# COMP6036: Advanced Machine Learning
# An investigation into DBSCAN

Ashley J. Robinson
ajr2g10@ecs.soton.ac.uk

School of Electronics and Computer Science
University of Southampton

12th March, 2014

**Abstract**

*The paper chosen for the research report is entitled* ***A density-based algorithm for discovering clusters in large spatial databases with noise*** *which introduces the algorithm DBSCAN. DBSCAN is used for clustering sparse spatial databases using data point density. The algorithm performs well at this but has some shortcomings when applied to tasks which hold clusters of different densities. Only one priori, cluster density, is required but the algorithm is very sensitive to this and can be difficult to tune in high dimensional space. DBSCAN is compared against other common cluster implementations using a machine learning toolkit for Python before more recent extensions are considered.*

## 1 Motivation for Algorithm

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is an unsupervised application of machine learning introduced by Ester et al. (1996). Intended to address Spatial Database Systems (SDBS) which can be produced from naturally occurring geometric and geographical data or engineering applications such a layout for integrated circuit design (Güting, 1994). It has three main objectives. To minimise the required domain knowledge needed to set input parameters, have the capability to discover clusters of arbitrary shapes and to perform well on large spatial databases.

At the time of creation the algorithms was compared to a recent development called CALARANS (Raymond and Jiawei, 1994) which is an extension of CLARA (Clustering LARge Applications) (Kaufman and Rousseeuw, 1990). Both algorithms are intended for use on large databases but CLARANS uses random noise to improve performance. Apart from traditional clustering algorithms where first principles are introduced, such a K-means, DBSCAN was a breakthrough in terms of a density approach to datasets.

## 2 Technical Explanation

DBSCAN uses cluster density to classify data. It is intuitive to build a community of data points by attempting to draw a path of connectivity between points. This leads to the first input parameter to the algorithm, $\epsilon$, which is a threshold for the distance that DBSCAN is permitted to move yet remain in the same cluster. Equation (1), adapted from Ester et al. (1996), is the basic function used to determine membership by thresholding; euclidean distance is used in this case but the measure of distance can be replaced with a Manhattan norm to scale down computational overheads therefore favouring large datasets (Krause, 1986). Only two input patterns are compared at once and all belong to the set of training data, $\mathbf{x}$, containing $n$ input patterns.

$$N(\mathbf{p},\mathbf{q},\epsilon) = \begin{cases} 1, & ||\mathbf{p}-\mathbf{q}|| \leq \epsilon \\ 0, & ||\mathbf{p}-\mathbf{q}|| > \epsilon \end{cases} \quad where\ \mathbf{q},\mathbf{p} \in \mathbf{x} \tag{1}$$

This approach is simple for compact clusters but when noise is introduced the algorithm will identify a few points as a whole cluster. This is possibly down to a badly tuned value for $\epsilon$ but can be negated by introducing a second threshold to set the minimum number of members a cluster can have however this is unnecessary to perform basic DBSCAN clustering. Equation (2) takes the sum of connected data points attributed to a single point which used for the comparison in Equation (3). The parameter, $\lambda$, if successfully applied to Equation (3) decides when to build a cluster around the point.

$$C(\mathbf{x}_i,\mathbf{x},\epsilon) = \sum_{\substack{j=1 \\ j\neq i}}^{n} N(\mathbf{x}_i,\mathbf{x}_j,\epsilon) \tag{2}$$

$$1 \leq \lambda \leq C(\mathbf{p},\epsilon)\ where\ \lambda \in \mathbb{N} \tag{3}$$

A low dimensional representation of the constraints used in DBSCAN is held in Figure 1 which contains seven points labelled **A** to **H**. A string of points are density connected in the centre of the diagram, two points on the left and is **G** is unconnected. Increasing $\epsilon$ will merge these three possible clusters. Equation (4) describes how possible values of $\lambda$ can change the clustering of Figure 1.

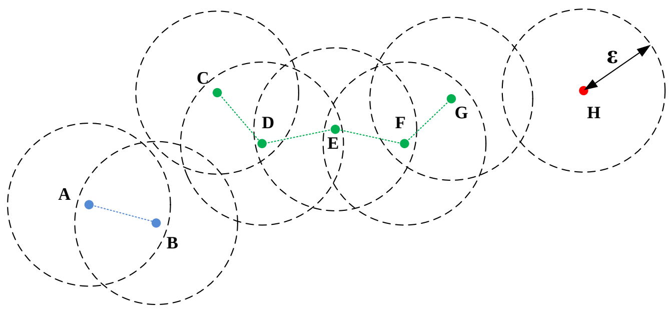


FIGURE 1: 2D application of DBSCAN.

$$\#Clusters = \begin{cases} 0\ if\ \lambda \geq 6 \\ 1\ if\ \lambda = 5 \\ 2\ if\ 2 \leq \lambda \leq 5 \\ 3\ if\ \lambda = 1 \end{cases} \tag{4}$$

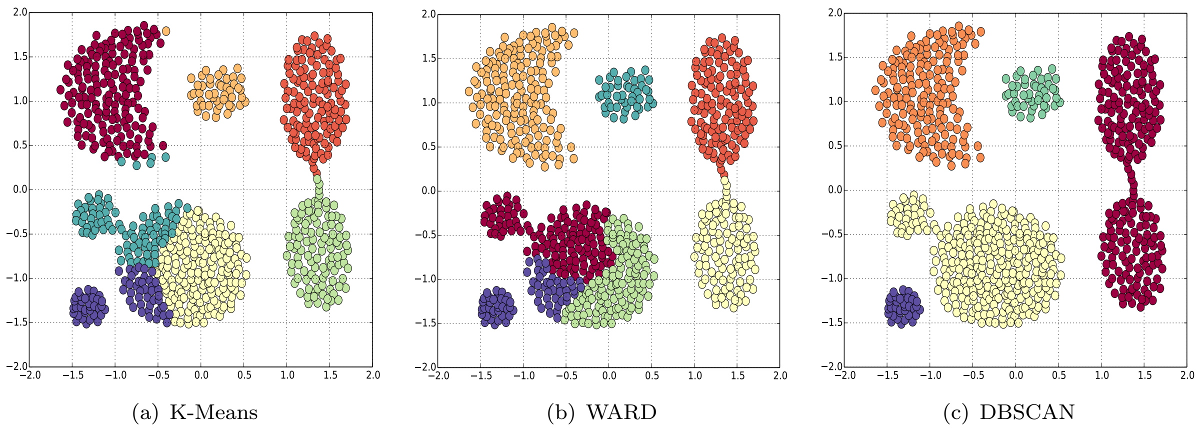
It is clear there is a lot of scope for optimisation in practice. A list containing all data points at the start of the algorithm can be used to check off points which have been found to belong to a cluster. This means for an input space containing $\alpha$ clusters that can be perfectly segmented by DBSCAN requires only $\alpha$ calls of Equation (2). Formally a set of unclassified vectors, $\mathbf{b} \in \mathbf{x}$, can be defined and passed to Equation (2) requiring much fewer evaluations.

## 3 Performance

The worst case performance of the basic algorithm is determined by the number of end cluster found by DBSCAN compared to the number of input patterns. If all the input patterns combined with a massive value of $\epsilon$ produce single cluster classification it is possible that $O(n)$ holds; this is based on the number of times Equation (1) has to be computed. Equation (5) shows how as the number of clusters increases the algorithm tends towards $O(n^2)$. The first input pattern requires $n-1$ calls of Equation (1) regardless but in the worst case is has no reachable neighbours. The second requires $n-2$ as one pattern has already been identified as solitary yet again has no reachable neighbours. If this continues so that the last pattern is known to be solitary then Equation (5) shows how this geometric progression places an lower bound on performances. A well tuned value of $\epsilon$ will allow the algorithm to complete in polynomial time.

$$\sum_{i=1}^{n-1} i = \frac{n}{2}(n-1) \approx O(n^2) \tag{5}$$

DBSCAN suffers from sensitivity to input parameters and clustering in high dimensional space Han et al. (2001). When attempting to cluster regions of different density it is clear that this is a major problem for the algorithm in certain applications. The graphs in Figure 2 are different clustering algorithms applied to a shape dataset taken from Gionis et al. (2005) which is chosen to empathise some less obvious shortcomings. Chosen for comparison is K-Means because it is arguably the most well-known clustering algorithm and WARD because of its performance on this dataset when compared against other available algorithms. Implementations[1] were taken from a machine learning toolbox for Python (scikit learn, 2013). The dataset contains seven clusters but in two cases there are *bridges* between the clusters. On the far right K-Means and WARD correctly divides the data where DBSCAN incorrectly groups both clusters together; shown in Figures 2(a), 2(b) and 2(c) respectively. DBSCAN deals well with the remaining data where the other two algorithms fail due to its ability to negate relatively tight clusters. These sorts of issues are the reason for $\epsilon$ sensitivity.



(a) K-Means (b) WARD (c) DBSCAN

FIGURE 2: A comparison against DBSCAN.

[1]Code in Appendix A

The algorithm exhibits variation in performance as per the standard bias-variance dilemma. Figure 3 shows how the error varies for with $\epsilon$ when applied to data used in Figure 2. Objectively DBSCAN doesn't perform well on this dataset missing two clusters completely however this is a crafted corner-stone case intended to trap algorithms and is not particularly sparse.

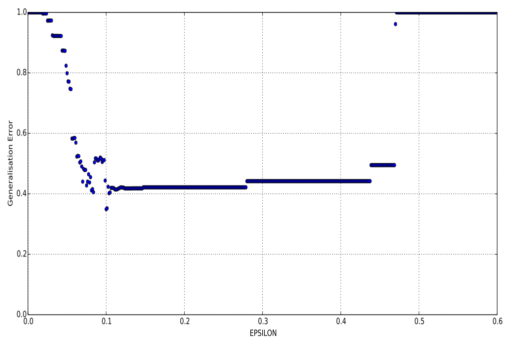


Figure 3: Generalisation error while varying the $\epsilon$ input parameter.

## 4 Extensions

The sensitivity to input paramters has tried to Duan et al. (2007) considered the difficulty to select a value for $\epsilon$ and adapted the algorithm to LDBSCAN (Local-DBSCAN).

## 5 Conclusions

DBSCAN performs well on spatial database and can be optimised to be extremely efficient. As per most unsupervised clustering algorithms this is particularly appealing for good results with using simple methods. There are downsides to algorithm

## References

Lian Duan, Lida Xu, Feng Guo, Jun Lee, and Baopin Yan. A local-density based spatial clustering algorithm with noise. In *Informations Systems Journal*, volume 32, pages 978–986, Now 2007.

Martin Ester, Hans peter Kriegel, Jrg S, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *2nd International Conference on Knowledge Discovery and Data Mining (KDD-96)*, pages 226–231. AAAI Press, 1996.

Aristides Gionis, Heikki Mannila, and Panayiotis Tsaparas. Clustering aggregation. In *In Proceedings of the 21st International Conference on Data Engineering (ICDE)*, pages 341–352, 2005. Hosted by Speech and Image Processing Unit, University of Eastern Finland, `http://cs.joensuu.fi/sipu/datasets/` (Visited on 27/02/2014).

Ralf Hartmut Güting. An introduction to spatial database systems. *Very Large Database Journal (VLDB J.)*, 3(4):357–399, 1994.

Jiawei Han, Micheline Kamber, and Anthony Tung. Spatial clustering methods in data mining: A survey. In *Geographic Data Mining and Knowledge Discovery, Research Monographs in GIS*. Taylor and Francis, 2001.

Leonard Kaufman and Peter J. Rousseeuw. *Finding groups in data: an introduction to cluster analysis*. John Wiley and Sons, New York, 1990.

E.F. Krause. *Taxicab Geometry: An Adventure in Non-Euclidean Geometry*. Dover Books on Mathematics Series. Dover Publications, 1986. ISBN 9780486252025.

Ng T. Raymond and Han Jiawei. Efficient and effective clustering methods for spatial data mining. In *20th Very Large Database Conference, Santiago, Chile, 1994*, pages 144–155, 1994.

scikit learn. machine learning in python. `http://scikit-learn.org/stable/index.html`, 2013. (Visited on 27/02/2014).

## A Code Listings

```
import numpy as np
from sklearn.cluster import KMeans
from sklearn import metrics
from sklearn.datasets.samples_generator import make_blobs
from sklearn.preprocessing import StandardScaler
import pylab as pl
pl.ion()

##############################################################################
# Load data
f = open('Datasets/Aggregation.txt','r')
X = []
labels_true = []
for line in f:
  x = line.split('\t')[0]
  y = line.split('\t')[1]
  label = line.split('\t')[0]
  label = label.replace('\n','')
  X.append([x,y])
  labels_true.append(label)
X = StandardScaler().fit_transform(X)

##############################################################################
# K-means Parameters
num = 7

##############################################################################
# Compute DBSCAN
db = KMeans(init='random',n_clusters=num).fit(X)
labels = db.labels_
# Number of clusters in labels, ignoring noise if present.
n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)

##############################################################################
# Plot result
# Black removed and is used for noise instead.
#fig = fig + 1
pl.figure(1)
unique_labels = set(labels)
colors = pl.cm.Spectral(np.linspace(0, 1, len(unique_labels)))
for k, col in zip(unique_labels, colors):
    if k == -1:
        # Black used for noise.
        col = 'k'
        markersize = 6
    class_members = [index[0] for index in np.argwhere(labels == k)]
    for index in class_members:
        x = X[index]
        pl.plot(x[0], x[1], 'o', markerfacecolor=col,
                markeredgecolor='k',markersize=10)
pl.grid()
pl.show()
raw_input("Press Enter to continue...")
```

Listing 1: K-Means clustering.

```
import numpy as np
from sklearn.cluster import Ward
from sklearn import metrics
from sklearn.datasets.samples_generator import make_blobs
from sklearn.preprocessing import StandardScaler
import pylab as pl
pl.ion()

##############################################################################
# Load data
f = open('Datasets/Aggregation.txt','r')
X = []
labels_true = []
for line in f:
  x = line.split('\t')[0]
  y = line.split('\t')[1]
  label = line.split('\t')[0]
  label = label.replace('\n','')
  X.append([x,y])
  labels_true.append(label)
X = StandardScaler().fit_transform(X)

##############################################################################
# K-means Parameters
num = 7

##############################################################################
# Compute WARD
db = Ward(n_clusters=num,connectivity=None).fit(X)
labels = db.labels_
# Number of clusters in labels, ignoring noise if present.
n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)

##############################################################################
# Plot result
# Black removed and is used for noise instead.
#fig = fig + 1
pl.figure(1)
unique_labels = set(labels)
colors = pl.cm.Spectral(np.linspace(0, 1, len(unique_labels)))
for k, col in zip(unique_labels, colors):
    if k == -1:
        # Black used for noise.
        col = 'k'
        markersize = 6
    class_members = [index[0] for index in np.argwhere(labels == k)]
    for index in class_members:
        x = X[index]
        pl.plot(x[0], x[1], 'o', markerfacecolor=col,
                markeredgecolor='k',markersize=10)
pl.grid()
pl.show()
raw_input("Press Enter to continue...")
```

Listing 2: WARD Clustering.

```python
import numpy as np
from sklearn.cluster import DBSCAN
from sklearn import metrics
from sklearn.datasets.samples_generator import make_blobs
from sklearn.preprocessing import StandardScaler
import pylab as pl
pl.ion()

##############################################################################
# Load data
f = open('Datasets/Aggregation.txt','r')
X = []
labels_true = []
for line in f:
  x = line.split('\t')[0]
  y = line.split('\t')[1]
  label = line.split('\t')[0]
  label = label.replace('\n','')
  X.append([x,y])
  labels_true.append(label)
X = StandardScaler().fit_transform(X)

##############################################################################
# DBSCAN Parameters
eps = 0.2
min_samples = 1
##############################################################################
# Compute DBSCAN
db = DBSCAN(eps, min_samples).fit(X)
core_samples = db.core_sample_indices_
labels = db.labels_
# Number of clusters in labels, ignoring noise if present.
n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)

##############################################################################
# Plot result
pl.figure(1)
unique_labels = set(labels)
colors = pl.cm.Spectral(np.linspace(0, 1, len(unique_labels)))
for k, col in zip(unique_labels, colors):
    if k == -1:
        # Black used for noise.
        col = 'k'
        markersize = 6
    class_members = [index[0] for index in np.argwhere(labels == k)]
    cluster_core_samples = [index for index in core_samples
                            if labels[index] == k]
    for index in class_members:
        x = X[index]
        if index in core_samples and k != -1:
            markersize = 14
        else:
            markersize = 6
        pl.plot(x[0], x[1], 'o', markerfacecolor=col,
                  markeredgecolor='k', markersize=10)
#pl.title('DBSCAN - %d Clusters' % n_clusters_)
pl.grid()
pl.show()
raw_input("Press Enter to continue...")
```

Listing 3: DBSCAN Clustering.

```
import numpy as np
from sklearn.cluster import DBSCAN
from sklearn import metrics
from sklearn.datasets.samples_generator import make_blobs
from sklearn.preprocessing import StandardScaler
import pylab as pl
pl.ion()

##############################################################################
# Load data
f = open('Datasets/Aggregation.txt','r')
X = []
labels_true = []
for line in f:
  x = line.split('\t')[0]
  y = line.split('\t')[1]
  label = line.split('\t')[0]
  label = label.replace('\n','')
  X.append([x,y])
  labels_true.append(label)
X = StandardScaler().fit_transform(X)
##############################################################################
# DBSCAN Parameters
eps = 0.47
min_samples = 1
score = []
para = []
for test in range(1,480):
  eps = float(test)/float(800)
  ##############################################################################
  # Compute DBSCAN
  db = DBSCAN(eps, min_samples).fit(X)
  core_samples = db.core_sample_indices_
  labels = db.labels_
  # Number of clusters in labels, ignoring noise if present.
  n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)
  score.append(metrics.adjusted_mutual_info_score(labels_true, labels))
  para.append(eps)
  if(score[test-1] < 0):
    score[test-1] = 0;
  score[test-1] = 1 - (score[test-1]*10)
  print("%0.3f: Score: %0.3f" % (eps, score[test-1]))
pl.plot(para,score, '.', markeredgecolor='k', markersize=10)
#coefs = np.lib.polyfit(para, score, 4) #4
#fit_y = np.lib.polyval(coefs, para) #5
#pl.plot(para, fit_y, 'b--') #6
pl.ylim([0,1])
pl.grid()
pl.xlabel('EPSILON')
pl.ylabel('Generalisation Error')
raw_input("Press Enter to continue...")
```

Listing 4: Tuning DBSCAN.