SCC.403 – Data Mining
Coursework 2021

Rachel Sumner

***Abstract*—In this paper we use libraries in Python 3.7.7 to analyse a sample of publicly available data concerning Manchester's climate from 2010 to 2014, consisting of 938 records that each have 5 features. We perform standard pre-processing techniques, including dimensionality reduction, which is achieved by Principal Component Analysis (PCA). We then performed basic hierarchical and k-means clustering, before finally classifying the data using a decision tree classifier.**

I. Introduction

For this project we were given a sample of publicly available data concerning Manchester's climate from 2010 to 2014, consisting of 938 records that each have 5 features; temperature (degrees C), wind speed (mph), wind direction (degrees), precipitation (mm), and humidity. The overall aim is to find patterns in the data so that we may come up with a useful, interpretable, system of classifications for weather that is commonly observed in Manchester. To do this, a number of steps are involved, starting with pre-processing, where data is sanitised and its dimension reduced so that proper statistical analysis becomes possible and clustering and classification become easier. There were many possible ways to approach this, and overall we opted for a relatively standard approach.

II. Pre-processing

With this project we opted for standard pre-processing methods. In particular, we started by standardising and then normalizing the data. Standardisation was done by simply calculating each data point's Z-score at each feature. Overall, the vast majority of data points had features with Z-score within the $[-3, 3]$ interval, that is to say, they are within three standard deviations of the mean. Given a normal distribution, the odds of an individual feature having a Z-score outside of $[-3, 3]$ is 0.3% (1). As there are 938 data points in our sample, each with 5 features, this translates to approximately 15 data points. If we only assume Chebyshev's inequality holds, as it does for a much wider class of distributions, this would be a maximum of approximately 104 potential outliers. In practice, we identified 27 potential outliers, which hints at the underlying distribution not being normal, although due to the small sample size we cannot know for certain at this stage. We might intuitively expect the distribution to be bimodal, due to the oscillating weather between seasonal extremes. Even though they mostly fit with the distribution, we then removed these 27 outliers, so as to prevent potential distortion in our analysis.

We then normalized the data within $[-1, 1]$. As normalization is simply a matter of scaling, the normalized data maintains the shape of the standardised data. We chose $[-1, 1]$ over $[0, 1]$ purely because immediately after this step we had to centralise the data, as we performed Principal Component Analysis (PCA). PCA is a method of feature extraction, and its primary purpose is to reduce the dimension of our problem space. We achieve a reduction in dimension by using an algorithm that maximises for variance, which is done by operating on the covariance matrix, which in our case was done using the *scipy* library. In doing so, we create new linearly independent orthogonal vectors that serve as a basis that we can project our original data onto. As the principal components are orthogonal, they are uncorrelated, and thus more suited to statistical analysis than the original sample, as we can analyze each component individually. As we are operating on 5 features, we would expect most of the variance to be contained in the first 2 or 3 principal components, and that is indeed the case – see figure 1.

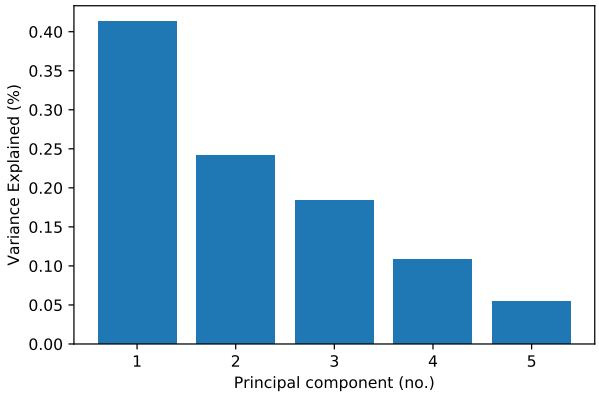

Fig. 1. The first principal component accounts for 41.3% of the variance, while the second accounts for 24.1%, and the third 18.4%.

A simple method to decide how many principal components to use is a 'scree' test, or 'elbow' test (2), wherein we plot the sum of the variance between the first n principal components, and pick the number of principal components nearest the point of inflection. While we do not show the scree test we did here due to space constraints, we found that the point of inflection was between 2 and 3 components. Despite their variance only summing to 65.4%, the first 2 components were chosen. Ideally we would have used the first 3 components, as they account for more than 80% of the variance, but a $3D$ plot proved too unwieldy to interpret, especially with regards to later clustering. See figure 2 for a plot of the transformed data, projected onto the first and second principal components. As each principal component is a linear combination of the

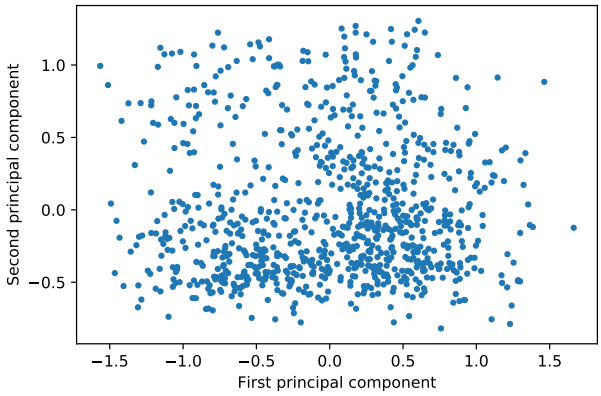

Fig. 2. A scatter plot comparing each point's first and second principal components.

5 features with different weights for each feature, we can roughly interpret each principal component as these weights. The first principal component has (to 3 significant figures) weights of -0.512 for temperature, 0.418 for wind speed, -0.0499 for wind direction, 0.326 for precipitation, and 0.671 for humidity. That is to say, a point that has a high value in its first principal component represents a day that had lower than average temperature, above average wind speed, average wind direction, above average precipitation, and quite high humidity. Similarly, the second component had weights of -0.139 for temperature, -0.244 for wind speed, -0.953 for wind direction, -0.110 for precipitation, and 0.0284 for humidity, representing slightly below average temperature, 'somewhat' below average wind speed, a rare wind direction, slightly below average precipitation, and approximately average humidity. Looking at figure 2, we notice that most of the data lies in the lower half of the graph, with the top left being noticeably sparse and separated from the rest. These groups may form the basis of our clustering.

After performing PCA, we attempted to confirm what kind of distribution the transformed data has. Figure 3 shows a Q-Q (Quantile-Quantile) plot; a type of graph that allows us to compare two distributions by plotting their values at different quantiles (3). In figure 3 we are comparing the normal distribution (the red 45 degree line) with our first principal component (the blue points). If the two lines were to match up perfectly, that would show we have a perfectly normal distribution, and can treat it as such in further analysis. Unfortunately, we do not have a normal distribution, and due to the variation in shape towards the middle and both ends, what distribution we do have is slightly ambiguous. General rules of thumb (4) indicate that our first principal component is distributed bimodally, and may be heavily-tailed. The second component has a similar shape (see Appendix fig. 7), however it is much more heavily-tailed than the first component, which may simply be a result of it representing less of the variance of the original data set. As these distributions are bimodal, this either indicates that there are 2 groups of overlapping weather data, or that the data follows a sinusoidal distribution (i.e. repeats periodically). There will likely be two main types of weather event that are more common than any other, and this is something that should become more apparent when we cluster and classify the data.

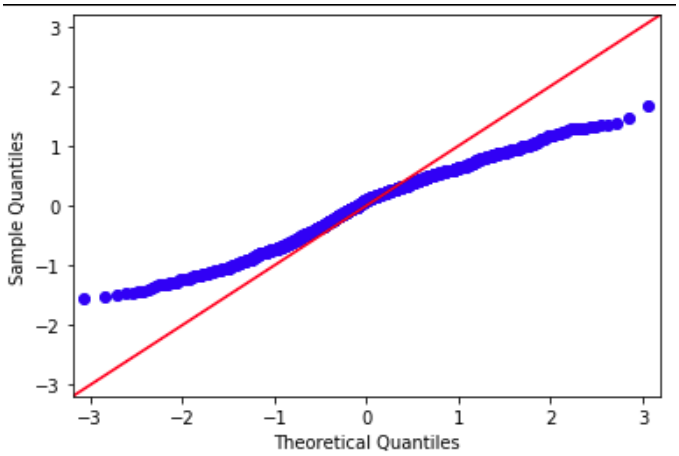

Fig. 3. A Q-Q plot comparing the distribution of our first principal component with a normal distribution.

III. Clustering

As our transformed data set is only 2-dimensional, we can get away with using some fairly inefficient clustering algorithms, as we have such a small data set that we don't have to worry about any objects ballooning out of control in size. In particular, we can use the (Agglomerative) Hierarchical Clustering algorithm, despite its complexity growing at a rate of $O(x^2)$ without having to worry about memory issues. The inefficiency of this algorithm is due to the increasing size of the 'proximity matrix', which stores the distance between every 2 points in our transformed data set. We used the Euclidean metric, and in general the choice of metric can have a large effect on what proximity matrix you end up with. A *dendogram* (see figure 4) is then generated: a branching diagram that puts more similar points closer to one another. Those further apart on the dendogram are more likely to be in separate clusters, and the way in which we decide what clusters we get is by setting an arbitrary limit on how far apart two elements of the same cluster are allowed to be. This is the main pitfall of HC, as there's no way to know whether the clusters generated by your chosen limit will be meaningful until you're attempting to classify the data, and you may have to classify with different arbitrary limits until you end up with a classification system that is robust. Another issue is simply the interpretability of the diagram. Without having already classified the data, we have no idea where say points 4 and 89 even are on our scatter plot (figure 2), and what kind of weather they may represent.

Perhaps this is putting the cart before the horse, but the incompatibility of the dendogram with our original scatter plot

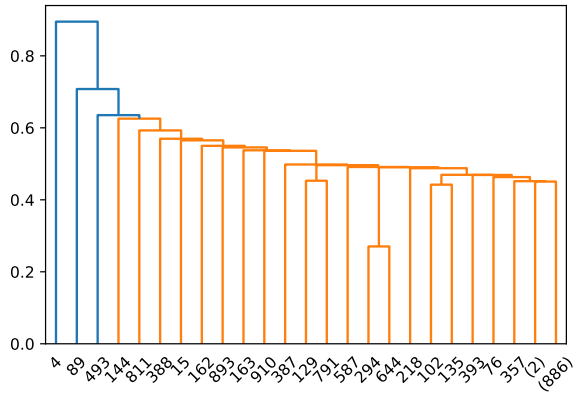

Fig. 4. A dendogram of our transformed data set. The x axis has each data point indexed by order in the standardised data set, starting at 0, while the y axis represents the distance between a point and its neighbours (e.g. 294 and 644 have a distance of approximately 0.3)

makes graphical methods such as k-means clustering, fuzzy k-means clustering, and DBSCAN (Density-Based Spatial Clustering of Applications with Noise (5)) much more appealing. Clustering algorithms based on k-means methods also solve the issue of having to choose an arbitrary cutting point, as similar to PCA we can perform a scree test (2) to find the optimal amount of clusters for our data set. As we are trying to come up with a relatively 'hard' classification system for sorting weather, we felt that regular k-means was more well-suited to the task than fuzzy k-means. This is because fuzzy k-means lets individual data points belong to multiple clusters, and it would obviously be absurd to say a day is, for example, both 'wet and windy' and 'dry and windy'.

The way k-means clustering works is first by placing k arbitrary points in the sample space, which are called the *centroids*, which partition the space into the first iteration of k clusters. The (Euclidean) distance from each point to each centroid is then calculated, and on the first iteration each point is assigned a cluster based on what its nearest centroid is. From every iteration thereafter, each point switches between clusters based on whether it is closer to the other points in its cluster, or some point in another cluster. This last step continues until either we have no points changing clusters, or until we have the same points switching back and forth. In our implementation, this is done by updating the centroids to the mean point of each the cluster at each iteration, and stopping once these points no longer change. Once this is done, we then have k separate clusters that can move on to classification. The method is simple, but its results are rather surprisingly not too dependent on where the centroids are placed, due to the iterative process by which the final clusters are constructed. As mentioned earlier, to find out what k we should choose, we can do a scree test, which is where we plot the amount of clusters against the sum of squared distances from each point to their associated centroid, and then choose the amount of clusters based on which is closest to the point of inflection. Plotting this (see figure 5) we see that the best number of clusters is likely 6, due to how the gradient of the lines change from one side to the other.

Fig. 5. A 'scree' test to find the best amount of clusters for k-means testing. The optimal number is the one closest to the point of inflection. In this case, the optimal amount is 6 clusters.

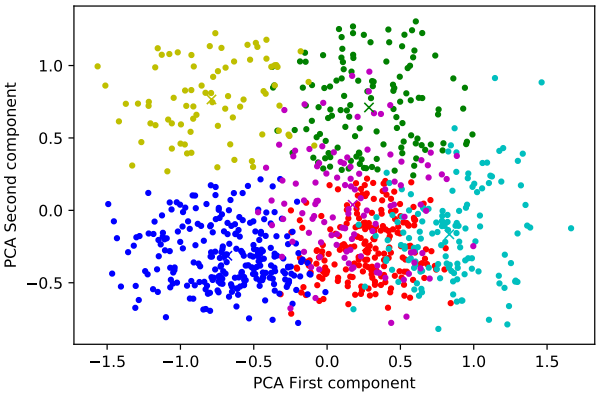

Fig. 6. A k-means classification of our transformed data into 6 clusters.

Now that we've worked out the optimal number of clusters, we can plot what we end up with (see figure 6), and can ideally go on to classify the clusters into distinct weather events. Before that, however, it's worth mentioning that we also could've used DBSCAN (Density-Based Spatial Clustering of Applications with Noise (5)). Unlike k-means testing, which partitions the sample space, DBSCAN is a density-based clustering algorithm, meaning that areas of higher density are placed in the same cluster. One advantage that DBSCAN has over k-means and Hierarchical clustering is that the amount of clusters generated need not be specified in advance (6), so there's no need for scree tests or any other kind of analysis to figure out what is optimal. The only real disadvantage in our case is that some parts of our data (cf. top left and top right of figure 6) may be too sparse to be classified at all. Due to the small size of our data set, however, they likely would both

perform about the same (6), so we felt it best to only test one of them.

IV. Classification

The first method considered was LDA (Linear Discriminant Analysis), which can be used for dimensionality reduction like PCA, but unlike PCA can then classify its results into 2 distinct categories. In our case, we would just be using LDA as a classifier, and not for its feature extraction capabilities, due to having already reduced our data set down to 2 dimensions. LDA is a *generative* classifier, meaning it can roughly be interpreted as an algorithm that generates a function which maps features describing objects onto discrete labels. The main algorithm for LDA is a binary classifier, but will work perfectly fine for any n labels, as we can simply consider one distinct label at a time versus the rest, and iterate the algorithm through these subsets which decrease in size until there are no more labels to be assigned. LDA assumes uncorrelated, independent features, which means it should work well with our transformed data set, as our first and second principal components are uncorrelated. Unfortunately, it also assumes a normal distribution, and as we've seen above, our data set is bimodal (i.e two overlapping normal distributions). This isn't necessarily a problem for most data, as towards the middle of our Q-Q plot (see figure 3) the two samples lined up quite well with one another, but it could lead to the misclassification of outlier data. Given the rarity of outliers, however, this may have been a risk worth taking. Due to the fact that we only have 938 data points, and no way of generating new ones, we will be training all classifiers used on a random sample of 900, and then validating with the last 38. As 900 is such a large majority, we will have to be wary of potential overfitting.

Another potential classifier is an SVM (Suppoert Vector Machine) classifier, which aims to maximise the distance between separate classes by mapping them into a new space where a straight border line can be drawn between them. This isn't always feasible, with the main issue being that overlapping classes are not linearly separable in this way. Looking at our 6 clusters from before (see figure 6), we have quite a large number of overlap between all of our clusters, so SVM would not be feasible. If our data was linearly separable, SVM would be the best choice to make, as classifying data is as simple as seeing if it's on one side of a line or another – although it does require handling for data on the border line itself. Similar to LDA, despite being a binary classifier, it generalises up to any number of classes that are required.

The method we actually went with was a decision tree classifier, as it's a system of classifiers that will work on any type of distribution, compared to classifiers such as LDA that rely on a normal distribution to function. While they are more prone to overfitting than SVM (due to the potential for far too many branches than necessary to be generated), the limits can be finely tuned so as to allow for intuitive classifications that are easy to interpret. The goal of a decision tree is to learn simple rules for classification based on the data, and it does this by extracting simple branching patterns in the trained data. We end up with a root node that consists of a condition, and then a series of branches down into other nodes with different conditions, that continues on until each branch terminates at a leaf representing the label of the piece of data. The main advantage of a decision tree classifier is that they are fast, due to being quite computationally simple, and for a low amount of features the tree generated is often very explainable. They are, however, biased towards features that have more levels of split, and the decision tree generated may be different each time. The biggest issue aside from increased risk of overfitting is that all decision tree algorithms are NP-complete, meaning it is computationally impossible to find an optimal solution. The resulting trees are normally good enough, however – especially on such a small data set.

The classifications we used were based on what clusters the training data points belonged to, resulting in 6 distinct labels. These labels are: 1; *average temperature, average wind*; 2; *cold, very windy, and wet*; 3; *hot, average wind, and wet*; 4; *cold and low wind*; 5; *slightly hot and low wind*; and 6; *slightly hot and very windy*, and were based on the location that each of the 6 centroids ended up, as these are consistent between different executions of the k-means clustering algorithm (up to permutation). We then used the decision tree classifier to decide which cluster a new data point should belong to, and labelled it.

After each of the 38 random points had been validated, we plotted a confusion matrix. A confusion matrix is a $n \times n$ matrix (where n is the amount of classes), that compares how many points ended up in the correct predicted class, versus how many ended up in the incorrect class. It can also be done with a 2×2 matrix that counts true/false negatives and true/false positives, in which case to calculate the accuracy we use the formula

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}.$$

Our result was an over 90% correct classification, which definitely raises concerns of overfitting, as these errors may simply be from points that are on the border between 2 classes. We can then also calculate the 'precision' and 'recall', by the formulas given below:

$$Precision = \frac{TP}{TP + FP}, \; Recall = \frac{TP}{TP + FN}.$$

That is to say, precision is given by the proportion of positives that were correct, while recall calculates the proportion of elements correctly identified for a certain group.

V. Conclusion

Overall we developed a robust classifier, although perhaps with less categories than a typical weather station might use. The main issue we kept running into throughout was that the distribution of the transformed data is either bimodal or sinusoidal, making it unsuited to many methods (such as LDA) that rely on a normal distribution to work. The way this could've been solved is by splitting the data in two (e.g. summer vs winter), however this proved too laborious to

implement. Ideally we also would've worked with DBSCAN, but due to time constraints had to merely leave it at a short description of how the algorithm works, as implementing it with *sklearn* and *scipy* felt rather obtuse, despite the extensive documentation. We were rather surprised that the first two principal components contained such a low number of the variance, and this will have impacted further analysis, as a reduction in variance means reduced accuracy in results obtained from the transformed data. Ideally we would have used the first 3 components, but these were much harder to visualise and interpret.

References

[1] S. Glen, "Empirical rule (68-95-99.7) & empirical research," https://www.statisticshowto.com/probability-and-statistics/statistics-definitions/empirical-rule-2/, 2021, accessed: 2021-08-17.

[2] R. B. Cattell, "The scree test for the number of factors," *Multivariate Behavioral Research*, vol. 1, no. 2, pp. 245–276, 1966, accessed: 2021-08-17. [Online]. Available: https://doi.org/10.1207/s15327906mbr0102_10

[3] M. B. Wilk and R. Gnanadesikan, "Probability plotting methods for the analysis of data," *Biometrika*, vol. 55, no. 1, pp. 1–17, 03 1968, accessed: 2021-08-18. [Online]. Available: https://doi.org/10.1093/biomet/55.1.1

[4] G. B. (https://stats.stackexchange.com/users/805/glen b), "How to interpret a qq plot," https://stats.stackexchange.com/q/101290(version:2017-09-20), accessed: 2021-08-18.

[5] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise." AAAI Press, 1996, pp. 226–231, accessed: 2021-08-19.

[6] "Difference between k-means and dbscan clustering," https://www.geeksforgeeks.org/difference-between-k-means-and-dbscan-clustering/, accessed: 2021-08-21.

Acknowledgements

Libraries used include *math*, *matplotlib*, *numpy*, *pylab*, *scipy*, *sklearn*, and *statsmodels*, which are all freely available through the *Anaconda* navigator and the *pip* package installer.

Appendix

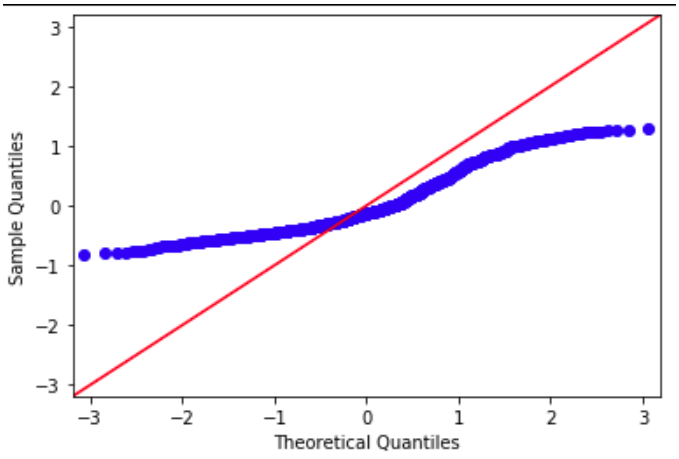

Fig. 7. A Q-Q plot comparing the distribution of our second principal component with a normal distribution.