Name:

Course: CAP 4601

Semester: Summer 2013

Assignment: Assignment 04

Date: 19 JUN 2013

Complete the following written problems:

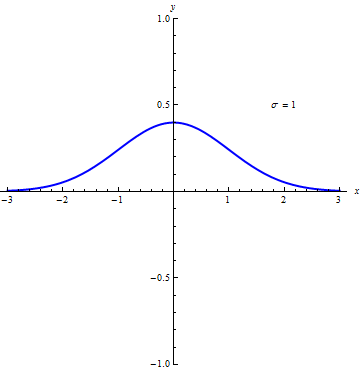
1. The Probability Density Function of the Normal Distribution (50 Points).

The [Normal Distribution](http://en.wikipedia.org/wiki/Normal_distribution) has the following [Probability Density Function](http://en.wikipedia.org/wiki/Probability_density_function) (a.k.a. the "Gaussian"):

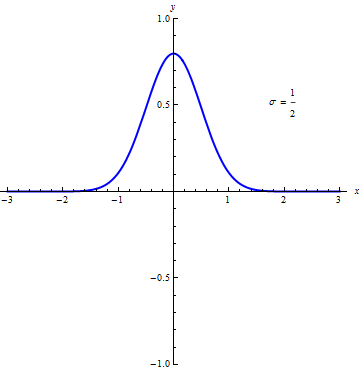
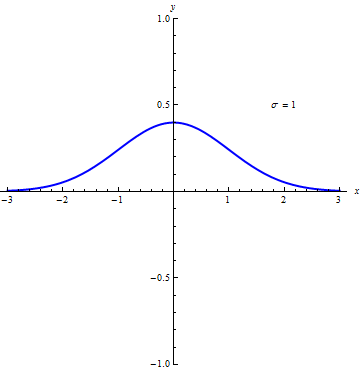
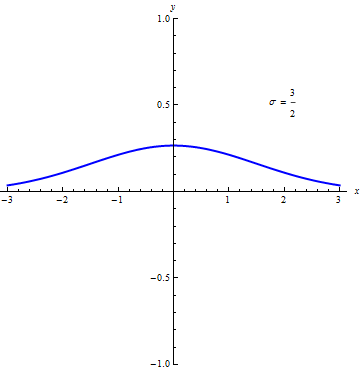


where  is the [mean](http://en.wikipedia.org/wiki/Mean),  is the [standard deviation](http://en.wikipedia.org/wiki/Standard_deviation), and .

If the [mean](http://en.wikipedia.org/wiki/Mean)  and the [standard deviation](http://en.wikipedia.org/wiki/Standard_deviation)  are such that  and , then we have the following bell-shaped curve:



Note: The [mean](http://en.wikipedia.org/wiki/Mean)  translates this curve left or right; while the [standard deviation](http://en.wikipedia.org/wiki/Standard_deviation) makes this curve narrower or wider. For instance, the following plots show how this bell-shaped curve changes as  changes. The plot on the left is for , the plot in the center is for , and the plot on the right is for :

Therefore, as  decreases, the bell-shaped curve shoots up (i.e. gets narrower). Similarly, as  increases, the bell-shaped curve flattens out (i.e. gets wider).

Moreover, this function has the following property:



where  is the [Error function](http://en.wikipedia.org/wiki/Error_function), , and .

Therefore, regardless of what the value is for  and , the "area under the curve" for the entire function is always 1 for this function. This is a desirable property that we can take advantage of in Probability.

a. Given the function for the Normal distribution:



Derive the  values that production [inflection points](http://en.wikipedia.org/wiki/Inflection_point) in the function above. In other words, using Calculus and Algebra, find the  values that make  for any value of  and .

b. C++11 added the [Error function](http://en.wikipedia.org/wiki/Error_function)  shown above to the header <cmath> as the function erf(). The following block of code shows how to use this [Error function](http://en.wikipedia.org/wiki/Error_function):

#include <iostream>

#include <cmath>

int main () {

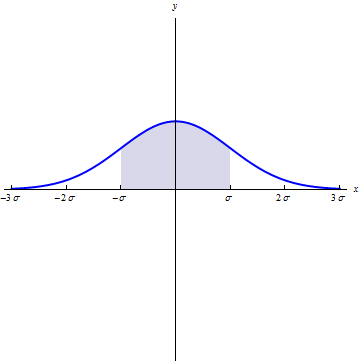
std::cout << erf( 1.0 ) << '\n'; // 0.842701

}

Use the indefinite integration above to calculate the exact value of the following definite integrals where . Exact value means keeping the square roots and reducing down to one  function. Note: . Also, use the C++11 erf() function to calculate the decimal values of those same definite integrals:

|  |  |  |
| --- | --- | --- |
| **Definite Integral** | **Exact Value** | **Decimal Value** |
|  |  |  |
|  |  |  |
|  |  |  |

c. For these bell-shaped curves, what is the percentage of the "area under the curve" that is within one standard deviation from the [mean](http://en.wikipedia.org/wiki/Mean)? In other words, if we fixed , what is the blue shaded area in this plot as a percentage of the overall "area under the curve":



d. For these bell-shaped curves, how many standard deviations  from the [mean](http://en.wikipedia.org/wiki/Mean) covers approximately  of the "area under the curve"?

2. Expectation Maximization using a Gaussian Mixture Model (50 Points).

OpenCV can perform [Expectation Maximization](http://en.wikipedia.org/wiki/Expectation-maximization_algorithm) using the [cv::EM](http://docs.opencv.org/2.4.4/modules/ml/doc/expectation_maximization.html) class. Here is an excerpt of the OpenCV code needed to perform Expectation Maximization to pull out two "Gaussians" (i.e. the [multivariate version of the Normal distribution](http://en.wikipedia.org/wiki/Multivariate_normal_distribution) we covered above in Written Problem 1):

int max\_number\_of\_iterations = 1024;

double threshold = 0.000001;

cv::TermCriteria termination\_criteria(

cv::TermCriteria::COUNT + cv::TermCriteria::EPS,

max\_number\_of\_iterations,

threshold

);

int number\_of\_clusters = 2;

cv::EM em(

number\_of\_clusters,

cv::EM::COV\_MAT\_GENERIC,

termination\_criteria

);

em.train( dataset );

auto means = em.getMat( "means" );

std::cout << "means:\n" << means << "\n\n";

auto covs = em.getMatVector( "covs" );

std::cout << "covs:\n" << covs << "\n\n";

The OpenCV class cv::TermCriteria is used to set up the termination criteria (i.e. when to stop the EM algorithm). This is same as the termination criteria that we have used to stop gradient descent in previous assignments. Similarly, we are stopping here after either 1024 iterations of EM or when the relative change in the likelihood logarithm is under the threshold 0.000001.

The OpenCV enumeration value cv::EM::COV\_MAT\_GENERIC ensures that we will receive "Gaussians" that are both scaled and rotated (both of these, vice just one or the other). In other words, this enumeration value ensures that we will receive a full [Covariance Matrix](http://en.wikipedia.org/wiki/Covariance_matrix) for each cluster it will try to fit. A [covariance matrix](http://en.wikipedia.org/wiki/Covariance_matrix) functions in much the same way as the [standard deviation](http://en.wikipedia.org/wiki/Standard_deviation) functioned in Written Problem 1 above.

In the code excerpt above, we are requesting that "Gaussians" be fitted to two clusters of data; therefore, after we train on a dataset, we should receive two [means](http://en.wikipedia.org/wiki/Mean) and two [covariance matrices](http://en.wikipedia.org/wiki/Covariance_matrix).

These multivariate "Gaussians" (i.e. the [Probability Density Function](http://en.wikipedia.org/wiki/Probability_density_function) of the [Multivariate Normal Distribution](http://en.wikipedia.org/wiki/Multivariate_normal_distribution)) have a similar function to that of the bell-shaped curve we saw in Written Problem 1 above:



where  in the exponent means matrix or vector [transpose](http://en.wikipedia.org/wiki/Transpose),  means the [determinant](http://en.wikipedia.org/wiki/Determinant),  is the input dimension such that , the [mean](http://en.wikipedia.org/wiki/Mean) , and the [covariance matrix](http://en.wikipedia.org/wiki/Covariance_matrix)  is an  matrix such that  where  is a  [parametric rotation matrix](http://en.wikipedia.org/wiki/Rotation_matrix) and  is a  [parametric scaling matrix](http://en.wikipedia.org/wiki/Scaling_matrix). As in Written Problem 1, the "area under the curve" (or "volume under the curve") is 1. Note: Here  is a variable, not the summation symbol. Since we are dealing with the [multivariate](http://en.wikipedia.org/wiki/Multivariate_Gaussian_distribution) case here, we use the uppercase Greek letter S (uppercase sigma): . When we were dealing with the [univariate](http://en.wikipedia.org/wiki/Normal_distribution) case in Written Problem 1, we used the lowercase Greek letter s (lowercase sigma): .

For instance, if we have , a [mean](http://en.wikipedia.org/wiki/Mean) , a [parametric rotation matrix](http://en.wikipedia.org/wiki/Rotation_matrix) that rotates the "Gaussian" by , and a [parametric scaling matrix](http://en.wikipedia.org/wiki/Scaling_matrix)  that scales the "Gaussian" by  in the  direction and  in the  direction prior to the rotation, then we would have the following preliminary calculations:



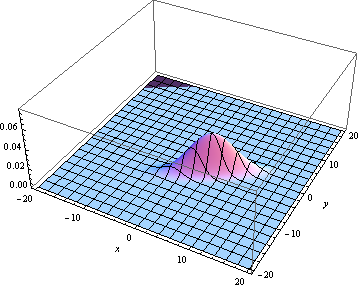
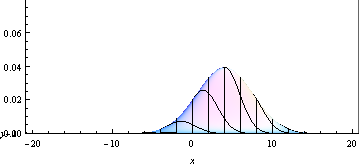
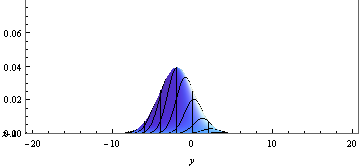
Then, we would have the following intermediate calculation:

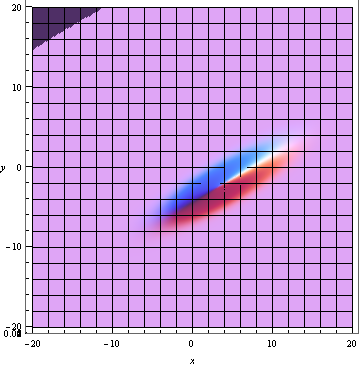
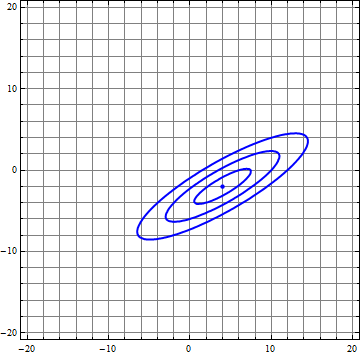


And finally, we would have:



If we graphed this, we would have the following plots (with the contour plot at the end):

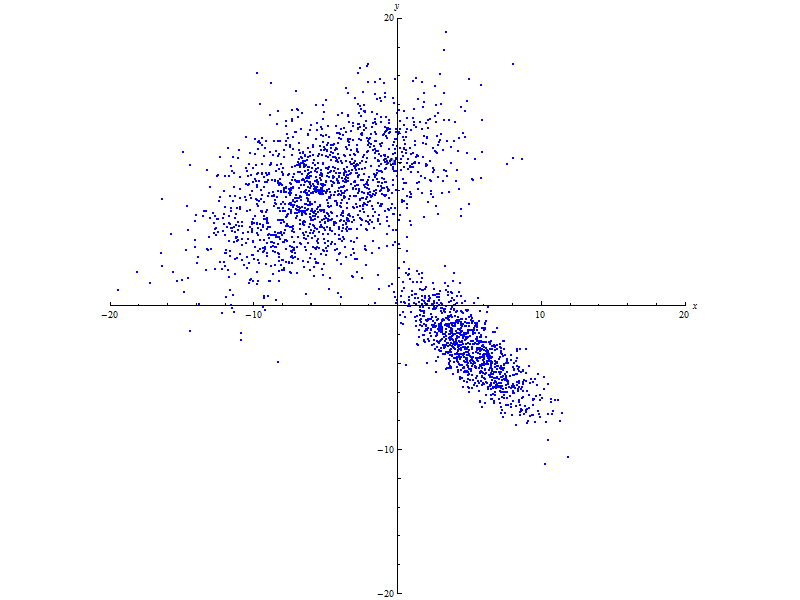
 

Note: The ellipses in the contour plot above are the first, second, and third "standard deviations" from the mean point at the center.

Note: Since we're dealing with ellipses, a  rotation for a scaled ellipse will look the same if the ellipse id rotated  in the opposite direction. In other words, a  rotation will look the same as a  rotation.

That's a lot of math! Thank goodness the OpenCV library does most of this for us.

The data from Assignment 1, Programming Problem 5 was the following:



This data is not separated into classes; however, it does appear that there are two classes of points in this dataset – a class for each cluster of data. These clusters also appear to be generated from a distribution similar to the [Multivariate Normal Distribution](http://en.wikipedia.org/wiki/Multivariate_normal_distribution) we just learned about. Why? Because these clusters are dense at their center, that density gradually falls off as we move away from their center, and the clusters are elliptical. Their center could be described by a [mean](http://en.wikipedia.org/wiki/Mean) (i.e. their [translation](http://en.wikipedia.org/wiki/Translation_(geometry))). Their elliptical nature could be described by a [covariance matrix](http://en.wikipedia.org/wiki/Covariance_matrix) (i.e. their [scaling](http://en.wikipedia.org/wiki/Scaling_(geometry)) and [rotation](http://en.wikipedia.org/wiki/Rotation_(geometry))).

We need the EM algorithm to return the [mean](http://en.wikipedia.org/wiki/Mean) and [covariance matrix](http://en.wikipedia.org/wiki/Covariance_matrix) of each of the clusters above so that we can separate those clusters into classes.

Once the EM algorithm returns the [mean](http://en.wikipedia.org/wiki/Mean) and [covariance matrix](http://en.wikipedia.org/wiki/Covariance_matrix) of each cluster, the following OpenCV code extracts the [elliptical](http://en.wikipedia.org/wiki/Ellipse) information from each cluster (Note: i is the index of each cluster [0 or 1]):

// Singular Value Decomposition

cv::Mat\_<double> U, W, Vt;

cv::SVDecomp( covs[ i ], W, U, Vt );

cv::Mat\_<double> center = means.row( i );

cv::Mat\_<double> semi\_major\_axis\_direction = Vt.row( 0 );

double semi\_major\_axis\_angle\_in\_radians = atan2(

semi\_major\_axis\_direction( 1 ),

semi\_major\_axis\_direction( 0 )

);

double semi\_major\_axis\_angle\_in\_degrees = (

semi\_major\_axis\_angle\_in\_radians \* degrees\_per\_radian

);

double semi\_major\_axis\_magnitude = sqrt(  W( 0 ) );

cv::Mat\_<double> semi\_minor\_axis\_direction = Vt.row( 1 );

double semi\_minor\_axis\_angle\_in\_radians = atan2(

semi\_minor\_axis\_direction( 1 ),

semi\_minor\_axis\_direction( 0 )

);

double semi\_minor\_axis\_angle\_in\_degrees = (

semi\_minor\_axis\_angle\_in\_radians \* degrees\_per\_radian

);

double semi\_minor\_axis\_magnitude = sqrt( W( 1 ) );

Compile and run the following code from [em.zip](http://www.cs.fsu.edu/~cop4601p/assignment/04/em.zip):

– [main.cpp](http://www.cs.fsu.edu/~cop4601p/assignment/04/em/main.cpp): The file containing the OpenCV calls for the EM algorithm.

– [data.cpp](http://www.cs.fsu.edu/~cop4601p/assignment/04/em/data.cpp): The dataset from Programming Problem 5 of Assignment 01.

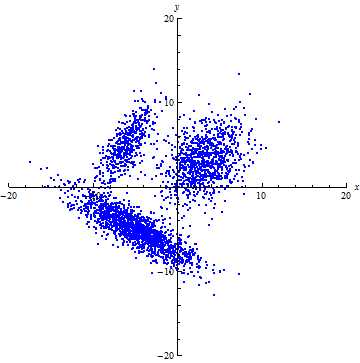
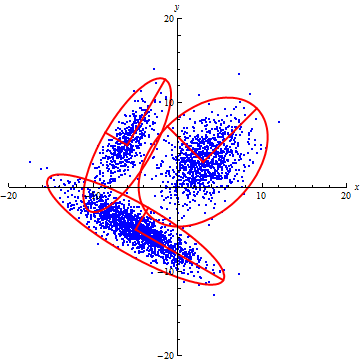
– [makefile](http://www.cs.fsu.edu/~cop4601p/assignment/04/em/makefile): The makefile for linprog4.cs.fsu.edu.

Using information about Cluster 0 from the output of the code above, the [Probability Density Function](http://en.wikipedia.org/wiki/Probability_density_function) of the [Bivariate Normal Distribution](http://en.wikipedia.org/wiki/Multivariate_normal_distribution) that generated that first cluster of data (Cluster 0) is approximately:

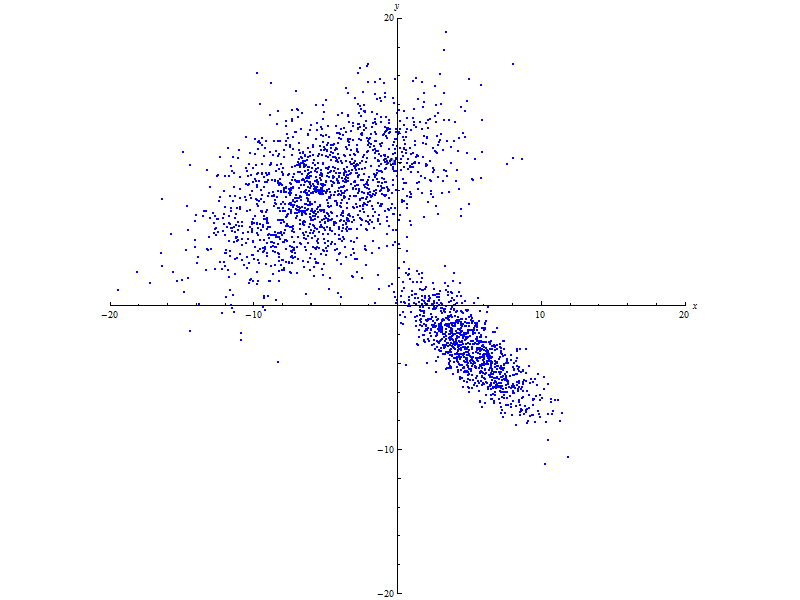


a. Using information about Cluster 1 from the output of the code above, calculate the approximate [Probability Density Function](http://en.wikipedia.org/wiki/Probability_density_function) of the [Bivariate Normal Distribution](http://en.wikipedia.org/wiki/Multivariate_normal_distribution) that generated that second cluster of data (Cluster 1).

b. Using the information about Ellipse 0 and Ellipse 1 from the output of the code above, draw the ellipse for each cluster that should contain 99.73% of the data. Draw these ellipses on the large picture below. Ensure that you include the line for semi-major axis and the semi-minor axis. Ensure that you pay close attention to the angle and the length of those semi-major and semi-minor axes. For example, given the initial plot of points on the left, you would produce the final plot of points with ellipses and axes on the right based on the output of the code above:

Draw the ellipses and axes on the following picture based on the output from the code above:



Keep in mind that this dataset did not contain class labels. This was an unlabeled dataset. We told the EM algorithm to find two clusters of points and it found two … without supervision … and the two clusters that the EM algorithm found happened to exactly match our expectations. Yay, Unsupervised Learning!

3. Neural Networks (100 Points).

Given the following Neural Network:

































where , , , and activation functions , , and  are step functions defined as follows:



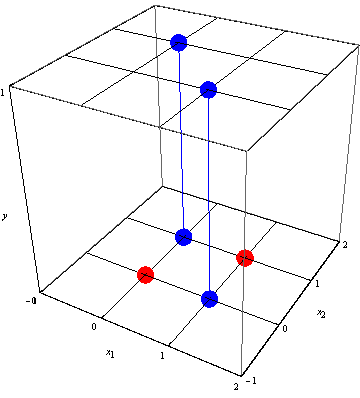
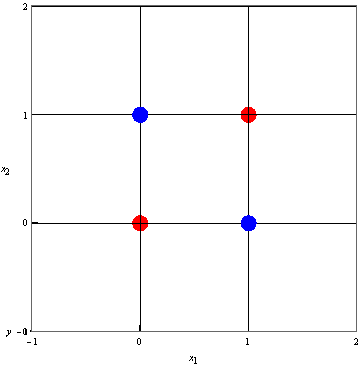
a. Place and orient the activation functions , , and  by hand and calculate the nine Neural Network parameters (i.e. , , , , , , , , and ) above needed to compute the XOR function. In other words, calculate the nine parameters needed to create this function:



HINT: It will be very helpful to plot the  points, draw the lines  and  on that plot, and shade each positive region (i.e. where  and ).

HINT: It will be very helpful to plot the  points, draw the line  on that plot, and shade the positive region (i.e. where ).

Note: The XOR function produces these graphs:

Remember that  is just a plane that we can think of in terms of the normal vector equation that we saw in Assignment 01:



Therefore, , , and . Note: We could also adjust the "slope" of that plane using the  as we saw in the Logistic Function problem from Assignment 03, but we won't need to do that here.

b. For the functions , , and above, calculate the following tables:

Note: For that last table, you may add additional  or  values as needed for various combinations of  and .

Complete the following programming problems on linprog4.cs.fsu.edu:

Download the ZIP file containing the directory structure and files for these programming problems: [assignment\_04.zip](http://www.cs.fsu.edu/~cop4601p/assignment/04/assignment_04.zip)

1. Cross Validation – Part 2 (200 Points):

Use either the "Cross Validation – Part 1" code you wrote for Assignment 03 or the following code:

– [main.cpp](http://www.cs.fsu.edu/~cop4601p/assignment/04/cross_validation_2/main.cpp): The file to be edited.

– [wdbc.data](http://www.cs.fsu.edu/~cop4601p/assignment/04/cross_validation_2/wdbc.data): The [Breast Cancer Wisconsin (Diagnostic) Data Set](http://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Diagnostic%29).

– [makefile](http://www.cs.fsu.edu/~cop4601p/assignment/04/cross_validation_2/makefile): The makefile for linprog4.cs.fsu.edu.

Do \_\_not\_\_ touch the Testing dataset that contains 5% of the malignant data and 5% of the benign data.

Program a -fold Stratified Cross Validation:

Using just the Training dataset, divide the training dataset into class datasets (for this problem, a "malignant" dataset and a "benign" dataset).

(Optional) Shuffle each class dataset.

Divide each class dataset into  equal sets. For this programming problem, let ; however, ensure that  is a variable that can change be easily changed.

THE START OF A FOLD OF CROSS VALIDATION.

Create a "Train" dataset and a "Validate" dataset. Copy  set from each class into the "Validate" dataset. Copy the remaining  sets from each class into the "Train" dataset.

(Required) Shuffle the "Train" dataset and then shuffle the "Validate" dataset that contains copies from each class dataset.

Choose a classifier from OpenCV's [Machine Learning Library (MLL)](http://docs.opencv.org/2.4.4/modules/ml/doc/ml.html) that interests you. Train a classifier using the "Train" dataset. Use the classifier's train() method (if available). Validate that classifier's performance using the "Validate" dataset. Use the classifier's predict() method (if available). Use std::cout to report the parameters that were used for the classifier and the performance of the classifier. For this programming problem, the format of the output is not important.

For performance, use "Overall Accuracy":



Store "Overall Accuracy" in a std::vector that is maintained throughout the life of the program.

THE END OF A FOLD OF CROSS VALIDATION.

From the sets that each class dataset has been divided into, use a different set for your "Validate" set … the next set of the  sets in each class. Use the remaining sets to create your "Train" set. Perform another fold of Cross Validation.

Repeat this  times until every set of each class dataset has been used in the "Validate" set. In other words, do -folds of this -fold Stratified Cross Validation.

Example: If , then the "malignant" class would be broken up into 3 subsets … let's call them malignant\_1, malignant\_2, malignant\_3. Similarly, we would have benign\_1, benign\_2, and benign\_3.

For the 1st fold, our "Validate" set would contain the data from malignant\_1 and benign\_1; while the "Train" set would contain the data from the rest of the sets.

For the 2nd fold, our "Validate" set would contain the data from malignant\_2 and benign\_2; while the "Train" set would contain the data from the rest of the sets.

For the 3rd (and final) fold, our "Validate" set would contain the data from malignant\_3 and benign\_3; while the "Train" set would contain the data from the rest of the sets.