Benjamin Limoges
Final Project

Project Design:

The results of my experiment are of course highly sensitive to my choice on how to create my synthetic dataset. My natural instinct was to transform this dataset into an unsupervised learning problem, but one in which I knew the labels – in this case it would be the synthetic labels. After pre-processing the data (transforming the variables into Z-scores), I would run a clustering algorithm to find the "normal" data – those that didn't cluster would be labeled as "outliers." I then propose collapsing the clustering that I find in this stage into a binary class problem – outliers versus normal data.

After finding these "labels," of which I have no way of verifying are true, I then do a cross validation with samples from the synthetic dataset to find how accurate I am at rediscovering the labels. Outliers will then have their "outlier measure" calculated and output to the file. In addition, several accuracy measures will be printed to the terminal window.

Creation of the Synthetic Dataset:

The way that I created the synthetic dataset was through agglomerative clustering. The reason I chose to do agglomerative clustering was that I wished for the data to speak for itself by allowing the data to select into clusters. Unlike K-Means, most agglomerative clustering algorithms do not force points to be in a cluster; this allows the algorithm to prune out noise. In this application, the noise is what we're interested in – the unclustered points are the anomalies in our dataset.

For the implementation, I chose a basic version of DBSCAN, the pseudo-code for which I've provided below. My classification is extremely sensitive to the DBSCAN parameters and the distance function used. In my implementation I used Euclidean Distance as my distance metric; the issue with this is that in high dimensional space Euclidean Distance can perform quite poorly, especially if the data are sufficiently sparse. Euclidean distance also weights all of the features equally – without domain knowledge to the contrary my default is to agree with this and weight each feature equally in the distance metric.

The parameters for DBSCAN are also a dicey issue – DBSCAN takes two parameters to help it define the minimum density required for a set of points to be considered a cluster. The epsilon parameter is a radius for the algorithm to look for points to add to it – if the parameter is set too large it becomes too easy to cluster, if it is set to low then it is too hard to cluster. The minpts parameter similarly helps define the density – in the epsilon radius, if there are at least as many points contained as minpts then it is considered a "core" point to the cluster. Setting minpts too high makes there be too few core points, which could result in a lot of points not clustering or a number of small clusters forming.

Setting appropriate values for epsilon and minpts is tricky, as theory doesn't tell us what the optimal values are. Since these measure density, it is highly dependent on the type of data; obviously for denser packed datasets, the minpts should be raised or the epsilon should be decreased. In the case of this project, I used the "domain knowledge" of the approximate percentage of outliers to choose these parameters for both of the datasets. Therefore, the major weakness with this method is that I have to pick the parameters, rather than have the algorithm determine the optimal number of clusters.

Since I am not a domain expert with these datasets, my choice of the clusters will have a huge impact on my results. I tackled this by setting a certain number for minpts for each of the datasets. I picked a slightly higher minpts for dataset1 (5) since it is in fewer dimensions; likewise I picked a lower number (3) for dataset2 since it is 19 dimensions – any points will be sparely populated and therefore I made the density constraint lower. My program iteratively searches for a better epsilon than the starting epsilon, both of which are initialized at 0.01. It terminates when if finds the first epsilon that yields in an outlier percentage within half a percentage point of the outlier percentage given in the assignment. This method is naturally subject to the criticism that its search method is greedy and could result in a suboptimal choice of epsilon. It is also subject to the criticism that there is now an additional parameter – the acceptable error margin.

DBSCAN Algorithm is copied from Wikipedia Directly. I've omitted quotes for ease of reading.

```
DBSCAN(D, eps, MinPts)
  C = 0
  for each unvisited point P in dataset D
    mark P as visited
    NeighborPts = regionQuery(P, eps)
    if sizeof(NeighborPts) < MinPts
      mark P as NOISE
    else
      C = next cluster
      expandCluster(P, NeighborPts, C, eps, MinPts)

expandCluster(P, NeighborPts, C, eps, MinPts)
  add P to cluster C
  for each point P' in NeighborPts
    if P' is not visited
      mark P' as visited
      NeighborPts' = regionQuery(P', eps)
      if sizeof(NeighborPts') >= MinPts
        NeighborPts = NeighborPts joined with NeighborPts'
    if P' is not yet member of any cluster
      add P' to cluster C

regionQuery(P, eps)
  return all points within P's eps-neighborhood (including P)
```

Measure and Detection of Outliers:

Now that I have synthetic labels from the first stage, I transformed the problem to an unsupervised learning problem, in which I will try to rediscover these labels. The algorithm that I implement here will have no knowledge of the synthetic labels.

Again I use DBSCAN but with 10 fold Cross Validation. The folds are created by assigning each data point a random number in the range [0,9]. The downside to this is that the folds may be slightly uneven in size.

The idea of 10 fold cross validation is admittedly a strange way to tackle this project; we usually think of cross validation techniques being used for supervised learning problems. My justification for this approach is that it is a sensitivity analysis – if the outliers can still be observed even if 1/10 of the data is missing, then those points are most likely outliers.

I compute the outliers using DBSCAN as I did in the first part. For those points that do not cluster, P, I calculate the Euclidean distance to the closest point that has clustered, C. I then find the point furthest from C that is within the same cluster as it, C'. The anomaly measure is the ratio of dist(P,C)/dist(C, C'). This measure is computed on each of the cross validations and is summed. Measures greater than 9 are most likely anomalies as P is on average as far away from the cluster as the cluster is wide (in high dimensional space). Measures less than 9 are less likely (but not definitely) to be anomalies, whereas measures greater than 9 are most likely anomalies in the entire dataset. These are the values that will be printed in the results section.

To test the accuracy of rediscovering the labels I created, I printed a confusion matrix for each iteration in the 10 fold cross validation. I have also printed the false positive and false negative rates.

Overfitting:

This deserves its own brief section. There is a high possibility that my model overfits the data. I use the same data to create the labels as I do to test if those labels are accurate. I use the same learner (with the same parameters) over both the cross validation and the initial creation. So it isn't a stretch to say that it rediscovers the labels pretty well.

The only argument I have against saying that my model definitely overfits is that I am using a density based clustering algorithm. As a result, on each of the cross validations, I am removing about 1/10 of the data randomly – this will make the data more sparse on average. So if it overfits, it won't make the learner more accurate – instead is should be less accurate as it should predict more items to be anomalous since the data isn't the same density as it was when I learned the synthetic labels.

Results:

Dataset1:

My false positive rate was rather high on dataset1 – on some runs I was misclassifying half of the synthetically labeled anomalies as normal data. While the accuracy was high, this was only due to the fact that anomalies are an extreme minority class. The number of reported anomalies is also much larger than the number of anomalies overall. I found there to be 18 anomalies in dataset1 when I created my synthetic dataset, but 47 anomalies were printed out. Granted about 15 clearly look like they were anomalous in only one of the folds, but this still leaves around 10 to 15 that were mistakenly printed out. I am not confident that the results produced by the program on this dataset are good – there are just too many potential anomalies. I am confident of the upper half of the list, but not the lower half.

Dataset2:

My false positive rate was low on dataset2, and when it was high it was only because there were a small number of anomalies. My accuracy was similarly high, but it was also similarly inflated. The accuracy in the end was that I almost perfectly matched my results from the creation of the synthetic

labels. I am fairly confident the results produced by the program are fairly accurate of the actual anomalies in this dataset.

Highly Probable Better Solutions:

The best way to create the synthetic labels is most likely a form of Principle Components Analysis (PCA). This would allow me to project the high dimensionality of one of the datasets to a smaller feature space while maintaining the maximum amount of variance in the original form. Then a form of agglomerative clustering could have been applied to this to detect the outliers. While I considered implementing this, I have a fear of matrix programming in Python; I've never used NumPy or SciPy. I didn't want to reinvent the wheel on this project - instead I wanted to reuse as much code from previous projects as possible. As a result I didn't have access to the libraries to easily compute eigenvectors and eigenvalues which would have made PCA a trivial exercise.

Rather than bootstrapping just the original data points, I could have also created synthetic data points in the likeness of the true data points. In particular, I was imagining something like the SIPP Survey – modeling the joint distribution of the sample of points and then making random draws from this distribution. This would most likely make my results more robust. Again, it is to my understanding that this can be rather easily done in SciPy.

References:

DBSCAN Algorithm : http://en.wikipedia.org/wiki/DBSCAN, accessed April 27, 2014.

SIPP Synthetic Beta Data Project : http://www.census.gov/programs-surveys/sipp/methodology/sipp-synthetic-beta-data-product.html, accessed April 29, 2014.