Identifying Outliers in the latest Quasar Catalog

Arman Irani
Bourns College of Engineering
University of California, Riverside
Riverside, CA
airan002@ucr.edu

Reza Monadi
Department of Physics and Astronomy, University of California, Riverside
Riverside, CA
rmon003@ucr.edu

Hasin Us Sami
Bourns College of Engineering
University of California, Riverside
Riverside, CA
hsami003@ucr.edu

Terrance Kuo
Bourns College of Engineering
University of California, Riverside
Riverside, CA
tkuo013@ucr.edu

ABSTRACT
We investigated the latest quasar catalog (SDSS DR16Q) including more than 750,000 objects with several physical measurements. Focusing on the color of quasars as our feature space, we implemented unsupervised machine learning to objectively recognize the population of outlier quasars. More specifically, we used Density-Based Spatial Clustering of Applications with Noise (DBSCAN), agglomerative clustering, and isolation forest techniques. Our results show that there is a cluster of 256 quasars which show different properties. They have inconsistent spectra compared to the main population, very red colors. Moreover, they have very high isolation forest scores consistent with the fact that they are an outlier population of quasars.
KEYWORDS
clustering, outliers, quasars, DBSCAN, Agglomerative Clustering, Isolation Forest, quasars spectra, color distribution

1 Introduction

Quasars reside in the galaxies with a central super massive black hole. The black hole devours surrounding materials and converts them to electromagnetic radiation in different wavelengths. Recording the energy in a specific wavelength (spectroscopy) gives us a 1D image which is known as quasar's spectrum. The quasar spectra have many features in common so that one can distinguish them from the spectra of other astronomical objects.

We can also measure the overall flux of quasars in bigger wavelength ranges (photometry) and obtain their colors by comparing these filtered fluxes. For example, we receive more (less) flux in longer wavelengths and less (more) flux in shorter wavelengths when a quasar is red (blue). The reason why quasars possess

a variety of colors is related to their physical conditions and probably to their evolutionary phase. For example, when they are young and more surrounded by gas and dust, they tend to have redder colors. On the other hand, when they go through their later phases they expel the gas and dust and the quasar reveals itself with bluer colors.

2 Focus of our Research

The goal of our project is to identify outliers among quasars using the three outlier detection techniques: DBSCAN, Isolation Forest and Agglomerative Clustering. The results would be evaluated by analyzing the deviation of outlier quasars' spectra and colors. Also we test the consistency of the results of different algorithms.

0.1 Related Research

Outlier quasars have been of interest and they have studied in [1], for example, with the help of self organized maps. However, relying on just one method is not a solid way to detect outliers.
However, we adopted the idea of using a variety of algorithms and checking their consistency from this paper [2].

In relation to the methods used in our project, several outlier detection techniques in terms of their efficiency and limitation have been studied. In paper [3], drawbacks of traditional DBSCAN algorithm have been discussed and a more improved version of this algorithm has been introduced and applied into image segmentation.Though DBSCAN is one of the most efficient algorithms in clustering and detecting outliers, the main drawback of this algorithm is choosing the parameters epsilon and minimum sample point. By trial and error method, an optimum value is chosen which works best for that specific dataset. If any changes in the dataset is made, those parameters need to be changed as well. That's why paper [3] has introduced a technique/formula to calculate these parameters by combining the concept of K-nearest neighbor. A curve is plotted to examine the behavior of the distance from a point to its k^{th} nearest neighbor. When points are placed on the x-axis based on their increasing distance from k^{th} nearest neighbor and the distance is placed on the y-axis, the sharp modification/increase in the curve resembles the optimum value for epsilon. Though this technique has been applied on image dataset for segmentation purpose, it can be extended to the quasar dataset in our project for outlier detection purpose. But the limitation of the paper is the technique they developed to calculate optimum for min_sample points based on the size of the pixel image and grey scale value (256), can be used for image dataset only and thus cannot be used for our project.
In paper [4], another drawback of the DBSCAN algorithm has been brought into light. Since the dataset is very large nowadays, DBSCAN is susceptible to high computation cost and memory allocation problems. To overcome this issue, this paper has introduced an efficient DBSCAN algorithm using Map-Reduce where a 3-stage approach is adopted: data partitioning, local clustering and global merging. Each partition is independently processed and thus computation time is significantly reduced. Each partition is assumed small enough to fit on memory and so memory allocation issues are also overcome. But it has not shed light in any technique to determine optimum value for epsilon or minimum sample point which is very much important for our project.

0.2 Data Preprocessing and Overview

This section explains preprocessing steps performed here and also gives a general overview of the structure and interrelation of features in our prepared data-set.

0.2.1 Defining the feature space

SDSS DR16Q is a table with more than 750,000 rows and 180 columns corresponding to the measurements. Since we are going to build up a multidimensional color space we need to pick up a pair of measured fluxes in two specific filters and calculate the color by comparing their fluxes. Each flux measurement comes with an uncertainty which is used for filtering out noisy data. We removed objects if at least one flux had S/N<3. Also we removed objects with a redshift warning flag.

SDSS DR16Q has several measured fluxes in various wavelength bands and we can pick up any pair and define a color. However, some of the pairs are less reliable and more susceptible to noise and contamination. Therefore we limited our colors to seven combinations, namely: r-z, r-W1, z-W1, r-W2, i-W2, and z-W2. We also needed to limit the redshift range of these quasars to ensure all important emission line components (like Lyman-alpha) are captured by SDSS filters. So we filtered out all quasars but those with 2.7<redshift<3 which led us to ∼180,000 high quality and reliable data points with 7 color features. Figure1 shows the pair-plot of these 7 colors and gives us a general overview of the whole data-set.

0.2.2 : Pair-plot for 7 different colors used in this study.

0.2.3 Dimensionality reduction

0.2.4 To visualize our 7D color space we used t-distributed Stochastic Neighbor Embedding (tSNE) to map the feature space to a projected 2D space. tSNE preserves the local similarity of data points in high dimensions when they are mapped to lower dimensions by minimizing probability distributions of distances in the original and mapped spaces. The tuning knob in tSNE is the perplexity parameter which is basically the number of nearest neighbors which the algorithm takes care more in the mapping procedure. We tried several perplexities and realized that a perplexity=20 gives us a better map (See Figure 3).

0.3 Outlier Detection Techniques

This section introduces the different types of outlier analysis techniques implemented in this paper.

0.3.1 DBSCAN (Density-Based Spatial Clustering of Applications with Noise)

Definition

DBSCAN is one of the most efficient outlier detection techniques. It is able to show satisfactory performance due to some of its unique properties:

1. ability to form clusters of arbitrary shape
2. not susceptible to outliers

3. no. of clusters need not be prespecified

For DBSCAN algorithm, there are two parameters that are needed to be specified:
Epsilon: It defines a certain region around a data point where a minimum number of data points should be presented in order to form a cluster. An optimal value for this parameter is chosen based on the distribution of data so that there would be less possibility for an outlier to be detected as part of a cluster or a normal data point to be detected as an outlier.
Minimum Sample Points: It defines the minimum number of data points that need to be present within epsilon in order to be qualified as a cluster. The more concentrated the data points, the larger value of the minimum sample point is chosen.

Implementation:

The following steps are followed to develop the algorithm in Python.

Algorithm 1 Building Clusters

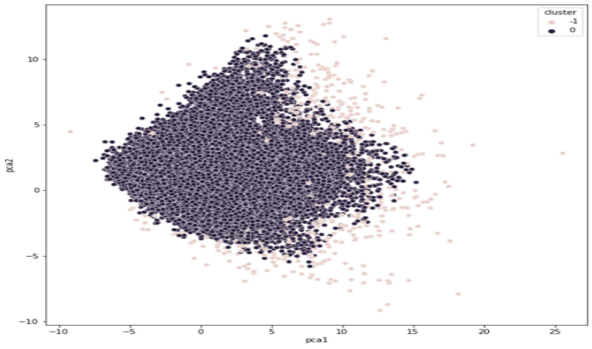
```
procedure DBSCAN(data,eps,minpts)
    System Initialization
    for i in dataset do
        Neighborslist = [ ]
        if not visited before then
            mark it as visited
            find no. of neighbors within eps
            if no. of neighbors < minpts then
                Do nothing
            else
                assign a cluster number to the point
                Neighborslist.append(neighbors)
                for i in neighbors do
                    if not visited before then
                        mark it as a visited point
                        find all other points within its eps neighborhood
                        if no. of neighbors ≥ minpts then
                            Neighborslist.append(new neighbors)
                    if a cluster number has not been assigned to this point then
                        assign the cluster number it belongs to
```

The above steps are repeated for all the data points in the dataset. Those points that do not belong to any clusters are detected as anomalies/outliers.

Performance Analysis

Before applying a self-implemented algorithm, DBSCAN built-in function in python has been experimented on the preprocessed dataset just to verify how accurate this detection technique would be in the case of our quasar dataset. Optimum values for DBSCAN parameters**(eps=1.5,min_points=10)** have been chosen using a trial and error method that best isolates the normal data points and outliers. Due to the quite large size of our dataset, the algorithm has been developed by taking memory limitation and run time into consideration. For each data point, K-D tree technique has been used for finding the points within the range of epsilon. To compare the result, Principal Component Analysis (PCA) function has been used to map the data points into a 2D plan for the purpose of visualization.

(a)



(b)

0.3.2 : Panel (a) 2-D Plot with clusters for proposed DBSCAN algorithm. Panel(b) 2-D Plot with clusters for built-in function

Both the DB built-in function and its implementation from scratch give the same result with the same number of outliers.

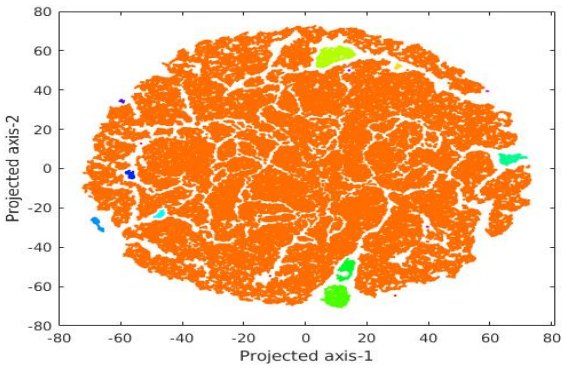


Figure3: Visualizing clusters using t-SNE

0.3.3 Isolation Forest

Implementation

The Isolation Forest technique for anomaly detection is employed as a direct counterpart to density and distance based outlier detection methods. An Isolation Forest is an unsupervised machine learning algorithm that makes the assumption that outliers are susceptible to isolation, meaning that anomalies satisfy the following two properties:

1. Anomaly instances occur few and far in-between.
2. Anomaly instances consist of feature values that are drastically different than normal instances.

The construction of many decision trees by random selection allows this method to be scaled to large datasets with high dimensionality.

The design of the Isolation Forest[5] used in this project was modelled off the following pseudocode.

Algorithm 2 : $iTree(X')$

Inputs: X' - input data
Output: an iTree

if X' cannot be divided **then**
  return $exNode\{Size \leftarrow |X'|\}$
else
  let Q be a list of attributes in X'
  randomly select an attribute $q \in Q$
  randomly select a split point p between the max and min values of attribute q in X'
  $X_l \leftarrow filter(X', q < p)$
  $X_r \leftarrow filter(X', q \geq p)$
  return $inNode\{Left \leftarrow iTree(X_l),$
    $Right \leftarrow iTree(X_r),$
    $SplitAtt \leftarrow q,$
    $SplitValue \leftarrow p\}$
end if

Figure4: Isolation Tree Pseudocode

The Isolation Tree is created recursively, and is applied on a random sub-selection of attributes. Once the path counter is equal to the maximum height limit, or only one data element remains, the tree will be returned to the Isolation Forest function and added to the ensemble. The algorithm for the Isolation Forest is described in the pseudocode below.

Algorithm 1 : $iForest(X, t, \psi)$

Inputs: X - input data, t - number of trees, ψ - subsampling size
Output: a set of t *iTrees*

Initialize *Forest*
for $i = 1$ to t **do**
  $X' \leftarrow sample(X, \psi)$
  $Forest \leftarrow Forest \cup iTree(X')$
end for
return *Forest*

Figure5: Isolation Forest Pseudocode

Parameter Selection

The following parameters are user defined:

X - Input data
t - Number of trees
ψ - Subsampling size
hlim - Tree height limit

The selection of these parameters were based on Liu et. al. 2012 as well empirically chosen. The number of trees controls the ensemble size[5] and it was found that the average path length converged to a maximum of 18 when the number of trees exceeded 20. So the number of trees was chosen to be 20, to limit unnecessary processing runtime.

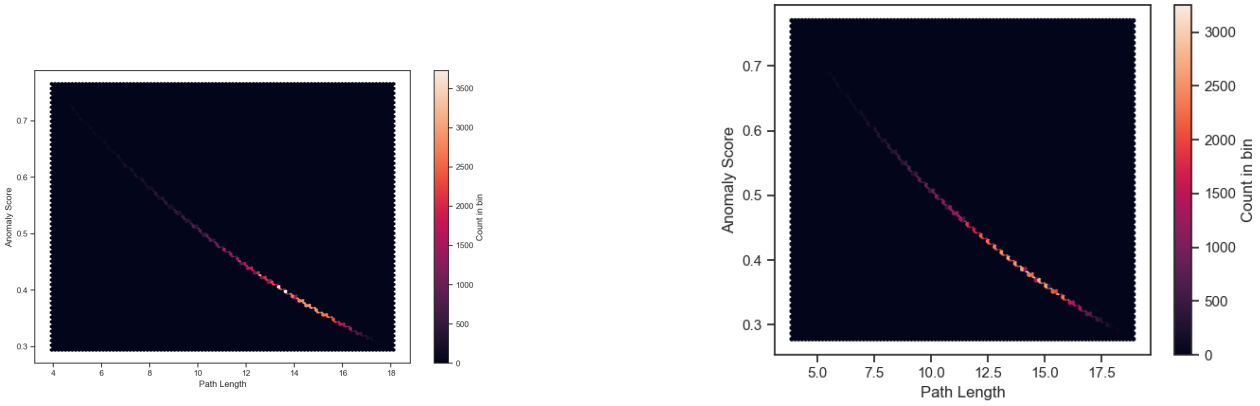


Figure 6: Comparison of anomaly distribution (Top: *t = 20, hlim= 20,* Bottom*: t = 50, hlim = 150)*

Isolation Forest algorithms incorporate subsampling because smaller selections contain less normal instances which cause less interference and allow outliers to be isolated more effectively. According to Liu et.al. 2012, it was observed that a subsampling size equal to 256 or 512 was the convergence limit of memory and training time optimization. Performance of outlier detection also remained stable across subsampling sizes larger than 256. Therefore for memory optimization purposes of our large datasets we chose a subsampling size of 256.

Performance

The algorithm itself is memory optimized due to its random subspace selection techniques. However, a space-time trade-off was made during implementation to use dictionaries as the data structure, to represent our decision tree versus using lists. Lists in python are memory optimized, however are not optimized for lookup. Whereas Dictionaries provide fast lookup but poor memory use. Dictionaries were chosen for this project's implementation since the dataset had a relatively average size, not necessitating memory optimization. Future work may include improving this aspect of the implementation.

Using an 8 core 32GB RAM computer, the construction of the Isolation Forest for the 180,000 instances in the preprocessed quasar dataset took approximately 8 seconds, and the process of recursively discovering path lengths for each instance and evaluating the anomaly score took 4.2 minutes. The performance of this algorithm increased linearly with the max height limit, as well as the number of trees.

Isolation Forest Evaluation

Scoring Isolation Forest is calculated by the following equation:

$$s(x, \psi) = 2^{-\frac{E(h(x))}{c(\psi)}},$$

Where $E(h(x))$ is the expected path length for each instance, and $c(\psi)$ is the average path length across all decision trees. This will output an anomaly score between 0 and 1. If the score for every instance returns 0.5, this means there are no anomalies within the dataset. If the scores returned are below 0.5, then those can be considered normal instances. If the scores are significantly above 0.5 and close to 1, then those instances can be considered outliers.

The path length algorithm is based on the following pseudocode implementation.

Algorithm 3 : $PathLength(x, T, hlim, e)$

Inputs : x - an instance, T - an iTree, $hlim$ - height limit, e - current path length; to be initialized to zero when first called

Output: path length of x

```
if T is an external node or e ≥ hlim then
    return e + c(T.size) {c(.) is defined in Equation 1}
end if
a ← T.splitAtt
if x_a < T.splitValue then
    return PathLength(x, T.left, hlim, e + 1)
else {x_a ≥ T.splitValue}
    return PathLength(x, T.right, hlim, e + 1)
end if
```

Figure 7: Path Length pseudocode for evaluation

0.3.4 Agglomerative Clustering

Definition

Agglomerative hierarchical clustering is the most common type of hierarchical clustering used to group objects in clusters based on their similarity. It is also known as *AGNES* (*Agglomerative Nesting*). The algorithm starts by treating each object as a singleton cluster. Next, pairs of clusters are successively merged until all clusters have been merged into one big cluster containing all objects. The result is a tree-based representation of the objects which we call a dendrogram.
Agglomerative clustering works in a “bottom-up" manner. Each item is considered its own cluster and at each step of the algorithm, the two clusters that are the most similar are combined into a new bigger cluster (nodes). This procedure is iterated until all points are members of just one single big cluster which we color into different clusters when we look at the result.

Implementation

Making the result has several steps:

1. Find the similarity or dissimilarity between every pair of objects in the data set. In this step, you calculate the distance between objects using the pdist function.

2. Group the objects into a binary, hierarchical cluster tree. In this step, one links pairs of objects that are in close proximity using the linkage function. The linkage function uses the distance information generated in step 1 to determine the proximity of objects to each other. As objects are paired into

binary clusters, the newly formed clusters are grouped into larger clusters until a hierarchical tree is formed.

3. Determine where to cut the hierarchical tree into clusters. In this step, one uses the cluster function to prune branches off the bottom of the hierarchical tree, and assign all the objects below each cut to a single cluster. This creates a partition of the data. The cluster function can create these clusters by detecting natural groupings in the hierarchical tree or by cutting off the hierarchical tree at an arbitrary point.

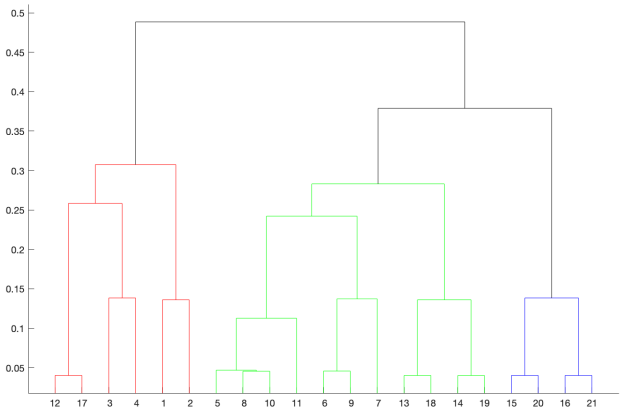
The 4 pieces of information he user needs to provide is :

1. Data as a numeric matrix with rows representing observations (individuals) and columns representing variables.
2. Distance method type. No input means euclidean. Can also put parameters for certain distance types.
3. The number of clusters wanted
4. Linkage type

The most important portion of the implementation is how the linkage of the data points are going to happen. Using different distance and linkage types are going to have different results when we put them into clusters. Examples of this is using complete, or maximum linkage, will cause us to get more compact clusters. Single or minimum linkage will give us more far reaching results. Average will give us a balanced result as it is getting the average distance between elements in cluster 1 and cluster 2.
Distance is also important because different distance calculations can result in different results for our clusters.

Performance

The algorithm can take a certain amount of data before it starts to break due to space limitations. Testing has noted that around 40k entries is when the algorithm begins to take a long amount of time. To remedy this, we can use the pca() function to reduce the dimensions of our data. The way pca() works reduces our data into a 21x21 matrix, which we reduce further into fewer columns for our clusters.



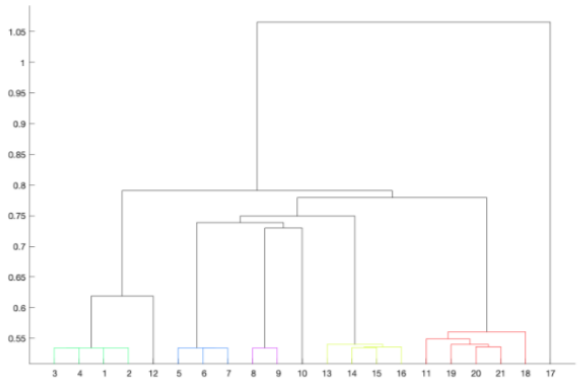


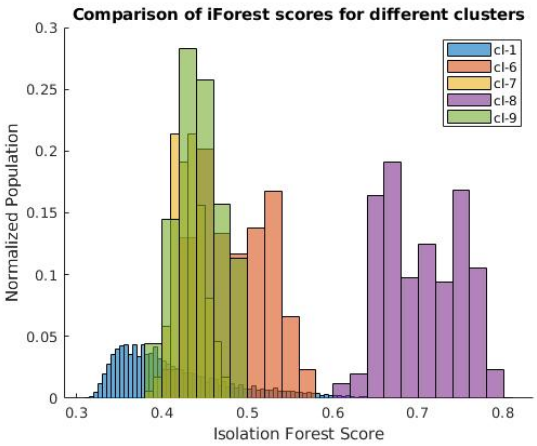
Figure8: Top panel shows the result of our clustering using “average" linkage, “chebyshev" distance, and 3 clusters
Bottom panel plots the data split between 7 clusters using “average" linkage and “euclidean" distance.

An issue using different linkages and measuring methods is that they have different results at times, making it inferior compared to DBSCAN.

0.4 Evaluation

0.4.1 Distribution of Isolation Forest Scores

Using the clusters found in the DBSCAN implementation, we ran the Isolation Forest algorithm within those clusters to evaluate the accuracy of our discovered clusters. It should be noted that we performed isolation forest on the original data before dimensionality reduction by tSNE. The rationale is that we are aware of the possible artifact clusters caused by tSNE. Therefore, we used Isolation Forest to measure the authenticity of our clusters found by DBSCAN on the tSNE mapped space. The results were promising and showed that both our Isolation Forest and DBSCAN implementation were able to find and evaluate similar instances to be outliers (see Figure9).



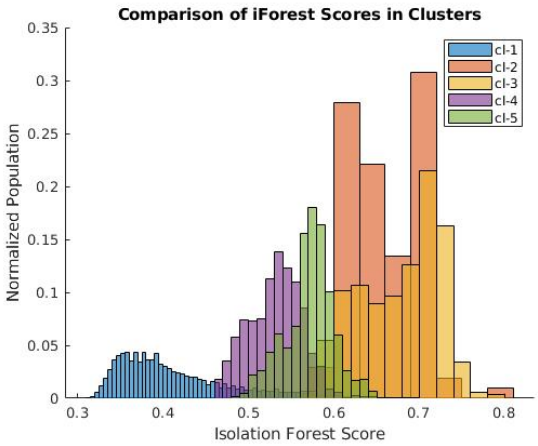


Figure9: Isolation forest anomaly score distribution for different clusters found by DBSCAN compared with the main cluster (blue).

0.4.2 Median Spectra of clusters

As it is stated before, quasars have similar spectra, generally. However, the environment that a quasar is living in (central regions of its hosting galaxy) and/or any special physical conditions (like strong winds emanating from quasars) can alter the shape of their spectra. Therefore, the most reliable way to compare quasars with each other is looking at their spectra. To do this, we first downloaded the spectra of our reduced data set from the SDSS database. Then we normalized each flux and took care of the redshift effect on the observed spectra by converting the observed wavelengths to the rest frame wavelengths of quasars. Finally, we augmented the spectra of all quasars in each cluster and obtained the median spectrum for each cluster (see Figure10). Median spectrum for cluster 1 is obtained from the majority of inlier objects. However, our outlier clusters are showing some deviation from that. Specially, cluster 8 has a very different median spectrum.

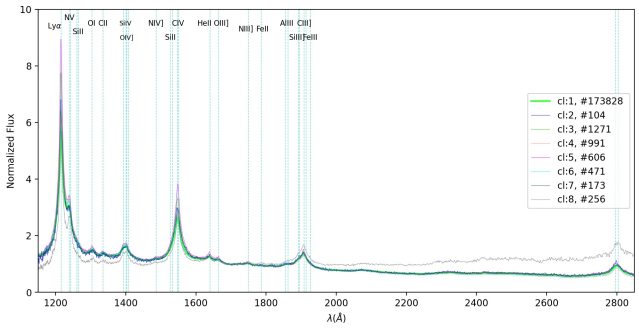


Figure10: Median spectra of quasars in each cluster.

0.4.3 Color distribution of clusters

It is useful to investigate physical properties within each cluster and look for any variation among them. The first thing that we can do here is looking at the distribution of quasars' color within each cluster.

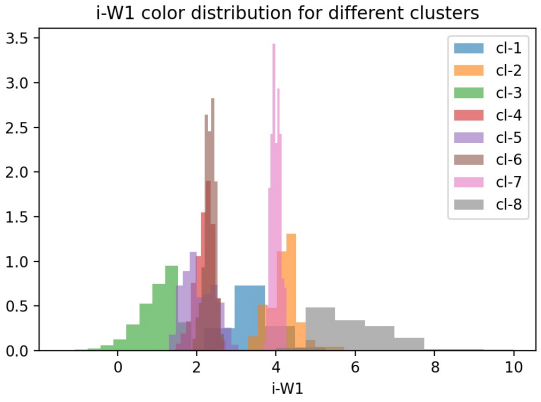


Figure11: Color distribution within various clusters of quasars.

Figure11 shows these color distributions for some of the significant clusters we found by DBSCAN. This plot confirms this fact that our clusters are really different from each other. More importantly, it also shows that cluster 8 has a very red color since higher i-W1 values correspond to redder colors.

0.5 Conclusion

In our project, we used three outlier detection techniques to detect outliers among quasars. We implemented DBSCAN and Isolation Forest from scratch but used MATLAB for agglomerative clustering and tSNE dimensionality reduction. Outliers obtained from each of the techniques have been evaluated in terms of their median spectra and compared with that of the normal quasar points. We found out that the median spectra of outlier quasars deviate by a significant amount from that of regular quasars which is consistent with their isolation from normal quasars in the 2D maped feature space. So it can be concluded that the detection technique has been quite successful in isolating the actual outliers. This work has displayed a potential to successfully contribute in further research on different classes of outliers and analysis of their properties. This also can be a good starting point for further detailed investigations of these outlier quasars, individually.

ACKNOWLEDGMENTS
RM thanks Frederick Hamann and Marie Wingyee Lau for their useful comments and discussions.

REFERENCES

1.H. Meusinger , P. Schalldach , R.-D. Scholz , A. in der Au , M. Newholm , A. de Hoon , and B. Kaminsky, A&A 541, A77 (2012), "Unusual quasars from the Sloan Digital Sky Survey selected by means of Kohonen self-organising maps"
2. Itamar Reis, Michael Rotman, Dovi Poznanski, J. Xavier Prochaska, Lior Wolf, "Effectively using unsupervised machine learning in next generation astronomical surveys", arXiv:1911.066823v2 [astro-ph.IM]
3.Suresh Kurumalla , P Srinivasa Rao, "K-Nearest Neighbor Based DBSCAN Clustering Algorithm for Image Segmentation" Journal of Theoretical and Applied Information Technology . Vol.92. No.2,2016
4. He, Y., Tan, H., Luo, W., Feng, S., & Fan, J. (2013). "MR-DBSCAN: a scalable MapReduce-based DBSCAN algorithm for heavily skewed dat". Frontiers of Computer Science, 8(1), 83–99.
5. Liu, Fei Tony, Kai Ming Ting, and Zhi-Hua Zhou. "Isolation forest." *2008 Eighth IEEE International Conference on Data Mining*. IEEE, 2008.
6.Jovana, et al. "Agglomerative Hierarchical Clustering." *Datanovia*, 20 Oct. 2018, www.datanovia.com/en/lessons/agglomerative-hierarchical-clustering/.