COMP4702 Case Study

Jordan West
41184882

16 May 2012

A Novel Kernel Method For Clustering

Introduction

This report gives an overview of the author's attempt to replicate the results of the proposed clustering method from the paper *"A Novel Kernel Method For Clustering" [1]*. As the original paper does not provide complete details on the implementation of the proposed clustering algorithm (hereby referred to as the *new algorithm*), some assumptions are made in an attempt to replicate the algorithm. In addition, the result of K-means clustering on the same data sets is attempted for comparison to the original paper. The other existing clustering algorithms (SOM, Neural Gas, Ng-Jordan) tested in the original paper are not investigated in this report.

Method

Data Sets

The first step of the process to reproduce results involved tracking down and retrieving the datasets referenced in the original paper.

The data sets used in the original paper are the IRIS, Wisconsin Breast Cancer Database, Spam Database, and the Delta Set which are available at the URLs below at the time of writing (one of which had changed since the original report was published) with the exception of the Delta Set:

Dataset	URL
IRIS Data Set	ftp.ics.uci.edu/pub/machine-learning-databases/iris
Wisconsin Breast Cancer Database	ftp.ics.uci.edu/pub/machine-learning-databases/breast-cancer-wisconsin
Spam Database	ftp.ics.uci.edu/pub/machine-learning-databases/spambase

Table 1 - Data set sources

The URL specified in the original paper for the Delta set was unavailable. The Delta Set is arguably the most important dataset on which to test the new algorithm as it is particularly difficult to cluster due to its linear inseparability, while the new algorithm appeared to perform favourably in the original paper. To solve this problem, the author of this report generated a Delta Set similar to the original using a MATLAB script (See Figure 1 for resulting dataset). The script can be found in *Listing 1 – Delta Set generator* under *Appendix A – Matlab Code.*

The structure of the data sets are listed in Table 2 below.

Dataset	ID	Attributes	Class {Cluster Names}
IRIS	None	Columns 1-4	Column 5 {'Iris-setosa', 'Iris-versicolor', 'Iris-virginica'}
Wisconsin	Column 1	Columns 2-10	Column 11 {2, 4}
Spam	None	Columns 1-57	Column 58 {0, 1}
Delta Set	None	Columns 1-2	None

Table 2 - Structure of Data Sets

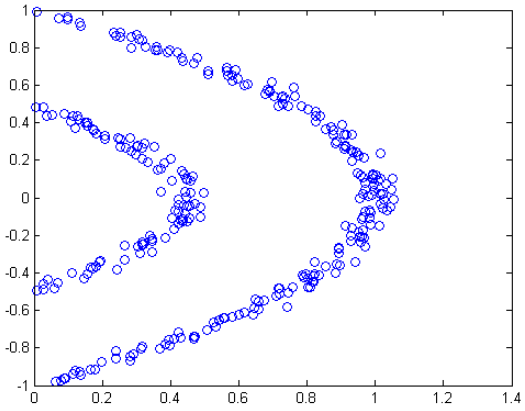

Figure 1 - Generated Delta Set

K-Means Clustering

K means clustering is a trivial clustering method which essentially involves randomly placing a number of 'Voronoi' points among the input space, then calculating which data points belong to each Voronoi point based on their distance. Once each data point has been attributed to a Voronoi point (the group of which are called a Voronoi set) the Voronoi point is moved to the mean location of all the points in its set. The process is then repeated for the new Voronoi points, until the location of the Voronoi points stabilises (converges).

K-Means clustering generally works quite well on distinctly separated, circular groups of data. However for data sets such at the Delta Set the algorithm fails as one cluster is surrounded by another, and the image is therefore linearly inseparable.

New Algorithm

Overview

The paper initially discusses two existing clustering algorithms, as the new algorithm is inspired by both. These algorithms are K-Means clustering (discussed above) and One-Class SVM. The idea behind the new algorithm is a hybrid which takes cues from both existing algorithms in its implementation. Therefore, it was important for the author of this report to understand both algorithms before attempting the new algorithm.

One-Class SVM

One-Class SVM is similar to Multi-Class SVM with the exception that it is trained to assume that all data points given are positive examples of a single class [2]. While a Multi-Class SVM guesses which class a particular observation belongs to, a One-Class SVM guesses whether or not a particular

observation belongs to a class. To produce such a model, the algorithm projects the data points into a feature space using a given kernel function, and a sphere of minimum radius is found to bound all of the given data points. When the SVM model is tested, any points contained within the sphere are considered to be positive examples of the class, while points falling outside of the sphere considered negative examples. Points falling close to the sphere may be included as positive examples, depending on the slack value specified. Finding the sphere of minimum radius is arguably the biggest challenge in implementing a one-class SVM, and was not attempted. Instead the LIBSVM library was utilised which contains such an implementation.

Installation of LIBSVM

To run the code shown in *Appendix A – Matlab Code, Listing 4 – New Algorithm* the LIBSVM library is required. This library is used for executing the One-Class SVM algorithm required for the new algorithm which is not available in MATLAB's SVM implementation. Installation is trivial and details can be found at the LIBSVM website [3]. Note that LIBSVM replaces MATLAB's built in "svmtrain" function with its own when it is included in the path.

Implementation of the New Algorithm

Although the new algorithm solves multi-class problems, it uses one-class SVM to do so. The resulting difference is that some data points will remain unclassified – that is unassigned to any cluster. This is favourable when looking to ignore outliers.

By reducing the original data set to several smaller sets of points, the one-class SVM algorithm is performed on each set of points in order to create a model for that cluster alone.

An overview of the algorithm taken from the original paper [1] is as below:

1. Initialize K Voronoi sets $V_k(\rho), k = 1, \dots, K$ using a subset of the l training points.
2. Train a one-class SVM for each $V_k(\rho)$
3. Update each $V_k(\rho)$
4. If no Voronoi set changes in Step 3, exit; otherwise, go to Step 2.

The interpretation of the above steps along with other details in the original paper yielded the following.

Step 1 involves initialising several sets, where each set contains some training points from the original data. The method of selection of these points is not explicitly mentioned in the original paper, however it is assumed that the points are assigned to the Voronoi sets via Euclidean distance in the feature space (since this is the same as determining which Voronoi region a point belongs to) in a similar manner to K-means, with the difference of being carried out in feature space after the application of the kernel function. There is also a constraint placed on the distance where the Euclidean distance from the data point to the Voronoi point is less than ρ. Therefore, it is expected that some points (hopefully outliers) will not belong to any Voronoi set.

Step 2 runs a one-class SVM on each Voronoi set – this step is relatively self-explanatory and is taken care of by LIBSVM, where the data points belonging to each $V_k(\rho)$ are sent to the one-class SVM algorithm.

Step 3 is quite ambiguous, and could be interpreted in several ways. There is no detail in the original paper as to which data points should be assigned to each $V_k(\rho)$. The author of this report interpreted step 3 as being one of two possibilities and both were attempted:

1. For each $V_k(\rho)$, given the trained one-class SVM model for those points – test said model on the complete dataset and note which data points are classified as positive examples. These data points then form the new $V_k(\rho)$.
2. Upon training of the one-class SVM, retrieve the support vectors which define the model and find the mean point in feature space, similar to K-Means. Update the Voronoi point to this point and update each $V_k(\rho)$ to contain the closest points again (with the distance constraint, ρ, as in Step 1).

Step 4 is presumed to mean that if all data points belong to the same, or don't belong to any $V_k(\rho)$ since the previous iteration – the algorithm has converged and is complete.

Results

K-Means Clustering

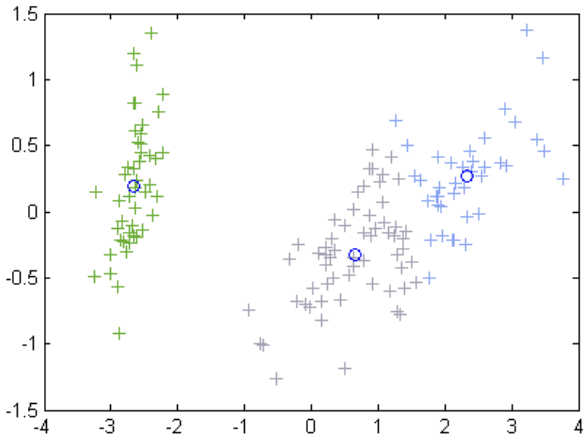

Figure 2 - K-Means clustering of the two principle components of the IRIS dataset (K=3)

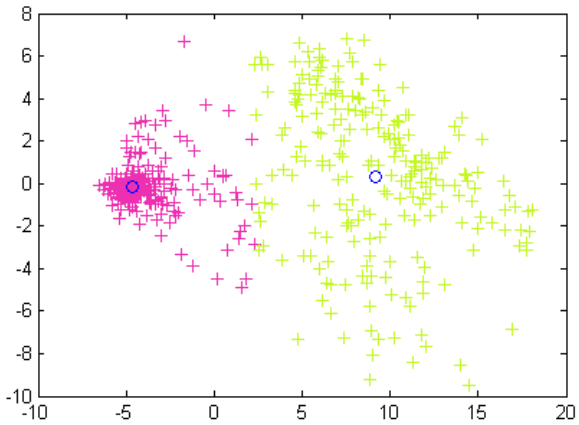

Figure 3 - K-Means clustering of the two principle components of the Wisconsin database (K=2)

Jordan West
41184882

	IRIS (2 dim. w/PCA)	Wisconsin (9 dim.)	Spam Database (57 dim.)
K-Means	132 (88.59%) 133.5 ± 0.5 (89.00%)	656 (96.19%) 656.5 ± 0.5 (96.1%)	3002* (65.25%) 1083 ± 153 (70.6%)

Table 3 - Clustering Results Obtained. The light gray values below the results indicate the results obtained in the original paper. *Note that the total number of instances in the database is currently 4601, while in the original paper the number of instances was only 1534.

The K-Means clustering algorithm implemented gave very similar results to those of the original paper. As in the original paper the results in Table 3 are from the K-Means algorithm run on the 2 principle components of the IRIS dataset, the 9 dimensions of the Wisconsin dataset, and the 57 dimensions of the Spam dataset. Note that Figure 3 shows the K-Means algorithm run on the two principle components of the Wisconsin dataset which yielded a very similar 655 correctly clustered instances.

The large difference in the results for the Spam dataset can most likely be attributed to the fact that the number of instances in the Spam dataset has increased from 1534 to 4601 since the original paper was published. As there is no indication of which instances have been added since, it is not possible to run the algorithm on the original 1534 instances. It is interesting, however that the error has increased, with only a 65.25% correct classification compared with 70.6% on the original smaller dataset.

Jordan West
41184882

New Algorithm

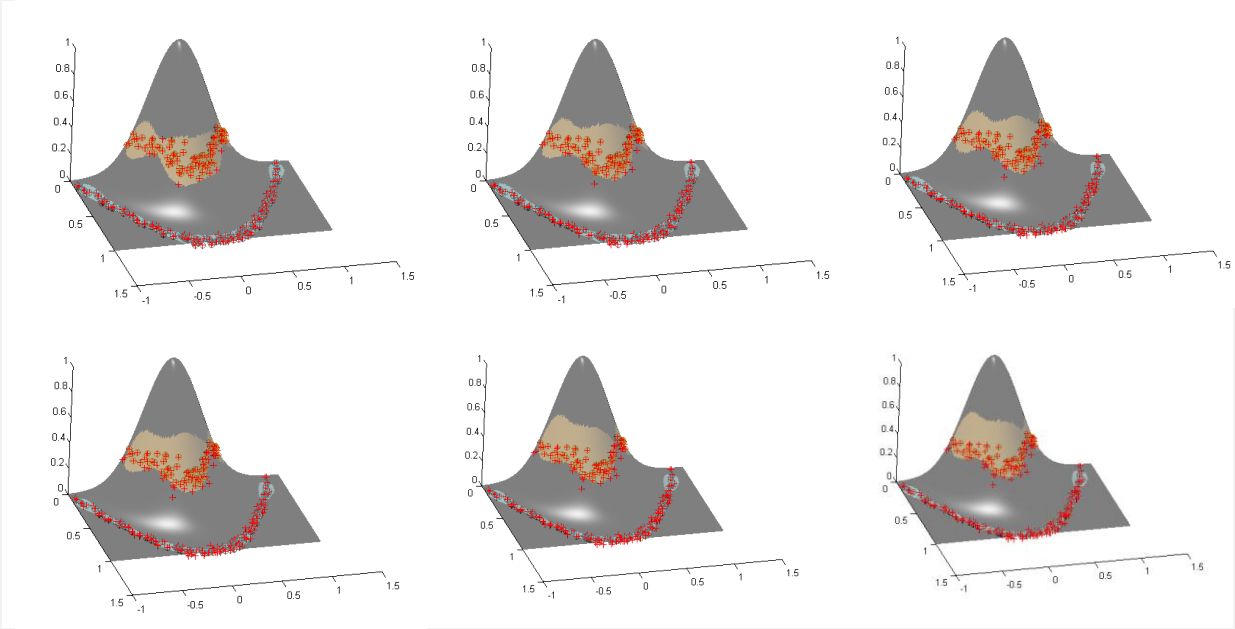

Figure 4 – New algorithm on the Delta set as plotted in feature space. Six iterations from top left to bottom right. Here the kernel function is a Gaussian with σ = 0.4 as specified in the original paper. Grey indicates input space which is not attributed to any class.

The implementation of the new algorithm was relatively unsuccessful. The most successful attempt is shown in Figure 4 above. To achieve this result, the initial Voronoi sets (Step 1 of the new algorithm) were manually and artificially set by linearly separating the clusters by the condition $V_1 = \{\xi_i \in \mathcal{F} \mid \xi_i < 0.1\}$ and $V_2 = \{\xi_i \in \mathcal{F} \mid \xi_i > 0.1\}$. This condition achieved 100% correct separation of the two sets in the initial iteration, however with each iteration the algorithm lost a few data points. Convergence is not shown as the algorithm took many iterations to converge; furthermore the convergence condition was achieved only when all Voronoi sets were empty.

No classification error values could be determined since convergence resulted in empty Voronoi sets and therefore no clusters.

Jordan West
41184882

Discussion

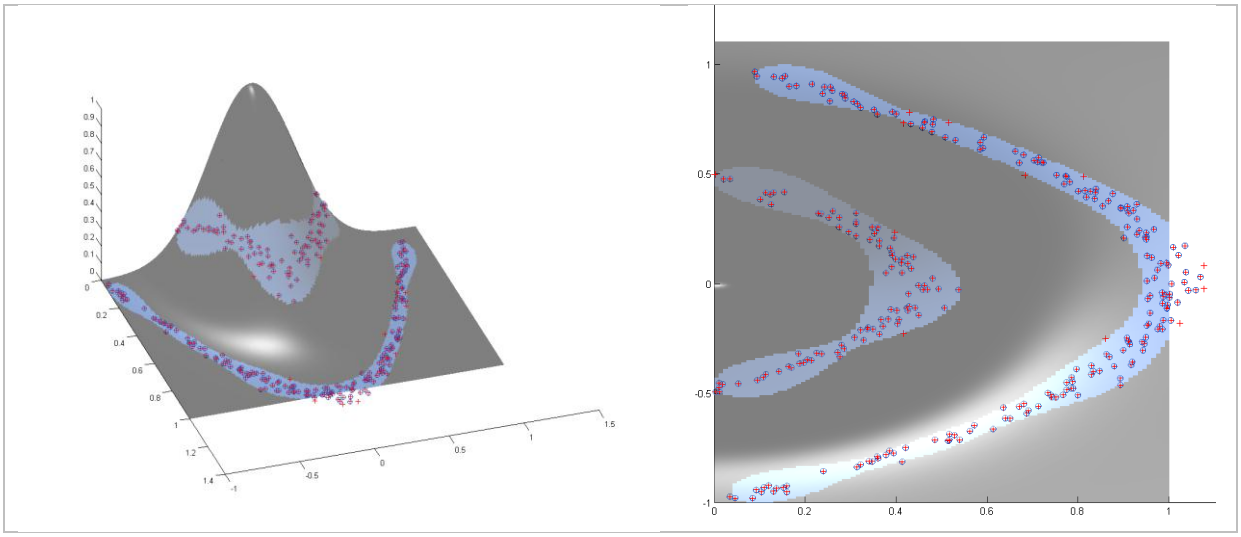

Figure 5 - Visualisation of first iteration of Delta Set (initial Voronoi sets manually specified with 100% accuracy) with the new algorithm in feature space and input space respectively. Note how tightly the regions encompass the data points – this is expected to be the cause of the failure of the algorithm.

Overall the results of the K-Means clustering on the relevant datasets matched quite closely to the original paper – with the exception of the Spam database due to the changes to the dataset. The implementation of the new algorithm however, was unsuccessful.

It is assumed that the author's second interpretation of Step 3 of the new algorithm was incorrect, as it yielded the same result as K-Means clustering – that is failure due to the data set being linearly inseparable. The first interpretation yielded much nicer results given some assistance (preloading the initial Voronoi sets with correct clusters) however still resulted in each cluster shrinking with each iteration until the Voronoi sets contained zero data points. The problem of shrinking with each iteration is expected to be the main cause of the failure of this implementation, as the cluster regions shown in Figure 5Figure 4 otherwise indicate a promising shape surrounding the data. Note however that the region does not provide any margin around the data points, the support vector lie exactly on the margin of the region. Following the initial iteration shown in the figure, the cluster regions shrink.

The author's best guess as to the shrinking nature of each SVM model is the lack of customisation of a slack variable in the one-class SVM algorithm. With little slack, data points just outside the cluster region were not classified as being part of the cluster, and therefore the cluster region was not given any opportunity to expand. Some borderline data points were also removed in each iteration, effectively shrinking the cluster region. As there are very few one-class SVM implementations in MATLAB, the author might have solved this by writing a custom one-class SVM algorithm implementation or modifying the algorithm used; however was unable to successfully do so. Other implementations of one-class SVM algorithms were difficult to use and provided little documentation, however may have yielded better results.

The MATLAB software was extremely helpful in visualising problems and discovering the state of the data at each step of the process. Particularly, a colour coded surface across the range of data (see Figure 5) helped visualise the model of each one-class SVM.

Conclusion

This report attempted to reproduce the results from the proposed new algorithm. Overall the implementation was near working; however limitations to the One-Class SVM algorithm were expected to cause the algorithm to fail by shrinking cluster regions with each iteration. Since the One-Class SVM algorithm used is open-source, future attempts might investigate the possibility of modifying the underlying SVM algorithm code to allow more flexibility in terms of adding a slack value to include bordering data points.

More positively, the results for K-Means on the same datasets were successfully replicated to a close margin.

Jordan West
41184882

References

[1] F. Camastra and A. Verri, "A novel kernel method for clustering," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 27, no. 5, pp. 801 –805, May 2005.

[2] M. Filippone, F. Camastra, F. Masulli, and S. Rovetta, "A survey of kernel and spectral methods for clustering," *Pattern Recognition*, vol. 41, no. 1, pp. 176–190, Jan. 2008.

[3] "LIBSVM -- A Library for Support Vector Machines." [Online]. Available: http://www.csie.ntu.edu.tw/~cjlin/libsvm/. [Accessed: 16-May-2012].

Appendix A – Matlab Code

Listing 1 – Delta Set generator

```
% COMP4702 Case Study
% Semester 1, 2012
%    Jordan West

% A Novel Kernel Method for Clustering
% Francesco Camastra, Member, IEEE, and
%       Alessandro Verri

% Delta Set Generator

Y1 = -1:0.01:1;
Y1r = Y1+(randn(1,201)*0.05);

Y2 = -0.5:0.01:0.5;
Y2r = Y2+(randn(1,101)*0.05);

A = (-Y1r.^2)+1;
Ar = A+(randn(1,201)*0.03);

B = 2*(-Y2r.^2)+0.45;
Br = B+(randn(1,101)*0.03);

% Combined dataset
X = [Ar Br];
Y = [Y1r Y2r];

D = [X.' Y.'];

% Randomly sort the dataset
D = D(randperm(size(D,1)),:);

% Clear out values out of range (due to the randomness)
D = D(D(:, 1) > 0, :);
D = D(abs(D(:, 2)) < 1, :);

hold off;
plot(D(:,1), D(:,2), 'o');

Dataset = dataset(D);

export(Dataset, 'file', 'delta.data','delimiter', ',');
```

Listing 2 – K-Means Clustering

```
% COMP4702 Case Study
% Semester 1, 2012
%    Jordan West

% A Novel Kernel Method for Clustering
% Francesco Camastra, Member, IEEE, and
%       Alessandro Verri

% K-Means Clustering

%% Load datasets
iris = dataset('File', 'datasets/iris.data', 'format', '%f%f%f%f%s',
'Delimiter', ',');
iris = set(iris, 'VarNames', {'A', 'B', 'C', 'D', 'Type'});

wisconsin = dataset('File', 'datasets/breast-cancer-wisconsin.data',
'format', '%d%d%d%d%d%d%d%d%d%d%s', 'Delimiter', ',');
wisconsin = set(wisconsin, 'VarNames', {'ID', 'A', 'B', 'C', 'D', 'E', 'F',
'G', 'H', 'I', 'Type'});

spam = dataset('File', 'datasets/spambase.data', 'format',
'%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f
%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%s', 'Delimiter', ',');
VNames = cell(1,58);
% Add labels to the attributes (unique values required so that the 'Type'
% name can be set)
for i = 1:57
    VNames(1,i) = strcat({'Attr'}, {int2str(i)});
end
VNames(1,58) = {'Type'};
spam = set(spam, 'VarNames', VNames);

%% K-Means Clustering

% == IRIS DATASET ==
%  attribs = iris(:,1:4);
%  data = iris(:,1:5);
%  K = 3;

% == WISCONSIN DATASET ==
% attribs = wisconsin(:,2:10);
% data = wisconsin(:,2:11);
% K = 2;

% == SPAMBASE DATASET ==
 attribs = spam(:,1:57);
 data = spam(:,1:58);
 K = 2;

% Perform principle component analysis on iris dataset to reduce the n
% dimensions down to 2 principle components
[COEFF, SCORE] = princomp(double(attribs));

plot(SCORE(:,1), SCORE(:,2), 'x');

%D = SCORE(:,1:4);         % 2-Dimensional K-means using 2 principle comps
D = double(attribs);       % N-Dimensional K-means
```

```
% Find cluster centres
C = kmeansj(D, K);

%% Classify data points then compare to actual class
[dimRows, dimCols] = size(data);
classes = zeros(dimRows,1);
for i=1:dimRows
    x_t = D(i,:);

    % Find the nearest cluster centre
    [Z, I] = min(pdist2(C, x_t));

    classes(i) = I;
end
d_kmeans = horzcat(data, dataset(classes, 'VarNames', {'Class'}))

classification_error(d_kmeans.Type, d_kmeans.Class)
```

Listing 3 – Classification Error Utility

```
% COMP4702 Case Study
% Semester 1, 2012
%    Jordan West

% A Novel Kernel Method for Clustering
% Francesco Camastra, Member, IEEE, and
%       Alessandro Verri

% Utility for calculating classification error

function instances = classification_error(classnames, clusternumbers)
% Returns a matrix which indicates how two different naming schemes for
% clusters could be interpreted as correct/incorrect instances. Can be used
% for calculating error in clusters

    [dimRows, dimCols] = size(classnames);

    % NOTE: The behaviour of the 'unique' function may change in R2012a?
    % This code was only tested in R2011b
    discoveredClasses = unique(classnames);

    discoveredClusters = unique(clusternumbers)

    classCount = size(discoveredClasses, 1);
    clusterCount = size(discoveredClusters, 1);

    instances = zeros(classCount, clusterCount);

    for class = 1:classCount
        for cluster = 1:clusterCount
            % Find number of instances where the specified class and
            % cluster are the same
            A = strcmp(discoveredClasses(class), classnames) == 1;
            B = clusternumbers == discoveredClusters(cluster);

            instances(class, cluster) = sum(A == B == 1);
```


Jordan West
41184882

```
        end
    end
end
```

Listing 4 – New Algorithm

```
% COMP4702 Case Study
% Semester 1, 2012
%    Jordan West

% A Novel Kernel Method for Clustering
% Francesco Camastra, Member, IEEE, and
%       Alessandro Verri

% Interpretation of method from above paper performed on the Delta Set

%% Load datasets

deltaset = dataset('File', 'datasets/delta.data', 'format', '%f%f',
'Delimiter', ',');
deltaset = set(deltaset, 'VarNames', {'X','Y'});

%% Set up algorithm
sigma = 0.4

% Number of clusters
K = 2;

deltaX = double(deltaset(:,1));
deltaY = double(deltaset(:,2));
deltaG = exp(-(abs(deltaX.^2 + deltaY.^2))/(sigma^2));

% Create a grid across the data range for plotting the kernel fn
[XX, YY] = ndgrid(0:0.01:1, -1:0.01:1.1);
ZZ = exp(-((abs(XX.^2 + YY.^2))/(sigma^2)));

hold off;
plot3(deltaX, deltaY, deltaG, 'r+');
hold on;
camlight left; lighting phong;

[dimRows, dimCols] = size(deltaX);
lbls = [1:1:dimRows].';

% First column is a label for each observation
% Fourth column is for storing the cluster id attributed to the observation
fullset = [lbls deltaX deltaY zeros(dimRows,1)];

% Pick two random points
init_points = (floor(rand(2,1)*dimRows));
%init_points = [1; 106];

voronoi_points = zeros(K,3);
for i = 1:K
    voronoi_points(i,:) = [deltaX(init_points(i)) deltaY(init_points(i))
deltaG(init_points(i))];
```


Jordan West
41184882

```
end

% Maximum euclidean distance from feature to voronoi point in feature space
MaxR = 0.4;

%% === ALGORITHM STEP 1 ===
% Initialize voronoi sets for clustering

for t = 1:length(deltaX)
    % find nearest voronoi points for each point in dataset
    x_t = [fullset(t, 2) fullset(t, 3) deltaG(t)];
    [C, I] = min(pdist2(voronoi_points, x_t));
    % Assign feature to nearest voronoi point cluster
%     if(C < MaxR)
%         fullset(t, 4) = I;
%     end

    % For testing, use a known hyperplane to slice the data in feature
    % space. The below clusters the Delta Set with 100% accuracy when
    % using a guassian kernel with sigma = 0.4
    if(deltaG(t) > 0.05)
        fullset(t, 4) = 1;
    else
        fullset(t, 4) = 2;
    end
end
clusterColors = rand(K, 3);

%% Iterate through steps 2 and 3
for iteration = 1:6

    hold off;
    figure(iteration);
    % Draw all the
    plot3(deltaX, deltaY, deltaG, 'r+');
    camorbit(110, 30);
    hold on;

    [surfRows, surfCols] = size(XX);
    surfColors = zeros(surfRows, surfCols, 3);

    % Train for each K
    newset = fullset;
    % Assume all points are unclassified first
    newset(:,4) = zeros(size(fullset,1), 1);
    for i = 1:K
        % Grab just the data points that belong to this Voronoi set
        V_current = fullset(fullset(:,4) == i,:);

        % === ALGORITHM STEP 2 ===
        % Train one-class SVM
        SVM = svmtrain(V_current(:,1), V_current(:,2:3), '-s 2 -t 2 -g 10 -
n 0.01');
        TEST = svmpredict(fullset(:,1), fullset(:,2:3), SVM);

        % Color of the current cluster
        Colors = clusterColors(i,:);
        % Train on the grid to get a visualisation of the cluster
        % regions
```

```
        for col = 1:surfCols
            CLIST = svmpredict(XX(:,col), [XX(:,col) YY(:,col)], SVM);
            for row = 1:surfRows
                if(CLIST(row) == 1)
                    surfColors(row,col,:) = Colors;
                end
            end
        end

        % Classified observations
        new_V = fullset(TEST == 1,:);

        for p = 1:size(new_V,1)
            if(fullset(p,4) > 0)
                plot3(new V(p,2), new V(p,3), deltaG(new V(p,1)), 'o',
'Color', Colors);
            end
        end

        % === ALGORITHM STEP 3 ===
        % Update new values into the voronoi set for the next iteration
        for z = 1:size(new_V,1)
            newset(newset(:,1) == new_V(z,1), 4) = i;
        end
    end

    surf(XX, YY, ZZ, surfColors, 'EdgeColor', 'none', 'FaceAlpha', 0.5);
    camlight left; lighting phong;
    fullset = newset;
end
```


Jordan West
41184882