CCP Data Scientist Challenge 2

Solution Abstract

Part 1

To answer the questions in this part it was first necessary to transform the Medicare summary data into a format that would be easier to work with. As the summary data is relatively small, I created some local scripts using Bash, csvkit and Python to combine the DRG and APC files and remove fields that would not be needed. I transformed the data into csv and tsv formats for later use with tools such as Hive and R.

```
See: solution/1-billing/etl/transform_csv.sh
```

For questions b, c and d in this part, I created test scripts and test data from the challenge instructions to help verify that my code was correct.

a) Which three procedures have the highest relative variance in cost?

In order to explore and get some intuition about the variance of the procedures, my first step was to create some simple point density plots using ggplot2 in R, shown in Figure 1.

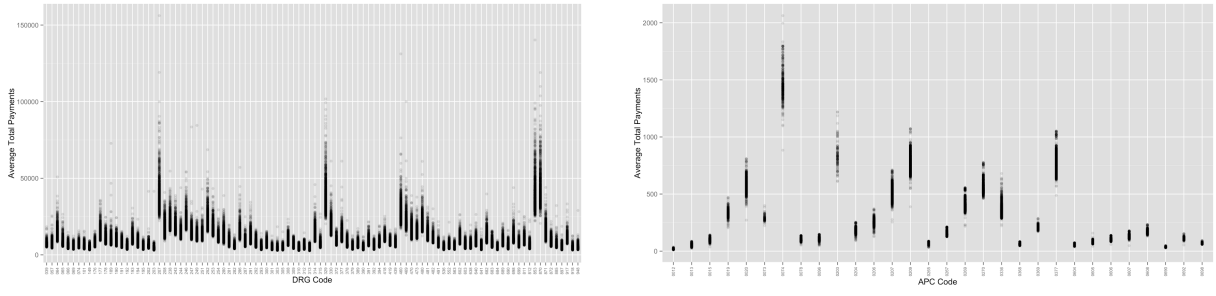

Figure 1

My next step was to compute the relative variance for each procedure, which I did using R. I have interpreted ‘cost’ as the average total payments a provider received for a procedure and ‘relative variance’ as the Coefficient of Variance, the standard deviation divided by the mean.

$$c_v = \frac{\sigma}{\mu}$$

```
See: solution/1-billing/part1a/part1a.R
Solution: solution/1-billing/part1a/part1a.csv
```

b) Which three providers claimed the highest amount for the largest number of procedures?

Rather than load the summary data into a SQL database I opted to solve this and the remaining part 1 questions using Python. Which for data of this size, I find to be easier to reason about when the queries become more complex. So I created a Python script that takes line by line input from stdin and aggregates it by keeping track of which provider charged the most for each procedure. It then counts the number of times each provider charged the most and outputs the top 3 from this list.

```
See: solution/1-billing/part1b/part1b.py
Solution: solution/1-billing/part1b/part1b.csv
```

c) The providers in which three regions claimed the highest average amount for the largest number of procedures?

As above I created a Python script to answer this question. First, the script finds the average charge for each procedure grouped by region (defined as Hospital Referral Region Description in the data). Second, the script finds the regions that charged the most for each procedure. Then it counts how many times a region charged the most for a procedure. Finally, it sorts this list and outputs the top 3.

```
See: solution/1-billing/part1c/part1c.py
Solution: solution/1-billing/part1c/part1c.csv
```

d) Which three providers had the largest claim difference for the largest number of procedures?

As with the previous questions I created a Python script to answer this. First the script finds the providers that had the largest claim difference for each procedure. Next it counts the occurrences by provider. And finally it sorts this list and outputs the top 3.

```
See: solution/1-billing/part1d/part1d.py
Solution: solution/1-billing/part1d/part1d.csv
Part 1 time spent: ~24hrs
```

Part 2

In order to understand which regions and providers are least like the others, we first need to define some notion of similarity between them. Then we can find the ‘expected’ or ‘average’ value of the sample and measure the difference between every region or provider and the ‘average’ value. The least similar will ones will be those that have the greatest difference from the ‘average’.

My approach is to transform what we know about providers and regions in terms of the procedures they carried out and the associated average charges and payments into feature vectors. I then use the Euclidean Distance metric to measure similarity between them and a centroid found by computing the component-wise mean of all feature vectors. Euclidean Distance is used here because magnitude may be important in comparing similar vectors, unlike in the Cosine Distance metric.

I have created scripts to vectorise the features in Python as the data is not too big to be processed on a single machine. The task of computing the similarity is more intensive, but is still not too big to be done on a single machine. I initially prototyped a tool for computing the similarity in C++ but later ported it to Scala. This tool includes some tests to validate that it computes certain values as expected.

```
See: solution/2-similarity/similarity.scala
```

a) Which three providers are least like the others?

First I transform what we can infer about a provider into a dense feature vector. The features are: Number of DRGs, number of APCs, for each procedure, the number of services, the average charges and the average payments. If the provider does not provide a procedure then NA is used for the charges and payments, missing values are later treated as zero.

```
See: solution/2-similarity/part2a/vectorize_providers.py
```

Next the feature vectors are processed by the similarity tool which first finds the centroid and then measures the distance of each provider to the centroid. This list is then sorted by distance and the top 3 are output.

Having found the least similar provider IDs, I was then able to explore what was different about them.

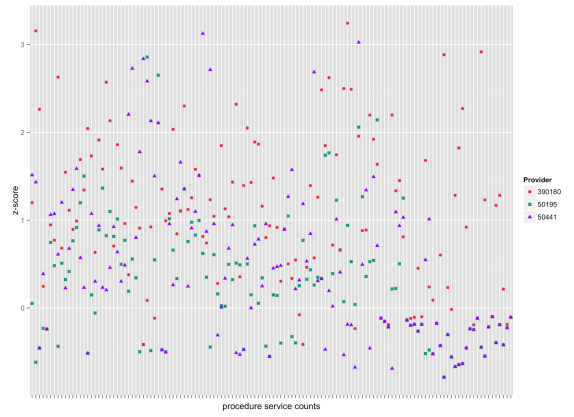

Figure 2

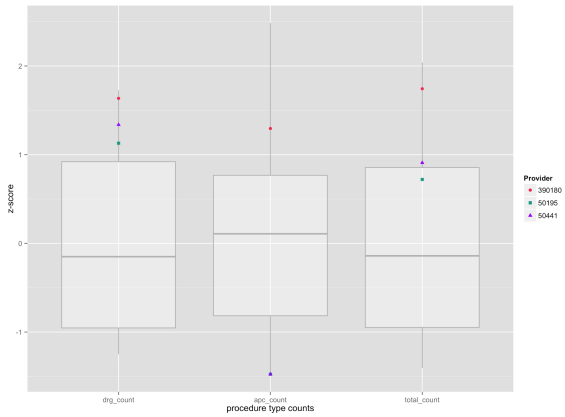

Figure 3

In figure 2 the number of times a provider carried out a procedure is plotted as a normalised z-score. It is quite noisy, but we can see that the three least similar providers are mostly above the mean and in many cases more than 1 standard deviation above the mean. In figure 3 the number of unique DRG and APC type procedures provided are plotted, along with the total. We can see that these providers are more than 1 standard deviation from the mean in many cases. One provider is unusual because it carries out a lot of both APC and DRG type procedures, whereas the other two carry out much less APCs.

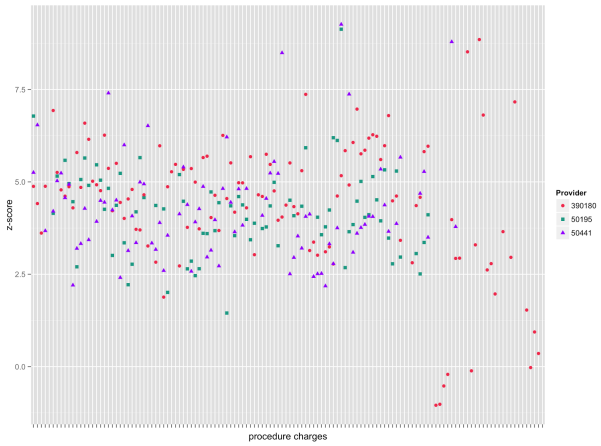

Figure 4

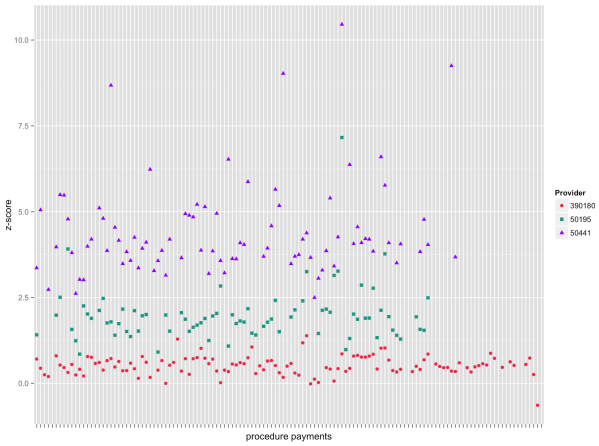

Figure 5

Figure 4 shows the average charges that the three providers made for procedures. We can see that in most cases they charged more than 2.5 standard deviations above the mean. In figure 5 the average payments that the providers received for procedures is plotted. Two of the providers mostly received payments that were more than 2 standard deviations above the mean, and one of them mostly received payments that were around 3 standard deviations above the mean.

```
See: solution/2-similarity/part2a/exploring/plots
Solution: solution/2-similarity/part2a/part2a.csv
```

b) Which three regions are least like the others?

Using a similar process to the previous part, I first created feature vectors aggregated for regions. The features are: Number of providers in the region, number of DRGs, number of APCs, for each procedure, the number of providers who provide the procedure, the number of services, the average charges and the average

payments. If the region does not provide a procedure then NA is used for the charges and payments, missing values are later treated as zero.

```
See: solution/2-similarity/part2b/vectorize_regions.py
```

The least similar regions are then found by the same similarity tools as above.

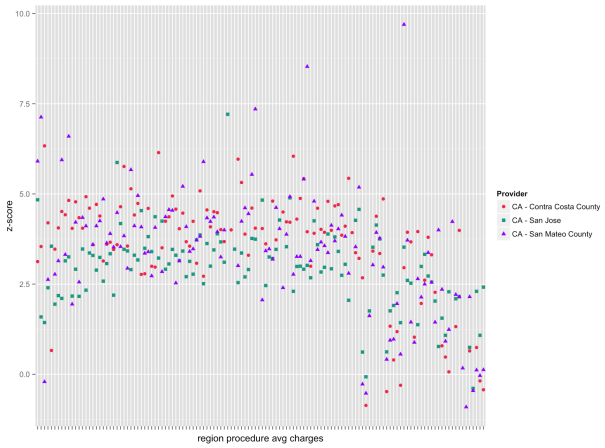

Figure 6

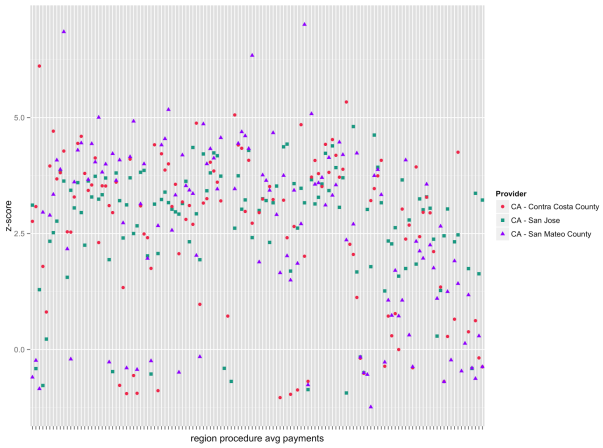

Figure 7

Here we can see that in many cases the three least similar regions charged more than 2 standard deviations above the mean for procedures (figure 6). And often paid more than 2 standard deviations above the mean for procedures (figure 7).

```
See: solution/2-similarity/part2b/exploring/plots
Solution: solution/2-similarity/part2b/part2b.csv
Part 2 time spent: ~40hrs
```

Part 3

Using the list of unusual records as a guide, identify 10,000 additional patient records that seem most likely to also need review.

In order to begin to answer this question it was first necessary to extract the provided patient, claim and review data, and transform it into a format that could be explored. I created a Bash script using Linux tools to extract the PCDR claim data and transform it from ASCII delimited text to tab separated text format and store it into HDFS.

```
See: solution/0-exploring/pcdr_import.sh
```

The patient data is provided as quite a large XML file. So I created a Hadoop MapReduce job using Scrunch to transform it into tab separated format. The records that have been flagged for review could simply be stored into HDFS directly.

```
See: solution/0-exploring/pnt_transform.sh
See: solution/tools/medicare/src/main/scala/medicare/etl/XmlTransform.scala
```

With the data in HDFS the next step was to join the patient records with their claims and the review flags so that I could start to explore what makes a patient worthy of a review. To do this I created Hive tables and used Hive queries to join the records.

```
See: solution/0-exploring/create_patient_claims.hql
```

Generally for anomaly detection, we aim to learn what is most representative of ‘normal’, then anything outside of some threshold can be considered an anomaly. The next step towards this goal was to create feature vectors representing a patient and the claims that they had made. I created a Spark script to vectorise the patient claims. This transforms the patient record (age, gender, income) into nominal and interval variables, and for every claim procedure code it adds a count of the number of claims the patient made for that procedure. I considered adding claim month, average provider charge and payment, and the variance of provider charge and payment as features, but experiments using Weka showed that they would not add much information that would help discriminate between claims.

```
See: solution/3-anomaly/spark-vectorize.scala
```

In order to verify that the expected number of patients and patients for review had been generated, I created a Hive table. I also used this table to take a sample of vectors to experiment with, which included 50,000 records marked for review and a further 100,000 unlabelled records randomly sampled.

```
See: solution/0-exploring/sample_claim_vectors.hql
```

Exploring the data further, some facts became apparent that would help inform the choice of suitable algorithms to employ on the dataset. There are over 100,000,000 patient vectors and only around 50,000 of these are labelled. The only label is for ‘suspicious’ records, everything else is unlabelled or ‘unknown’, so there are no positive and negative examples with which to train a supervised classifier. The large number of unlabelled records and the relatively few labelled records make the dataset not particularly well suited to a k-Nearest Neighbours classifier.

Using Weka I was able to determine that there was not very much variance in the patient ages, genders and incomes with which to discriminate between claims. Also, the information about each claim a patient has made is limited to the number of times the patient claimed for a procedure. Since no patient has claimed more than once for a procedure, each procedure in the patient vector is part of a binomial distribution.

I decided to try clustering the data to see if a clustering could be found where the labelled ‘anomalies’ are assigned to the same clusters. If such a clustering exists then the clusters which have a majority of anomalies could be labelled ‘anomalous’ and any unlabelled points close to the center of that cluster could also be assumed to be anomalous and selected for manual review.

To test this hypothesis, I used Weka to run k-means++ clustering using the Euclidean Distance metric on a small sample dataset to see if the anomalous points would tend to cluster together. This experiment showed that with certain values of k it was possible to find clusters which contained a majority of anomalous points. Experimenting with the features used confirmed that the claims a patient made carried more predictive power than the patients’ age, gender and income.

As the sample I took and indeed the full dataset is too big for a local run of k-means, my next step was to create a tool that could cluster a larger amount of data. For this I created a Spark script and tested it on a small sample to make sure that the results were comparable to Weka. I also created some R scripts to visualise the quality of the clustering using the ‘elbow’ method, and the composition of the clusters.

```
See: solution/3-anomaly/*.R
See: solution/3-anomaly/spark-kmeans.scala
```

I ran the Spark k-means tool over a larger representative sample with k from 2 to 20 in steps of 1 and from 20 to 50 in steps of 2, with a best of 3 runs configuration. Clustering on the sample suggested that the best clustering in terms of Within Set Sum of Squared Errors (WSSSE) would be in the region of k=10..15 (figure 8). From these runs I gathered metrics to visualise the clusterings and choose the most appropriate, trading off clustering quality against grouping of known anomalies (figure 9).

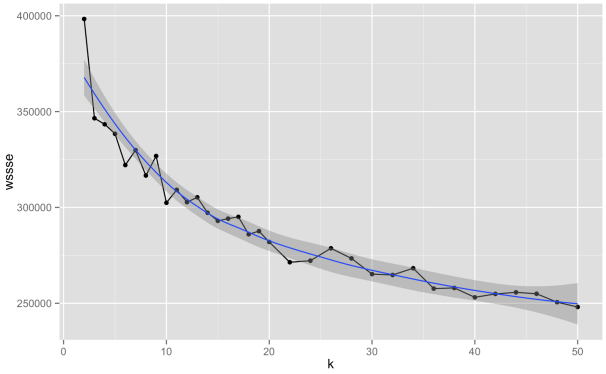

Figure 8

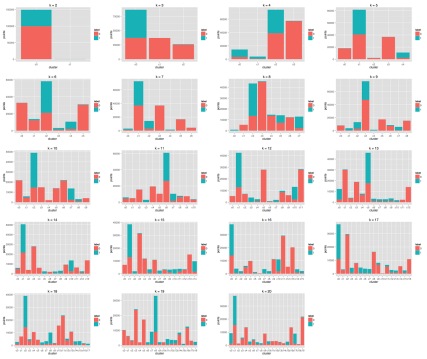

Figure 9

See: solution/3-anomaly/plots/cluster-elbow.png
See: solution/3-anomaly/plots/confusion-2to20by1.pdf

Looking closer at the clusterings, it was possible to identify those close to the optimum number of clusters, which also had clusters with a high ratio of anomalous to unknown points (figure 10). As a simple measure of determining which, out of the clusterings having one or more clusters with a majority of anomalies, had the overall strongest proportion of anomalies, I used the formula:

$$\frac{1}{N}\sum_{i=1}^{c}\frac{a}{u}$$

Where *N* is the total number of points in the subset of clusters, *c* is the number of clusters in the subset, *a* is the number of anomalies in the subset and *u* is the total number of points in the subset. The result of applying this formula to the cluster subsets in figure 10 is shown in figure 11. Here k=12 has a reasonable score and is also in the right vicinity on the elbow graph (figure 8) meaning that the clusters are not too sparse to be useful.

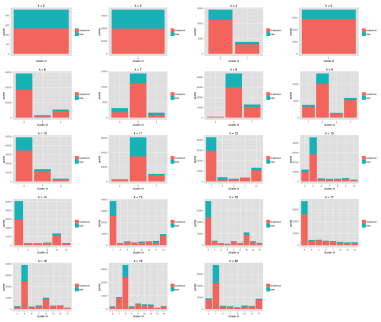

Figure 10

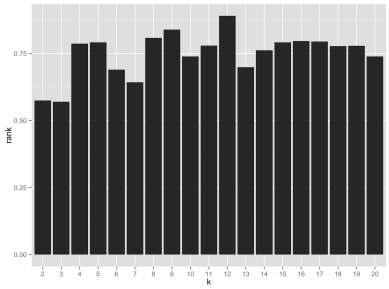

Figure 11

See: solution/3-anomaly/plots/review-2to20by1.pdf
See: solution/3-anomaly/plots/review-ranking-2to20by1.png

I declared the cluster IDs 1,3,4,7,10 as ‘anomalous’ and using the model given by k=12, I ran another pass through the patient vectors selecting those patient IDs that the model predicted to belong in an anomalous cluster and that was not already labelled as an anomaly. The vectors were sorted in terms of distance to the centroid, smallest distance first. The resulting list of patient IDs are worth review because they are similar to those that were hand picked by the staff. They are also worth review in order to confirm whether the label should be ‘anomaly’ or ‘normal’ to help improve the classification performance in future.

See: solution/3-anomaly/spark-kmeans.scala
Solution: solution/3-anomaly/part3.csv
Part 3 time spent: ~56hrs