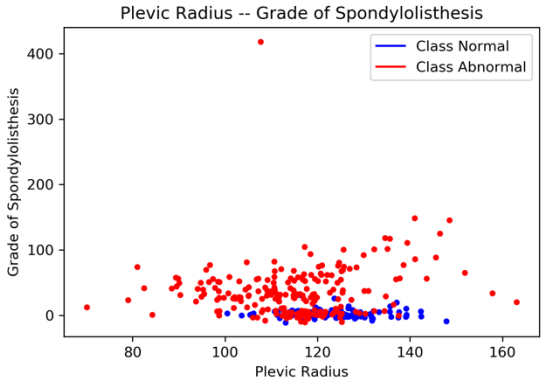
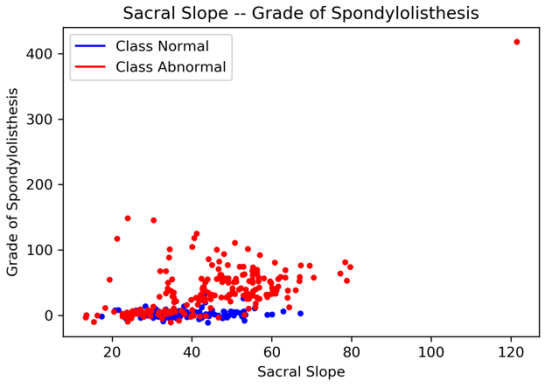
Homework 1 for EE559
Author: Chengyao Wang
USCID: 6961599816
Contact: chengyao@usc.edu
Topic: K Nearest Neighbors Classification Algorithm

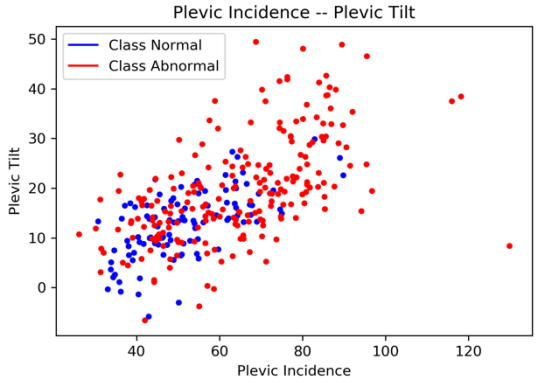
Results and Brief Discussion:

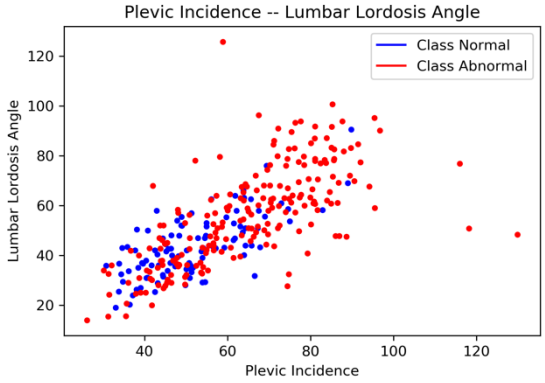
(b)[Pre-processing & Ploting]:

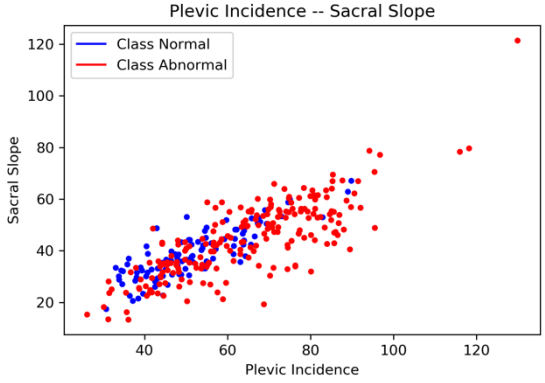
Results:

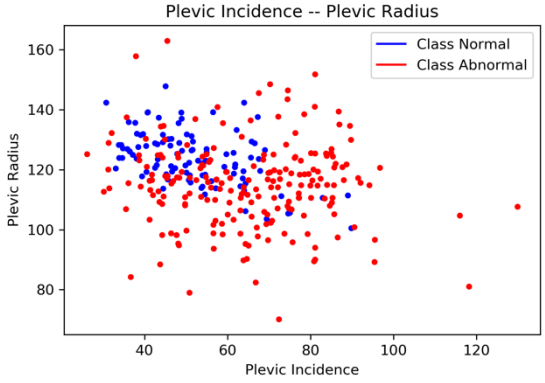


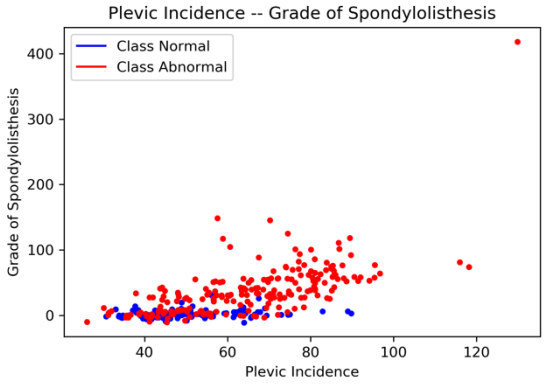


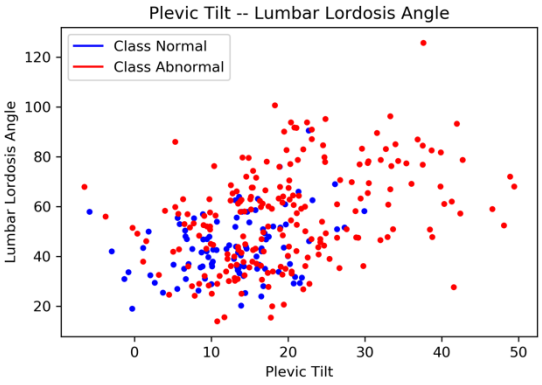


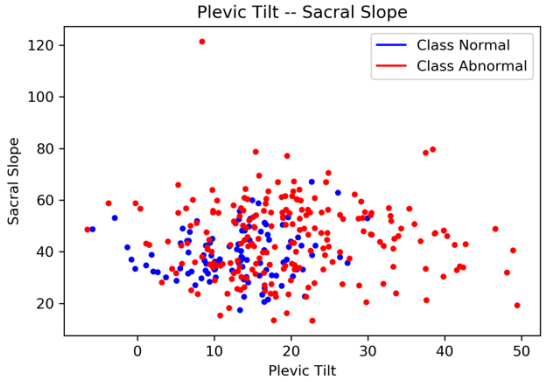


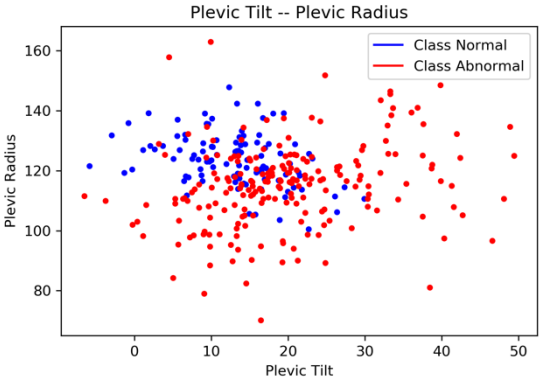


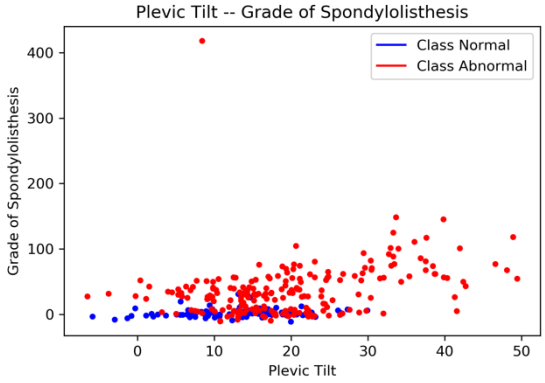


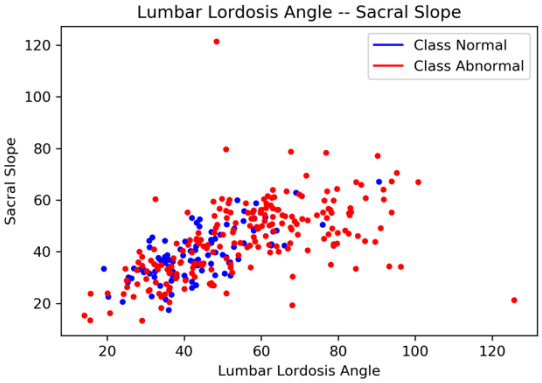

Plevic Incidence -- Grade of Spondylolisthesis
Class Normal
Class Abnormal
Grade of Spondylolisthesis
Plevic Incidence

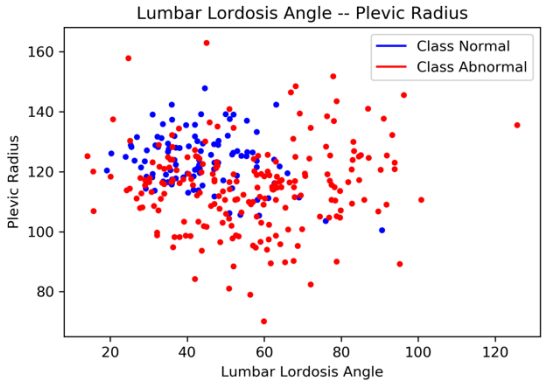

Plevic Tilt -- Lumbar Lordosis Angle
Class Normal
Class Abnormal
Lumbar Lordosis Angle
Plevic Tilt

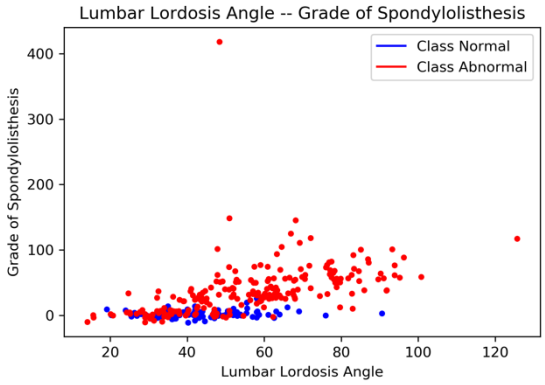

Plevic Tilt -- Sacral Slope
Class Normal
Class Abnormal
Sacral Slope
Plevic Tilt

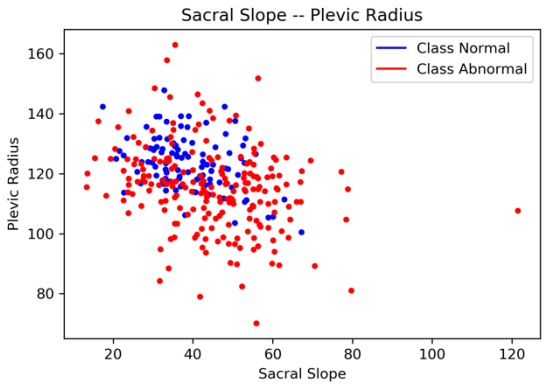

Plevic Tilt -- Plevic Radius
Class Normal
Class Abnormal
Plevic Radius
Plevic Tilt

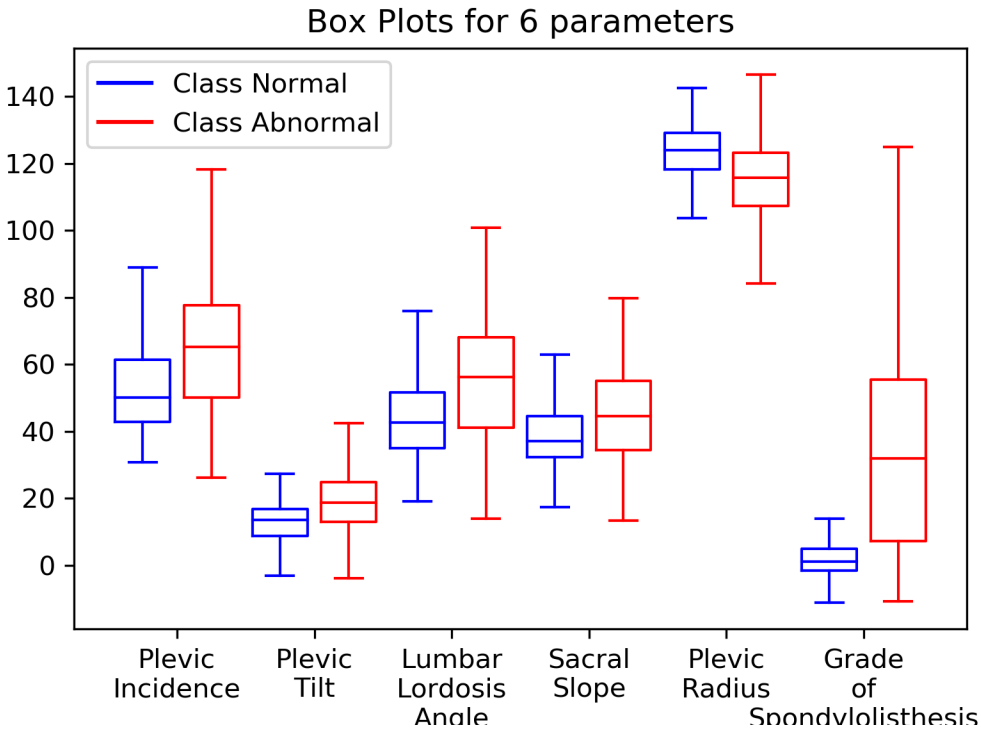

Plevic Tilt -- Grade of Spondylolisthesis
Class Normal
Class Abnormal
Grade of Spondylolisthesis
Plevic Tilt


Lumbar Lordosis Angle -- Sacral Slope
Class Normal
Class Abnormal
Sacral Slope
Lumbar Lordosis Angle









Sketch of Approach:

Used Matplotlib.pyplot to plot these scatter plots. There are six parameters associated with each individual instance, taking two to form one scatter plot and making the total to

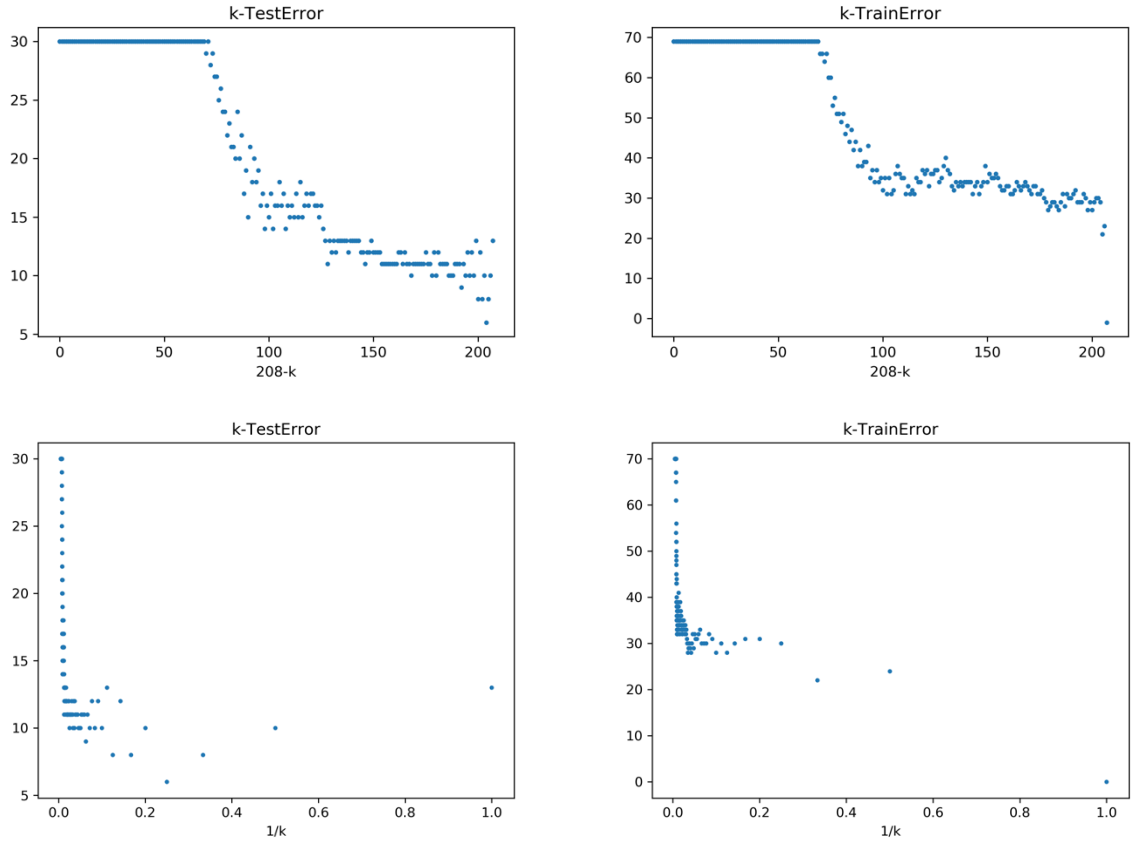
$C_6^2 = 15$ number scatter plots. Blue points are instances with true label "Normal", with red ones "Abnormal".

Discussion:

There is not much to discuss regarding plotting. But as a data pre-processing step, we can notice from the scatter plots & box plots that **different parameters contributes differently to the true label.** "Grade of Spondylolisthesis" in obvious a good critiria for classifying Normal/Abnormal, while "Plevis Tilt" may isn't. Kicking out parameters that are less related to classification may improve the performance of KNN because it reduces noise. But in this project, **all of them** are used anyway.

(cii)[KNN-Finding k^*]:

Result:



The Optimal K found is: 4 With Mis-shots: 6 Out of 100 tests
Correctness under optimal K:
Test error: 6%
# of True Positive: 69
# of False Positive: 5
# of False Negative: 1
# of True Negative: 25
Confusion Matrix:

	True Label (Positive)	True Label (Negative)	Total
Predicted Label (Positive)	69	5	74
Predicted Label (Negative)	1	25	26
Total	70	30	

True Positive Rate: 0.9857
True Negative Rate: 0.8333
Precision: 0.9324
F-Score: 0.9583

Noticing:

1. x-axis for the previous two plots is the value of "208-k".
2. x-axis for the previous two plots is the value of "1/k".
3. y-axis is the number of errors, rather than error rate.
4. Increment step size of k is 1.

Sketch of Approach:

Way of classifing one test data using KNN:

1. Load the data.
2. Choose k and metric type.
3. Iterate through every training data point and calculate the distance between test data and each training data. And also associate the true label of the training data point with this distance.
4. Sort the calculated distances in ascending order based on distance values.
5. Get top k rows from the sorted array, and get the most frequent class among these rows. Take this class as the classification result of the test data.

Under the same k, iterate the above process through every test data. Comparing the "true label" and "classification result" of these test datas to acquire the test error rate. Try different k, and take the **ones which have the lowest test error rate**. Theortically, every one of them can be the optimal k. In this project **the median of the ks** that shares the lowest error rate are taken as k^*.

Perform KNN again using the k^* acquired to achieve:

i) Confusion matrix, i.e. number of true positive, true negative, false positive, false negative.

ii) True positive rate $\frac{tp}{tp+fn}$, True negative rate $\frac{tn}{tn+fp}$.

iii) Precision $\frac{tp}{tp+fp}$and F-score $\frac{tp\_rate}{tp\_rate+precision}$.

Discussion:

1. KNN is quite effective in this case, with merely 6% of test errors.
2. Parameter "k" implemented in KNN **has to be an integer** in [1, $\#_{train\_data}$]. Because when selecting the closest neighbor measured in the metric type specified, **we have to choose at least one nearest test data and at most all test data.**

3. The plots of the two errors with k is consistent with the preceeding claim. When k=208, nearly all test datas contribute to the classification, making classification result always "Abnormal" since we have more "Abnormal" test data than "Normal" ones, leading to $test\ error = \frac{30}{30+70} = 30\%, train\ error = \frac{70}{70+140} \approx 33.3\%$. When k=1, training error is 0 because the closest training data is itself, thus is always correct.

4. **As k decreases, trainig error decreases, while test error decreases in the beginning and goes up after passing the optimal k**. In this scenario, k^* is close to 1 compared with the range of k, so **the rise of test error is hard to spot** in the scatter plot, but **still doubled the optimal test error** when k=1.

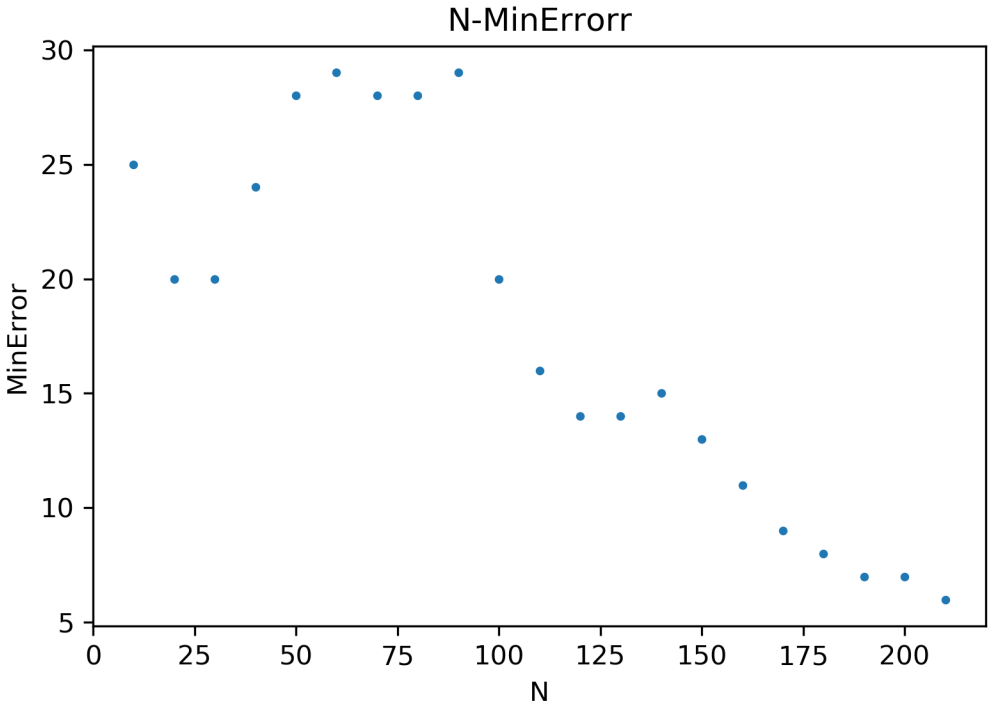
5. **F-score is also a good criterion to choose k^* other than test error**, the former performs better in situtations where test datas are extremely biased. In this case where the ratio of two classes are 2:1, which is not so biased, both assessment parameters will do.

6. **The two errors stopped changing when k gets larger than approximately 130**. One reason might be: when k gets large enough, the statistical characteristics (ratio of two classes in this case) of the neighbor of any individual test data **started to behave like** that of the complete training data set – converging to 30%/33.3%. On the other hand, the absence of even a ± 1 fluctuation probabily resulted from this model, which means it's **unique to this particular probelem**.

7. **True positive rate is larger than true negative rate**. It is partially resulted from **the bias of the data**, and will continue to increase if the bias gets larger, which we **can and should** avoid by using techniques like up-sampling or SMOTE.

(ciii)[KNN-different training data size]

Result:



Sketch of Approach:

Select a $N = \{10, 20, 30, \ldots\ldots, 210\}$ total number of training data with a distribution of 1:2 between "Normal" & "Abnormal" data from the top of complete "Normal" and "Abnormal" data tests to perform KNN, finding its k^* & min_{test_error}. The test data set is not changed. **The k increment step remains 1** to get a better description of the relationship between min_{test_error} and N.

Discussion:

1. The minimum test error achieved by varying k with a given training data size is **overall decreasing as the training data size gets larger**. It is both consistent with intuition and theory – **the more training data used, a better estimation of the true classifer model we'll get.**

2. The unusual uplift of minimum test error when $k \in [50, 90]$ and 140 **may be unique to this particular scenario**, meaning that the true classifier model also has this unusual property as well, or can also be a result of **un-randomness of the orders by which every training instance is listed in raw data**, since we are taking a certain number of points from **top to bottom without randomizing.**

3. When N=210, the test error rate has reached a considerably low rate of 6%, both relatively and absolutely. **The marginal benefit of using more data in training is decreasing as the training size increases.** So there also exists an **optimal point that balances the cost (time, money, etc) and result procision.**

(d)[KNN-different metric types]

Result:

Minimun Error Summary (Optimal K, Minimum Error):
Manhattan Distance: (10, 10)
Euclidean Distance: (4, 6)
P-Minkowski Distance with log(p)= 0.1 : (4, 8)
P-Minkowski Distance with log(p)= 0.2 : (4, 6)
P-Minkowski Distance with log(p)= 0.3 : (4, 6)
P-Minkowski Distance with log(p)= 0.4 : (6, 7)
P-Minkowski Distance with log(p)= 0.5 : (4, 6)
P-Minkowski Distance with log(p)= 0.6 : (5, 6)
P-Minkowski Distance with log(p)= 0.7 : (4, 6)
P-Minkowski Distance with log(p)= 0.8 : (4, 7)
P-Minkowski Distance with log(p)= 0.9 : (8, 7)
P-Minkowski Distance with log(p)= 1 : (8, 7)
Chebyshev Distance: (2, 6)
Mahalanobis Distance: (4, 12)

The test error using P-Minkowski Distance under k=10 is:
P-Minkowski Distance with log(p)= 0.1 : (10, 11)
P-Minkowski Distance with log(p)= 0.2 : (10, 11)
P-Minkowski Distance with log(p)= 0.3 : (10, 10)
P-Minkowski Distance with log(p)= 0.4 : (10, 11)
P-Minkowski Distance with log(p)= 0.5 : (10, 11)
P-Minkowski Distance with log(p)= 0.6 : (10, 10)
P-Minkowski Distance with log(p)= 0.7 : (10, 10)
P-Minkowski Distance with log(p)= 0.8 : (10, 10)
P-Minkowski Distance with log(p)= 0.9 : (10, 10)
P-Minkowski Distance with log(p)= 1 : (10, 11)
Notice: the increment step size of k is 1.

Solution:

Run the KNN with different metric types.

Discussion:

1. Manhattan, Euclidean, P-Minkowski and Chebyshev Distance are all induced by p-norm in Euclidean space when $p = 1, 2, p, \infty$ respectively. Also:

$$log_{10}(p) = \{0.1, \dots\dots, 1\} \Rightarrow p = \{1.26, 1.58, 1.99, 2.51, 3.16, 3.98, 5.01, 6.31, 7.94, 10\}$$

2. **The relationship between p and minimum test error is not apparent**. Minimum test error decreases when $p < 1.5$ and stays **approximately the same when p continues to increase.**

3. The k^* associated with minimum test error **doesn't vary much** as p increases, **expect** when $k^* = 10$ in Manhattan Distance and $k^* = 8$ when $p = 7.94$ & 10.

4. The performance of KNN using P-Minkowski Distance under:

$$k_{p-Minkowski} = k^*_{Manhattan} = 10$$

Doesn't vary much as p changes.

5. Mahalanobis Distance is a **multi-dimensional generalization of the idea of measuring how many standard deviations away the points are from each other**, which in

other words **normalization**. Normalization **may improve or worsen** the classification error, **in this case, definitely worsen.**

6. Choice of best metric type **should take both the minimum test error and k^* into consideration. The smaller error and k^* is, the better the metric type**. Also, the calculation is **faster if p takes integer values**. In this particular case, **Euclidean Distance is the best.** Mahalanobis Distance may lead to a better result **when the parameters contribute similarly to the classification or parameters that contributes less to the classification has a larger standard deviation / variance**, which is not the case in this scenario. From the box plot presented above, the major variable in classification "Grade of Spondylolisthesis" actually **has the largest standard deviation / variance.**

(d)[KNN-weighted decision]

Result:

Minimun Error Summary of Weighted Polling (Optimal K, Minimum Error):
Manhattan Distance: (26, 10)
Euclidean Distance: (6, 10)
Chebychev Distance: (101, 70)
Notice: the increment step size of k is 1.

Solution:

In KNN without weighted decision, the frequency of the k nearest neighbors of the test data classifies the test data. After introducing weighted decision, the contribution of every nearest neighbor is now $\frac{100}{distance}$ rather than 1. The rest is the same.

Discussion:

1. **In this scenario**, weighted decision polling is worse than majority polling. **But it can't draw any conclusions on the general case**. It's just another trial of improving KNN algorithm, and couldn't **guarantee on its positive effect on the final result**.

Appendix:

Written using Spyder(Python 2.7)

```
# -*- coding: utf-8 -*-
"""
Created on May Mon 10:38:04 2019
Homework1 for EE559
Author:Chengyao Wang
USCID:6961599816
Contact Email:chengyao@usc.edu
"""
import os
import numpy as np
import matplotlib.pyplot as plt
#currentDirectory = os.getcwd()
#print currentDirectory
```

```
os.chdir("/Users/Gaara/Desktop/USC/EE559/Homework/Homework1/vertebral_column_data")
with open("column_2C.dat") as f:
    dataset=f.readlines()
#Pre-processing Data: Turning Strings into float Arrays
#Column 1~6:Value of parameters
#Column 7:True label, Normal=0, Abnormal=1
no_count=0
ab_count=0
noSet=np.zeros((100,7))
abSet=np.zeros((210,7))
for i in range(len(dataset)):
    dataBuff=np.fromstring(dataset[i],dtype=float,count=6,sep=' ')
    if dataset[i][len(dataset[i])-2]=='O':
        noSet[no_count,:len(dataBuff)]+=dataBuff
        noSet[no_count,6]=0
        no_count+=1
    elif dataset[i][len(dataset[i])-2]=='B':
        abSet[ab_count,:len(dataBuff)]+=dataBuff
        abSet[ab_count,6]=1
        ab_count+=1
#Covariance Matrix Calculation
cov_mtx=np.zeros((6,6))
mean_mtx=np.zeros(6)
for i in range(0,70):
    mean_mtx+=noSet[i,:6]/210
for i in range(0,140):
    mean_mtx+=abSet[i,:6]/210
for i in range(0,6):
    for j in range(i,6):
        for k in range(70):
            cov_mtx[i][j]+=(noSet[k][i]-mean_mtx[i])*(noSet[k][j]-mean_mtx[j])
        for k in range(140):
            cov_mtx[i][j]+=(abSet[k][i]-mean_mtx[i])*(abSet[k][j]-mean_mtx[j])
        cov_mtx[i][j]/=(210-1)
        cov_mtx[j][i]=cov_mtx[i][j]
#print mean_mtx
#print cov_mtx
#print np.linalg.inv(cov_mtx)
#Present scatter plots, total number of 15
#Each scatter plots are one-one combination of 6 parameters
#Save the plots for a higher resolution
def scatter_plot():
    count=0 #used for plot labeling
    for i in range(6):
        for j in range(i+1,6):
            plt.scatter(noSet[:,i],noSet[:,j],c='blue',s=10)
            plt.scatter(abSet[:,i],abSet[:,j],c='red',s=10)
            plt.plot([],c='blue',label='Class Normal')
            plt.plot([],c='red',label='Class Abnormal')
            plt.legend()
            if count==0:
                plt.title("Plevic Incidence -- Plevic Tilt")
                plt.xlabel("Plevic Incidence")
                plt.ylabel("Plevic Tilt")
                plt.savefig('1.png', dpi=300)
            elif count==1:
                plt.title("Plevic Incidence -- Lumbar Lordosis Angle")
                plt.xlabel("Plevic Incidence")
                plt.ylabel("Lumbar Lordosis Angle")
                plt.savefig('2.png', dpi=300)
            elif count==2:
                plt.title("Plevic Incidence -- Sacral Slope")
                plt.xlabel("Plevic Incidence")
                plt.ylabel("Sacral Slope")
```

```
            plt.savefig('3.png', dpi=300)
        elif count==3:
            plt.title("Plevic Incidence -- Plevic Radius")
            plt.xlabel("Plevic Incidence")
            plt.ylabel("Plevic Radius")
            plt.savefig('4.png', dpi=300)
        elif count==4:
            plt.title("Plevic Incidence -- Grade of Spondylolisthesis")
            plt.xlabel("Plevic Incidence")
            plt.ylabel("Grade of Spondylolisthesis")
            plt.savefig('5.png', dpi=300)
        elif count==5:
            plt.title("Plevic Tilt -- Lumbar Lordosis Angle")
            plt.xlabel("Plevic Tilt")
            plt.ylabel("Lumbar Lordosis Angle")
            plt.savefig('6.png', dpi=300)
        elif count==6:
            plt.title("Plevic Tilt -- Sacral Slope")
            plt.xlabel("Plevic Tilt")
            plt.ylabel("Sacral Slope")
            plt.savefig('7.png', dpi=300)
        elif count==7:
            plt.title("Plevic Tilt -- Plevic Radius")
            plt.xlabel("Plevic Tilt")
            plt.ylabel("Plevic Radius")
            plt.savefig('8.png', dpi=300)
        elif count==8:
            plt.title("Plevic Tilt -- Grade of Spondylolisthesis")
            plt.xlabel("Plevic Tilt")
            plt.ylabel("Grade of Spondylolisthesis")
            plt.savefig('9.png', dpi=300)
        elif count==9:
            plt.title("Lumbar Lordosis Angle -- Sacral Slope")
            plt.xlabel("Lumbar Lordosis Angle")
            plt.ylabel("Sacral Slope")
            plt.savefig('10.png', dpi=300)
        elif count==10:
            plt.title("Lumbar Lordosis Angle -- Plevic Radius")
            plt.xlabel("Lumbar Lordosis Angle")
            plt.ylabel("Plevic Radius")
            plt.savefig('11.png', dpi=300)
        elif count==11:
            plt.title("Lumbar Lordosis Angle -- Grade of Spondylolisthesis")
            plt.xlabel("Lumbar Lordosis Angle")
            plt.ylabel("Grade of Spondylolisthesis")
            plt.savefig('12.png', dpi=300)
        elif count==12:
            plt.title("Sacral Slope -- Plevic Radius")
            plt.xlabel("Sacral Slope")
            plt.ylabel("Plevic Radius")
            plt.savefig('13.png', dpi=300)
        elif count==13:
            plt.title("Sacral Slope -- Grade of Spondylolisthesis")
            plt.xlabel("Sacral Slope")
            plt.ylabel("Grade of Spondylolisthesis")
            plt.savefig('14.png', dpi=300)
        elif count==14:
            plt.title("Plevic Radius -- Grade of Spondylolisthesis")
            plt.xlabel("Plevic Radius")
            plt.ylabel("Grade of Spondylolisthesis")
            plt.savefig('15.png', dpi=300)
        plt.show()
        count+=1
#Present boxplot
```

```
#There are 6 scatter plots with 6 parameters
#Save the plots for a higher resolution
def set_box_color(bp,color):
    plt.setp(bp['boxes'],color=color)
    plt.setp(bp['whiskers'],color=color)
    plt.setp(bp['caps'],color=color)
    plt.setp(bp['medians'],color=color)
def box_plot():
    plt.figure()
    data_a=[noSet[:,0],noSet[:,1],noSet[:,2],noSet[:,3],noSet[:,4],noSet[:,5]]
    data_b=[abSet[:,0],abSet[:,1],abSet[:,2],abSet[:,3],abSet[:,4],abSet[:,5]]
    bpl=plt.boxplot(data_a, positions=[1,3,5,7,9,11],sym='',widths=0.8)
    bpr=plt.boxplot(data_b, positions=[2,4,6,8,10,12],sym='',widths=0.8)

plt.xticks([1.5,3.5,5.5,7.5,9.5,11.5],['Plevic\nIncidence','Plevic\nTilt','Lumbar\nLordosis\nAngle','Sacral\nSlope','Ple
vic\nRadius','Grade\nof\nSpondylolisthesis'])
    set_box_color(bpl,'blue')
    set_box_color(bpr,'red')
    plt.plot([],c='blue',label='Class Normal')
    plt.plot([],c='red',label='Class Abnormal')
    plt.legend()
    plt.xlim(0,13)
    plt.title("Box Plots for 6 parameters")
    plt.savefig('box.png', dpi=300)
#-------------------------------------------------------------------------------------------
#KNN Perform
#-------------------------------------------------------------------------------------------
#Calculation of different metrics, chosen by string metric_type
def metric_cal(data1,data2,metric_type='euclidean',p=1):
    distance=0.0
    if metric_type=='euclidean':
        for i in range(len(data1)):
            distance+=np.square(data1[i]-data2[i])
        return np.sqrt(distance)
    elif metric_type=='manhattan':
        for i in range(len(data1)):
            distance+=np.abs(data1[i]-data2[i])
        return distance
    elif metric_type=='chebyshev':
        metric_max=0
        for i in range(len(data1)):
            if metric_max<np.abs(data1[i]-data2[i]):
                metric_max=np.abs(data1[i]-data2[i])
        return metric_max
    elif metric_type=='p-minkowski':
        for i in range(len(data1)):
            distance+=np.power(np.abs(data1[i]-data2[i]),np.power(10,p))
        return np.power(distance,1.0/np.power(10,p))
    elif metric_type=='mahalanobis':
        covMtxInv=np.linalg.inv(cov_mtx)
        for i in range(6):
            for j in range(6):
                distance+=covMtxInv[i][j]*(data1[i]-data2[i])*(data1[j]-data2[j])
        return distance
#Function for sorting a list of row arrays, by certain keywords
#Used in procedure of KNN
def sortFirst(data):
    return data[0]
#KNN Classifier Function
#Takes in a test instance and predict its label
#testInstance: 1*7 float array
#k: int Parameter of KNN
#no_size: int Number of normal data used in training
#ab_size: int Number of abnormal data used in training
```

```
#metric_type: string Different distance measurements
#p: float parameter of Minkowski distance
def KNN_classifier(testInstance,k,no_size,ab_size,metric_type='euclidean',p=1):
    #distance calculation
    distance=np.zeros((no_size+ab_size,2))
    for i in range(no_size):
        distance[i]=(metric_cal(noSet[i,:6],testInstance[:6],metric_type,p),0)
    for i in range(ab_size):
        distance[i+no_size]=(metric_cal(abSet[i,:6],testInstance[:6],metric_type,p),1)
    #Sort distance[] according to distance[0]
    sorted_dis=sorted(distance,key=sortFirst)
    #Classification of testInstance
    no_count=0
    ab_count=0
    for i in range(0,min(k,no_size+ab_size)):
        if sorted_dis[i][1]==0:
            no_count+=1
        elif sorted_dis[i][1]==1:
            ab_count+=1
    #Output Classification result
    #print no_count,ab_count
    #print distance
    if no_count>ab_count:
        #print "This testInstance is Classified Normal via KNN"
        return 0
    elif no_count<ab_count:
        #print "This testInstance is Classified Abnormal via KNN"
        return 1
    else:
        #Equal Situations are considered normal for now
        #May be modified later
        return 0
#Function to perform KNN
#go though all test data and draw conclusion on:
#1. k-test.error relationship
#2. k-train.error relationship
#3. optimal k(determined by choosing the median of the set of k that reaches the minimum test.error simultaneously)
#4. returns: optimal k, minimum test.error
#Train_err Bool: Calculate train.error if TRUE
#Plot Bool: Plot the k-test.error/train.error if TRUE
#k_incre int: k increment/decrement step size
def KNN_perform(no_size,ab_size,train_err,metric_type,plot,k_incre,p=1):
    test_result=np.zeros((100,2))
    test_error=np.zeros((208,1))
    if train_err==True:
        train_result=np.zeros((no_size+ab_size,2))
        train_error=np.zeros((208,1))
    k=208
    #Used for recording minimum test error
    #Choose 100 since all wrongs are not possible
    err_min=100
    while k>=1:
        #test all test data on KNN with current k
        for i in range(0,30):
            test_result[i,:]=(KNN_classifier(noSet[70+i],k,no_size,ab_size,metric_type,p),0)
        for i in range(0,70):
            test_result[i+30,:]=(KNN_classifier(abSet[140+i],k,no_size,ab_size,metric_type,p),1)
        #test all train data on KNN with current k
        if train_err==True:
            for i in range(0,no_size):
                train_result[i,:]=(KNN_classifier(noSet[i],k,no_size,ab_size,metric_type,p),0)
            for i in range(0,ab_size):
                train_result[i+70,:]=(KNN_classifier(abSet[i],k,no_size,ab_size,metric_type,p),1)
        #Calculate the Mistakes of KNN with the specific k
```

```
        #There is always a success when testing train data, subtract it out
        for i in range(0,100):
            test_error[208-k]+=(test_result[i,0]!=test_result[i,1])
        if err_min>test_error[208-k]:
            err_min=test_error[208-k]
        #Calculate the Mistakes of KNN with the specific k
        if train_err==True:
            for i in range(0,210):
                train_error[208-k]+=(train_result[i,0]!=train_result[i,1])
        #Decrease k and error_index
        k-=k_incre
    if plot==True:
        plt.scatter(np.arange(0,len(test_error)),test_error,s=5)
        plt.xlabel("208-k")
        plt.title("k-TestError")
        plt.savefig("K-testerror.png",dpi=300)
        plt.show()
        plt.scatter(np.arange(0,len(train_error)),train_error,s=5)
        plt.title("k-TrainError")
        plt.xlabel("208-k")
        plt.savefig("K-trainerror.png",dpi=300)
        plt.show()
    #Data Post-Process
    #go through the ks which shares the err_min
    optimal_k=np.zeros((208))
    count=0
    for i in range(208):
        if test_error[i]==err_min:
            optimal_k[count]=i
            count+=1
    k_star=208-np.median(optimal_k[:count])
    return int(k_star),int(err_min)
#KNN different k function
#Determine the values of Model assessment
#true.positive, true.negative, false.positive, false.negative
#true.positive rate, true.negative rate, precision, F-score
def KNN_differK():
    k_star,err_min=KNN_perform(70,140,True,'euclidean',True,1)
    print "The Optimal K found is:",k_star,"With Mis-shots:",err_min," Out of 100 tests"
    #Test again with optimal k found
    tp,fp,tn,fn=0,0,0,0
    optimal_test=np.zeros((100,2))
    for i in range(0,30):
        optimal_test[i,:]=(KNN_classifier(noSet[70+i],k_star,70,140,'euclidean',1),0)
    for i in range(0,70):
        optimal_test[i+30,:]=(KNN_classifier(abSet[140+i],k_star,70,140,'euclidean',1),1)
    #Determine tp,fp,tn,fn
    for i in range(0,100):
        classifier=optimal_test[i][0]
        true_label=optimal_test[i][1]
        if (classifier==1)&(true_label==1):
            tp+=1
        elif (classifier==1)&(true_label==0):
            fp+=1
        elif (classifier==0)&(true_label==1):
            fn+=1
        elif (classifier==0)&(true_label==0):
            tn+=1
    #Calculate the rest Parameters
    tp_rate=float(tp)/(tp+fn)
    tn_rate=float(tn)/(tn+fp)
    precision=float(tp)/(tp+fp)
    f_score=2.0*precision*tp_rate/(precision+tp_rate)
    print "Correctness under optimal K",err_min/100
```

```
    print "# of True Positive:",tp
    print "# of False Positive:",fp
    print "# of False Negative:",fn
    print "# of True Negative:",tn
    print "True Positive Rate:",'%.4f'%tp_rate
    print "True Negative Rate:",'%.4f'%tn_rate
    print "Precision:",'%.4f'%precision
    print "F-Score:",'%.4f'%f_score
#KNN traning Size change
#Change the size of training set, perform KNN again
def KNN_trainingSizeChange():
    N=10
    result=np.zeros((21,2))
    while N<=210:
        no_size=int(N/3)
        ab_size=N-no_size
        k,err_min=KNN_perform(no_size,ab_size,False,'euclidean',False,1)
        result[N/10-1]=(err_min,N)
        N+=10
        print "Indicator of Progress, N=",N," Completed"
        #print "Indicator of Completion of N=",N
    plt.scatter(result[:,1],result[:,0],s=5)
    plt.xlabel("N")
    plt.ylabel("MinError")
    plt.title("N-MinErrorr")
    plt.savefig("sizechange.png",dpi=300)
    plt.show()
#KNN metric change
#Change the Metric types, perform KNN again
#Output (int, int): (Optimal K, Minimum Error)
def KNN_metricChange():
    order_list=(0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1)
    #print "Minimun Error Summary (Optimal K, Minimum Error):"
    #print "Manhattan Distance:",KNN_perform(70,140,False,'manhattan',False,1)
    #print "Euclidean Distance:",KNN_perform(70,140,False,'euclidean',False,1)
    print "The test error using P-Minkowski Distance under k=10 is:"
    for i in range(10):
        print "P-Minkowski Distance with log(p)=",order_list[i],":",KNN_perform(70,140,False,'p-
minkowski',False,198,order_list[i])
    #print "Chebyshev Distance:",KNN_perform(70,140,False,'chebyshev',False,1)
    #print "Mahalanobis Distance:",KNN_perform(70,140,False,'mahalanobis',False,1)
#KNN classifier introducing weighted sum
def KNN_weightedClassifier(testInstance,k,metric_type='euclidean'):
    distance=np.zeros((210,2))
    for i in range(70):
        distance[i]=(metric_cal(noSet[i,:6],testInstance[:6],metric_type),0)
    for i in range(140):
        distance[i+70]=(metric_cal(abSet[i,:6],testInstance[:6],metric_type),1)
    sorted_dis=sorted(distance,key=sortFirst)
    no_count=0
    ab_count=0
    for i in range(0,min(k,210)):
        if sorted_dis[i][1]==0:
            no_count+=100.0/sorted_dis[i][0]
        elif sorted_dis[i][1]==1:
            ab_count+=100.0/sorted_dis[i][0]
    if no_count>ab_count:
        return 0
    elif no_count<ab_count:
        return 1
    else:
        return 0
#Function to perform KNN with weighted sum
def KNN_weightedPerform(k_incre,metric_type='euclidean',p=1):
```

```
    test_result=np.zeros((100,2))
    test_error=np.zeros((196,1))
    k=196
    err_min=100
    while k>=1:
        for i in range(0,30):
            test_result[i,:]=(KNN_weightedClassifier(noSet[i+70,:],k,metric_type),0)
        for i in range(0,70):
            test_result[i+30,:]=(KNN_weightedClassifier(abSet[i+140,:],k,metric_type),1)
        for i in range(0,100):
            test_error[196-k]+=(test_result[i,0]!=test_result[i,1])
        if err_min>test_error[196-k]:
            err_min=test_error[196-k]
        k-=k_incre
    optimal_k=np.zeros((196))
    count=0
    for i in range(196):
        if test_error[i]==err_min:
            optimal_k[count]=i
            count+=1
    k_star=196-np.median(optimal_k[:count])
    return int(k_star),int(err_min)
#Function to perform KNN under different metric types with weighted sum
def KNN_weightedMetricChange():
    print "Minimun Error Summary of Weighted Polling (Optimal K, Minimum Error):"
    print "Manhattan Distance:",KNN_weightedPerform(10,'manhattan')
    print "Euclidean Distance:",KNN_weightedPerform(10,'euclidean')
    print "Chebychev Distance:",KNN_weightedPerform(10,'chebychev')
#-------------------------------------------------------------------------------------------
#Function Call
#-------------------------------------------------------------------------------------------
#scatter_plot()
#box_plot()
#KNN_differK()
#KNN_trainingSizeChange()
KNN_metricChange()
#KNN_weightedMetricChange()
```