=-1
    for i in neighbors:
        clas = i[-1]
        if clas in classes:
            classes[clas] += 1
        else:
            classes[clas] = 1
        if classes[clas] > maxx:
            maxx=classes[clas]
            maxclass=[clas]
        elif classes[clas] == maxx:
            maxclass.append(clas)
    for i in neighbors:
        if maxclass[0]==i[-1]:
            return i[-1]

#Predicts class of all the test data in 'testSet' according to 'trainSet'
def predict(trainSet,testSet,k):
    predictions=[]
    for i in xrange(len(testSet)):
        neighbors = nearestNeighbors(trainSet,testSet[i],k)
        predictions.append(classify(neighbors))
    return predictions

#Calculates Accuracy i.e, number of correct predictions by total predictions multiplied by 100
def calcAccuracy(testSet, predicted):
    correct=0
    length=len(testSet)
    for i in xrange(length):
        if testSet[i][-1]==predicted[i]:
            correct+=1
    return (correct/float(length)) * 100.0

#Plotting graph of Mean Accuracy with K values
def plot(K, plotData,plotData2):
    cols=["red","darkgreen","blue","orange"]
    for i in xrange(4):
        plt.plot(K,plotData[i],label=(str(i+2)+" fold (Mean)"),color=cols[i])
        plt.errorbar(K,plotData[i],yerr=plotData2[i],ls='dotted',color=cols[i])
    plt.legend()
    plt.xlabel("K values",fontsize=20)
    plt.ylabel("Mean Accuracy",fontsize=20)
    plt.suptitle("Ecoli Data Set",fontsize=20)
    plt.xticks(K)
    plt.show()

def main():
    plotData=[]
    plotData2 = []
    data=loadDataset('ecoli.data',[0])
    #For each fold in [2,5] and for each k in [1,5], accuracy is measured
    for folds in xrange(2,6):
        dataSet = divideDataset(data,folds)
        plotFold=[]
        plotFold2 = []
        for k in xrange(1,6):
            averageAccuracy=0.0
            accuracies=[]
            for i in xrange(folds):
                testSet=dataSet[i]
                trainSet=[]
                for j in xrange(folds):
                    if j!=i:
                        trainSet+=dataSet[j]
                predictions=predict(trainSet,testSet,k)
                accuracies.append(calcAccuracy(testSet,predictions))
                averageAccuracy+=accuracies[-1]
            averageAccuracy/=float(folds)
            dev=0.0
            for i in accuracies:
                dev+=pow((i-averageAccuracy),2)
            dev/=float(folds)
            dev=math.sqrt(dev)
            plotFold.append(averageAccuracy)
            plotFold2.append(dev)
        plotData.append(plotFold)
        plotData2.append(plotFold2)
    K=[1,2,3,4,5]
    plot(K,plotData,plotData2)

main()
```

A graph is plotted between Mean Accuracy and K values for ecah fold for a particular dataset. 4 Datasets are used.

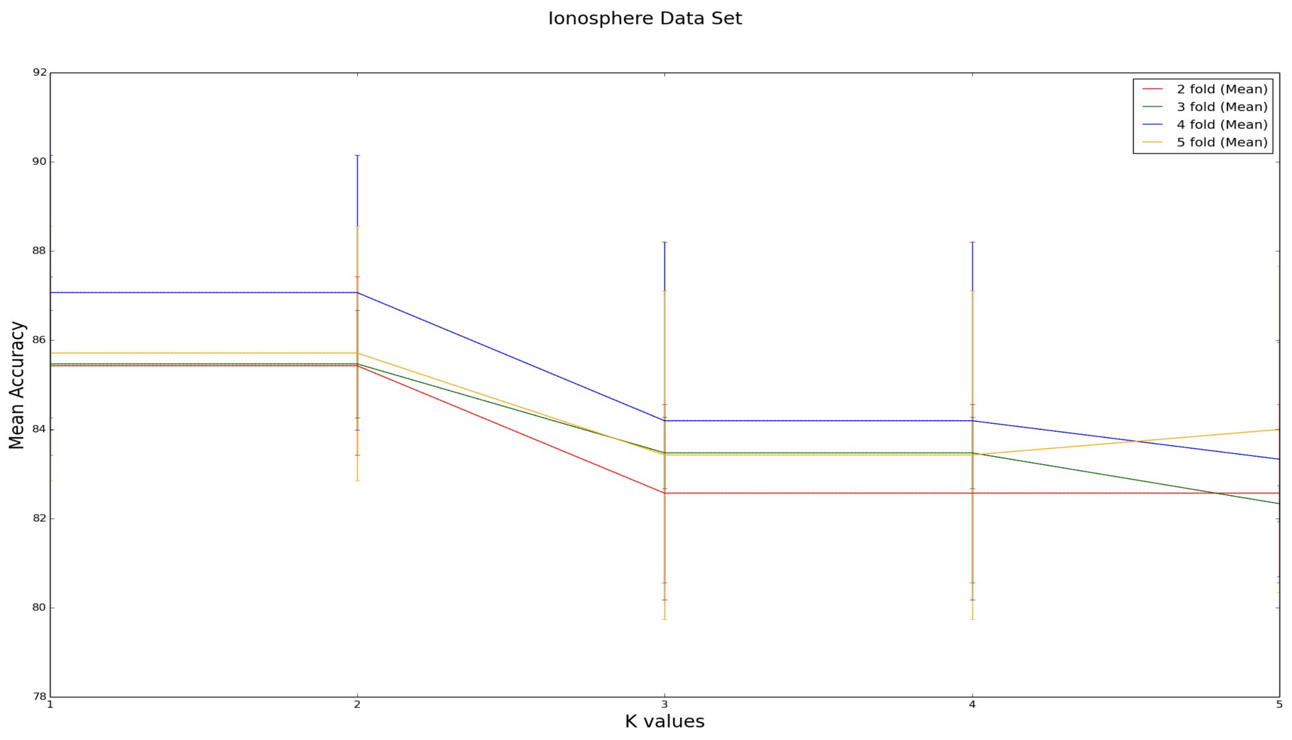
Datasets:

1. Zoo Data Set : https://archive.ics.uci.edu/ml/datasets/Zoo
   This data set contains different information about animals and each of the feature except name and class is boolean. Features include whether it has feathers or not, whether it is venomous or not etc. According to features, 7 classes are made. Since no algorithm is specified for classfication, Euclidean distance formula can be used on trial basis and see if the results are good.



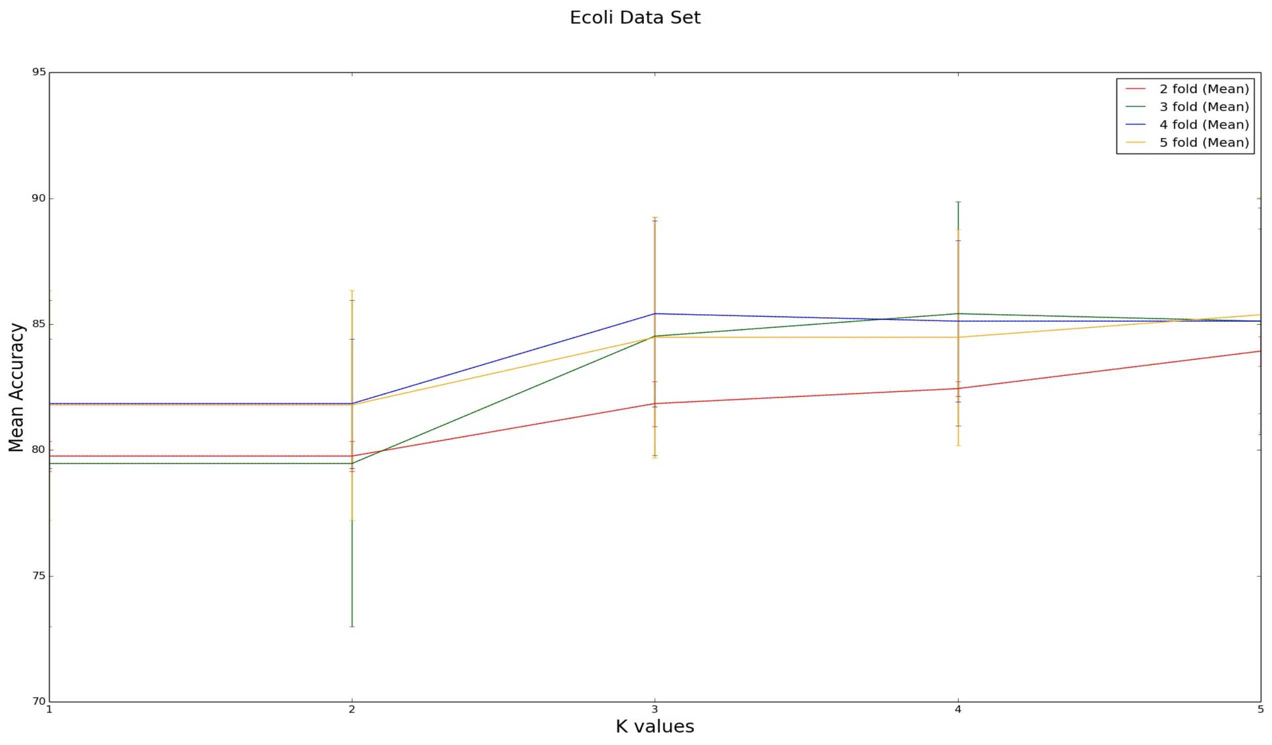
   As seen from the graph, for low value of k, the algorithm gives pretty good accuracy. Though the standard deviation of 3 fold and 5 fold are quit high (8-10%) , standard deviation for 2 fold is quite low. Standard deviation also increases with k. Overall taking a small k will give best results.

2. Ionosphere Data Set : https://archive.ics.uci.edu/ml/datasets/Ionosphere
   This data set consists of radar data which was collected by a system in Goose Bay, Labrador. This system consists of a phased array of 16 high-frequency antennas with a total transmitted power on the order of 6.4 kilowatts. The targets were free electrons in the ionosphere. "Good" radar returns are those showing evidence of some type of structure in the ionosphere. "Bad" returns are those that do not; their signals pass through the ionosphere. Euclidean distance formula should work fine because all data components are given equal weightage.



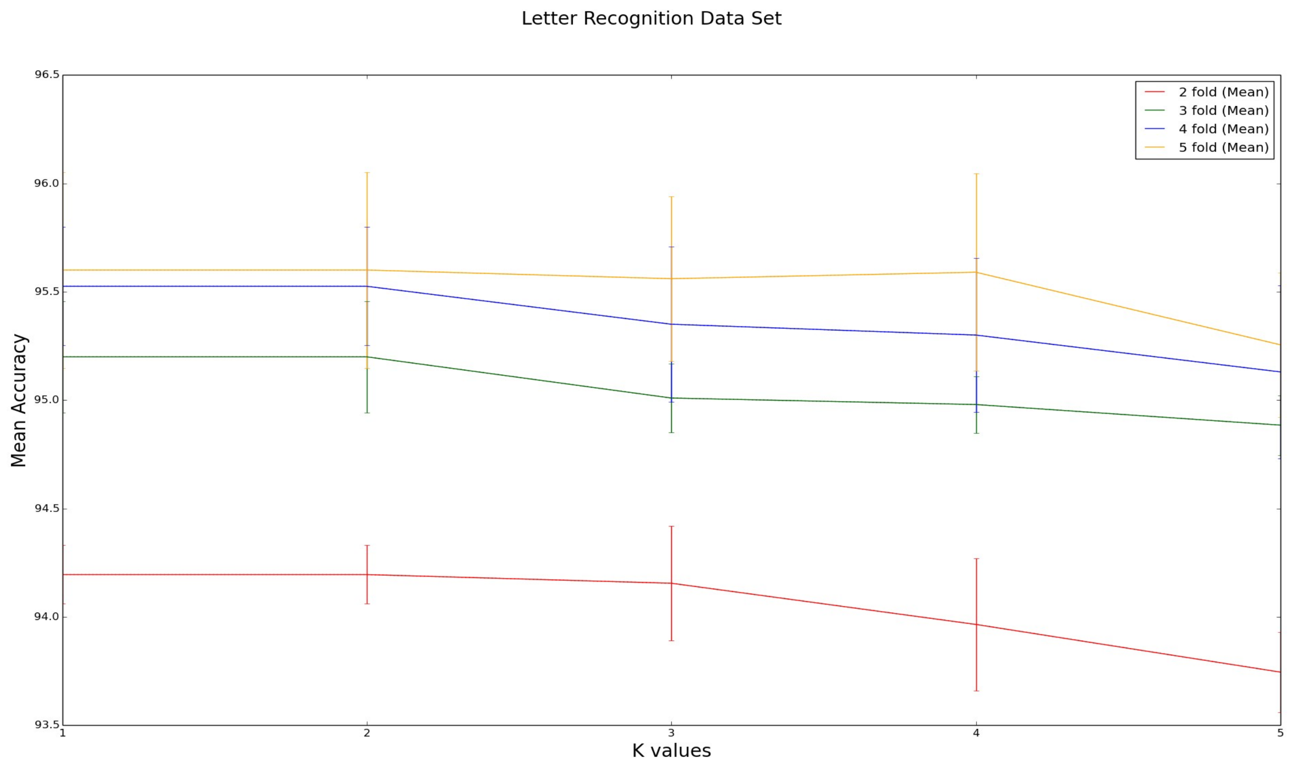
The results are quite similar to Zoo Data Set except the fact that now the accuracy is little less and 3 fold curve shows low error this time instead of 2 fold.

3. Ecoli Data Set : https://archive.ics.uci.edu/ml/datasets/Ecoli
   This data set gives a probablistic classification system for predicting the cellular localization sites of proteins. Attributes include different data like signal sequence recognition using different methods.



The results are quite different. The standard deviations are lower this time and the accuracy increases with k. Though the accuracy is lower than the zoo data set, KNN classifier works more consistantly on this data set.

4. Letter Recognition Data Set : https://archive.ics.uci.edu/ml/datasets/Letter+Recognition
   This data set contains information to identify each of a large number of black and white rectangular pixel displays as one of the 26 capital letters in the English alphabet.



This is a pretty large data set and took hours to complete.
This data set gave good results with high accuracy and low standard deviation. It doesn't depend much on the value of k(decreases slightly) and having mor data set is clearly more beneficial.