Analysis of Switzerland Weather Data Set Using Data Mining Techniques

Benjamin McKitterick, 34604413
Computing and Communications, Lancaster University
MSci Hons Computer Science, SCC.403: Data Mining

Abstract *– The objective of this paper is to select and justify specific data mining methods for pre-processing, clustering, and classification on a data set about the climate in Basel, Switzerland.*

1. Introduction

Data mining (DM) is the process of extracting patterns and discovering structure in large and complex data sets [1]. A plethora of techniques exist [2] that allow the extrapolation of advantageous intelligence for many application domains. In this paper, we focus on reviewing and analysing specific methods applicable to weather data clustering and classification.

The weather data set used for this study consists of 1763 records of 18-dimensional vectors. Data pre-processing is a mandatory step of DM that aims to convert prior unusable data into reformatted data that fits a DM process [3]. This stage is unsupervised, meaning no human involvement is needed. The various steps of pre-processing and justifications for them are addressed in section 2 of the paper.

Once the data set has been correctly prepared, section 3 covers clustering of the data; a main and unsupervised task of exploratory DM that involves partitioning and grouping data to identify patterns embedded within. A taxonomy of clustering algorithms exist that can be broadly divided into 6 types: partition, hierarchical, evolutionary, density-based, model-based, and graph-based [4].

In this paper, we choose to analyse a partitioning algorithm; K-Means, and a graph-partitioning algorithm; Spectral clustering. K-Means is deployed widely and commonly used for weather data analysis [5], Spectral clustering has the ability to adapt to different types of data and looks at the affinity of data points rather than their actual placement [6], a more detailed evaluation of these techniques is provided in section 3.1

Classification is one of the final steps in the DM process and a form of supervised ML that uses given labels to predict the class of a set of data points [7]. Classifiers can be either discriminative or generative, as M, Jordan et al., discuss, there exist several reasons to use discriminative classifiers; that model the posterior p(y|x) directly or learn a direct mapping of x to y, rather than generative classifiers [8]. In this paper, we analyse two discriminative classifier models: K-Nearest Neighbour (KNN) and Support Vector Machine (SVM). See section 4 for an overview of theses classification models.

2. Pre-processing

Data pre-processing involves various subtasks; this paper will focus on feature selection, data cleaning, data imputation, data normalisation, data reduction, and feature extraction [3]. The reasoning for why said subtasks were selected will be made clear in the subsequent sub-chapters.

2.1. Feature Selection

A critical issue in analysing complex data is the process of identifying the most influential subset of the original features to use. Removing any irrelevant or redundant features will enhance generalisation by reducing overfitting and improve the performance and interpretability of the processes the data represents. For the weather data set used in this study, all the minimum and maximum features are discarded as they already correlate with and measure the same underlying feature as the mean features. For instance, ‘Temperature (Min)” and “Temperature (Max)” are removed, leaving “Temperature (Mean)”; reducing the number of input variables to those believed to be most useful.

2.2. Data Cleaning

Data cleaning consists of operations necessary to correct anomalous data. In the context of weather datasets, a large storm would result in extreme rainfall, strong winds, and lower temperatures; an outlier. Outliers can significantly impact the mean and standard deviation of results. In this study, outliers are removed first; as opposed to eliminating after standardisation, since the proceeding data reduction method being used is principal component analysis (PCA). Scaled data is needed, as PCA seeks to maximise the variance of components; thus, the integrity of the data should be maintained.

We begin by inspecting each feature of the dataset, where the modality is visually and statistically examined. Figure 1 shows how histogram, time-series and quantile-to-quantile (Q-Q) plots where utilised for visual examination of the data. The time-series graphs showed there was no missing data in any of the features. Frequency distribution was illustrated by the histograms and Q-Q plots aided in assessment of whether data distribution was Gaussian.

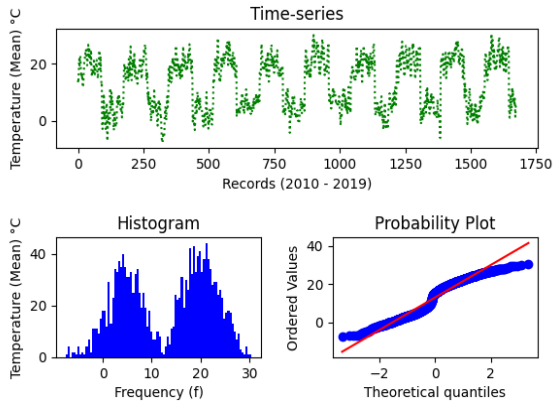


Figure 1. Plots used for visual inspection of each data feature.

For statistical examination, typically, the D' Agostino-Pearson test would be most appropriate, as it provides sensitivity for all conditions when using large data samples [9]. However, after conducting both the Shapiro-Wilk test and D' Agostino-Pearson test, the Shapiro-Wilk test was more powerful and better suited to identify abnormal distributions in this particular dataset.

Once the nature of the data was fully understood, the appropriate method for anomaly removal could be determined. Due to the variety of data distribution of the feature data, a Chebyshev outlier detection method was used; as this calculation holds no assumptions about the distribution of the data. Chebyshev's inequality gives a bound of what percentage of data is outside of k standard deviations from the mean [10]. For Gaussian distribution, 3σ was used for the threshold and 6σ where distribution was abnormal. A total of 103 anomalies were removed from the data set.

2.3. Data Normalisation

Now that the data is clean, it can be standardised to give more emphasis to variables with higher variance. It is necessary to standardise variables so that data can be compared and to eliminate the result of dissimilarity in magnitude or scale of data. In the dataset used for this study, each feature has a different physical meaning and uses independent units, i.e., hPa, Km/h, mm, cm etc. The data was standardised by subtracting the mean and dividing the results by the standard deviation.

2.4. Data Reduction

Dimensionality reduction can be divided into linear and non-linear methods, with PCA being the main linear technique [11]. Other algorithms like T-Distributed Stochastic Neighbour Embedding (T-SNE); a non-linear data visualiser, and Linear Discriminant Analysis (LDA) exist [12], however, PCA is the mother method for multivariate data analysis and a flexible, well-understood tool [11]. The algorithm produces linear combinations of the original features to generate axes, named principal components (PCs).

Having too many features in a dataset causes noise and difficulties, reducing the dimensions will provide a more clear perspective and better visualisation. Moreover, reducing the size of the data and eliminating redundancy is necessary before process-intensive supervised algorithms are used for clustering and classification. After implementing PCA on the dataset, correlated features will be removed, albeit, at the cost of readability and interpretability [13].

To understand how variance is distributed amongst PC's, it is useful to plot the data as in figure 2. To determine the number of PCs to use, a simple cut-off line is used to separate all PC's that have the variance of more than a single variable's worth of data. Thus, if each variable provided the same variance, the cut-off would be 1/p, where p is the number of variables [14]. The first three PC's account for 66% of the variance,

with a significant reduction in percent variance from the first to second PC. The first three PC's are above the cut-off line, and so, will be selected.

2.5. Feature Extraction

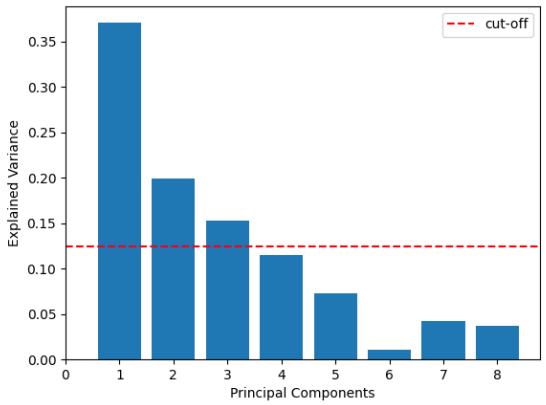


Figure 3. Bar graph to illustrate variance distribution among PCs.

Next, to understand which variables contribute most strongly to each PC, a loading table is examined. The table is composed of positive and negative loadings that can be analysed to give a good sense of what each principal component represents. Table 1, shown below, is the head of the results of the table loadings for the weather data set. From this, we can extract new orthogonal features; for instance, PC2 shows strong positive loadings with temperature and strong negative loadings with sea pressure.

Features	PC1	PC2	PC3
Temperature °C	- 0.377	0.458	0.197
Humidity % rh	0.380	- 0.376	0.332
Sea Pressure hPa	- 0.077	- 0.594	- 0.316
Precipitation mm	0.251	0.227	0.620
Snowfall cm	0.210	- 0.038	- 0.011

Table 1. Table of loadings showing correlation coefficients between variables and PCs.

To further highlight and to provide a more clear physical meaning of the variables' relationship with the PC's a distance biplot is used. The biplot in figure 3 shows how samples are similar to one another and how the variables control the similarity; it is a standard way of showing the sample score and variable loadings in a single plot. After careful analysis of the loadings, the following top correlative variables with PC's have been selected and roughly labelled: PC1 as 'cold, rainy, windy', PC2 as 'hot, dry, breezy', and

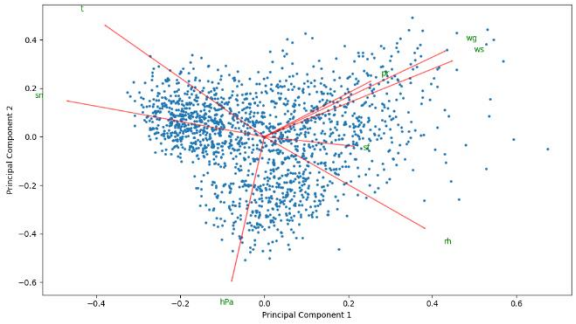


Figure 2. A distance biplot with labels to extrapolate meaning and interpretation of the PCs

PC3 as 'warm, wet, no wind'. Though labelled, further properties can still be extracted from these PCs.

3. Clustering

To identify natural patterns embedded within the data, clustering algorithms aim to minimise the distances between data points of the same clusters and maximise the distance between centres [15]. The following sections: method functionality, cluster analysis, and cluster interpretation aim to explain the functionality of and validate selected algorithms, then, extrapolate meaning to interpret the clusters formed.

3.1. Method Functionality

This paper focuses on two main clustering algorithms: k-means and spectral clustering. We will now discuss the functionality, limitations, and benefits of said algorithms:

While K-Means is an NP-hard problem [16], heuristic methods are capable of finding good estimations to the global optimum in polynomial time. It is also capable of handling datasets efficiently, no matter the shape or size of potential clusters. The K-Means clustering algorithm is a popular unsupervised machine learning (ML) algorithm, with the objective of partitioning high dimensional data based on the distance between data points. Centroids are generated and used to define clusters; a data point being part of a cluster if it is closer to that cluster's centroid than any other. It finds the best centroids by assigning points to clusters and then re-estimating the k cluster centres repetitively until reasonable clusters are obtained [17].

The main problems with K-Means are that it can be skewed by outliers, it struggles to cluster data of varying size and density, and it requires multiple restarts at times to find the local minima. Spectral clustering helps to solve a few of these problems while

avoiding the curse of dimensionality. For instance, formed spectral clusters do not assume any shape or distribution, in contrast to K-Means.

The Spectral clustering algorithm is an unsupervised ML graph partitioning algorithm, which relies on a Laplacian matrix and the proximity between data points to form clusters. It works by assembling a nearest neighbour graph or radius-based graph, and then it makes use of spectral embedding; forming an affinity matrix and applying spectral decompositions to the corresponding graph Laplacian, to embed the data points in low dimensional space. The relevant eigenvectors chosen for the clusters are the ones that correspond to the smallest several eigenvalues of the Laplacian [6].

A big issue with Spectral clustering is that very noisy datasets can cause huge problems, in this case, the relevant eigenvectors may not be in the top few, and computational performance could drop significantly [6]. Both K-Means and Spectral clustering suffer from having to choose the optimum number of clusters, though there are heuristics that help, as discussed in the proceeding subchapter.

3.2. Cluster analysis

There are numerous cluster analysis metrics that can provide insight into the quality of clustering results and the natural tendency of the data to amalgamate. These do not measure the validity of clusters, only the comparative performance against each other. For analysis of the K-Means and Spectral clustering algorithms, the Davies-Bouldin index (DB Index) and Silhouette Coefficient metrics have been selected.

The DB Index evaluates inter-cluster differences and intra-cluster similarities. As the DB index shrinks, the clustering is viewed more positively. The Silhouette Coefficient lets us know how well assigned each data point is to see if they belong to appropriate clusters. The notion of "good clustering" is unclear [18], it is relative to the problem, thus in this case, the chosen quality indexes are deemed a relevant evaluation of clusters for this data set, based on their evaluative criterion. Table 2 shows the scores of both metrics on the K-Means and Spectral clustering.

Metric	K-Means	Spectral
DB Index	0.986	1.195
Silhouette	0.393	0.339

Table 2. Table of DB index & Silhouette score for both the k-means and spectral clustering algorithms.

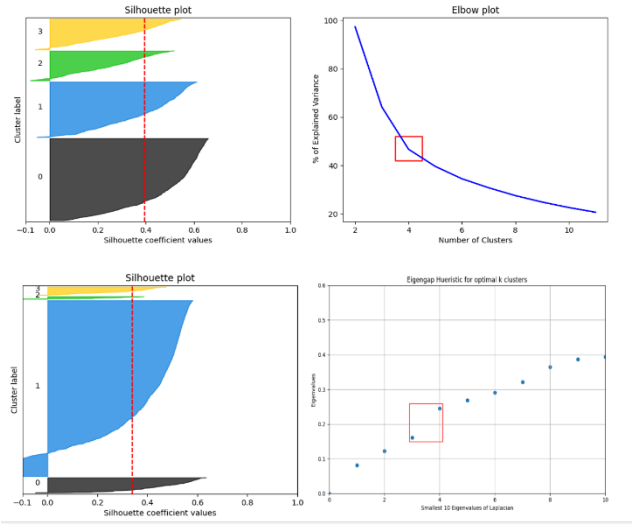


Figure 4. Silhouette plots alongside elbow and eigengap heuristic plots to analyse clusters and determine optimal k.

Determining the optimal k number of clusters is subjective and depends on the method used for measuring similarity as well as the parameters used for partitioning. To determine the optimal k for K-Means a well-known "elbow plot" [17] can be used to look at the total within-cluster sum of the square; a measurement of the compactness of the clustering. The location of a "knee" or bend in the plot is an indicator of the appropriate number of clusters.

To determine the optimal k for Spectral Clustering, Ulrike von Luxburg proposed an approach based on perturbation theory and spectral graph theory known as the eigengap heuristic [6]. The heuristic suggests the optimal k is the value that maximises the eigengap; the difference between consecutive eigenvalues. The red box highlights the optimal k for both Spectral and K-Means in figure 4.

3.3. Cluster Interpretation

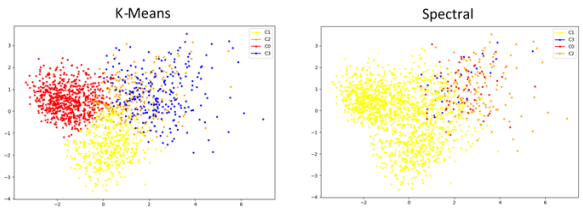


Figure 5. Scatter plot results of both K-Means and Spectral clustering.

As shown by figure 5, the K-Means clusters are more clear and divisible, whereas the spectral clusters have formed one large cluster with three much smaller clusters. The weather dataset is very noisy, and as

discussed in section 3.1, this has skewed the resulting clusters. Fortunately, the K-Means algorithm has effectively grouped the clusters into interpretable, well-defined chunks. To determine what these clusters represent, parallel coordinate plots; like the one shown in figure 6, can be used to analyse how the values for the variables compare across the clusters.

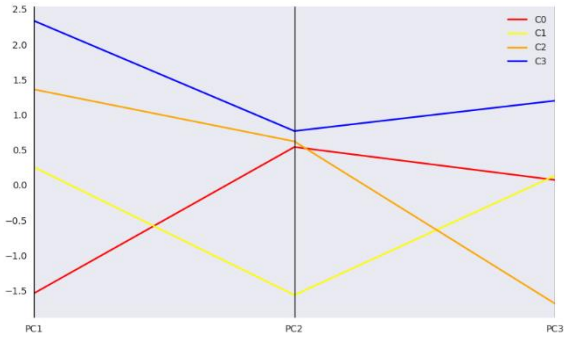


Figure 6. Parallel coordinates plot to visualise and interpret where data points sit across PCs.

After careful inspection of the variables that compose the PC's and examination of how the PC's relate to the clusters, labels with meaningful names have been obtained for use in the classification task. They are as follows: C0 is 'Hot, drizzly, no wind', C1 is 'Mild, rainy, breezy', C2 is 'Cold, dry. Blustery', and C3 is 'Cold, rainy, blustery'.

4. Classification

KNN is a prototype-based classifier and a type of instance-based learning, or lazy learning, as all computation is only done once a query has been made [19]. Data points are classified by a plurality vote of its neighbours, and the data point is then assigned to the most common label amongst its k nearest neighbours.

While KNN is simple to implement, requires no training; a time complexity of linear O(n) [20] makes it computationally quicker than SVM, and has few hyperparameters to tune; k value and distance function, it is unfortunately sensitive to noisy data and outliers. Furthermore, it doesn't work well with large datasets or high dimensions, and its hyperparameter k needs to be chosen wisely [21].

SVM is a non-probabilistic binary linear classifier and an eager learner. When given a set of training samples, the algorithm maps them to categories, while ensuring a clear gap between them. New sample points can then be assigned to divisible spaces, and then predicted to belong to one of these spaces depending on the side of the gap on which the point lies [22].

In contrast to KNN, SVM works relatively well in high dimensional spaces and is more effective when the number of dimensions is greater than the number of samples. SVM has a time complexity of non-linear $O(n^2)$ [23] but takes care of outliers better than KNN and functions greatly when there is a clear margin of separation between classes. SVM isn't suitable for large data sets, and like KNN is susceptible to datasets with more noise. Based on this brief review, KNN is expected to function better on the weather dataset.

4.2. Hyperparameter Tuning

In KNN, finding the hyperparameter k is difficult. A value too low will mean that noise will have a more significant effect, a value too high will make it computationally expensive. Research has shown that no optimal number of neighbours apply to all types of datasets [20], so, to find the optimal value for k on the weather dataset, the mean error rate that corresponds with each k value is plotted, where k ranges from 1 to 40 as shown in figure 7. From the output, we can see the mean error is below 0.010 when the value of k is between 20 and 21.

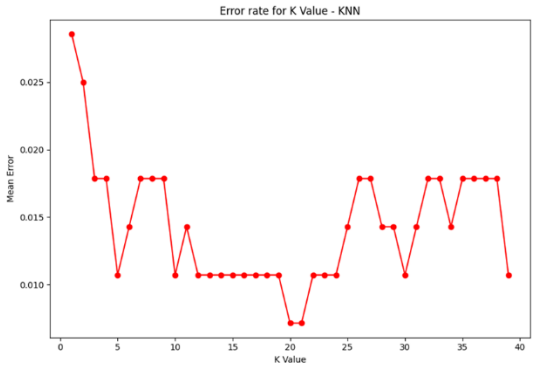


Figure 7. Plot to compare the error rate with k value.

4.2. Evaluating Performance

To determine whether a measured approximate level of accuracy on test data will be retained when the classification model is used on future unseen data, we must perform cross-validation. One drawback of this is that computations are repeated for each k value, which increases time complexity [24]. So, to acquire a less biased model, repeated k-fold cross-validation is performed to determine the number of groups that the data sample should be split into. For instance, when the KNN models neighbour parameter was 20, the optimum number of splits was 6.

Various performance metrics can be used to determine the quality of predictions made by each classifier. True positives, false positives, true negatives, and false negatives are used to calculate the metrics shown in table 3 below. The data shows that SVM was slightly worse at finding all positive instances, meaning the proportion of actual weather data points correctly classified was less than KNN. The classification accuracy of SVM was also slightly lower, seemingly KNN is the better classifier here, although the accuracy measurement can be misleading as it does not detect true negatives. As discussed in section 4.1, KNN is computationally quicker than SVM; however, the k value for this model was 20, explaining why SVM was faster.

	KNN	SVM
Precision	0.99	0.99
Recall	0.99	0.98
F1-Score	0.99	0.99
Support	280	280
Accuracy	0.993	0.986
Time (ms)	1.058	0.089

Table 3. Performance metric values and computational time for KNN and SVM classifiers.

The results shown in figure 8 and 9, visualise the classification of some of the labelled data for KNN and SVM on a 2D plane. The illustrations show which errors occur and reflect how far data points lie from the different classes. The visualisations for other tested kernels for SVM are provided in the appendix.

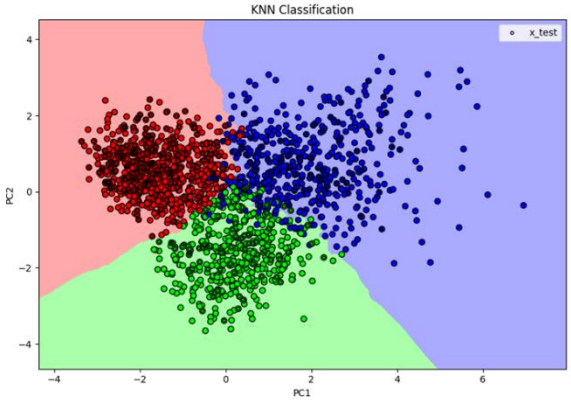


Figure 8. KNN classification model visualization

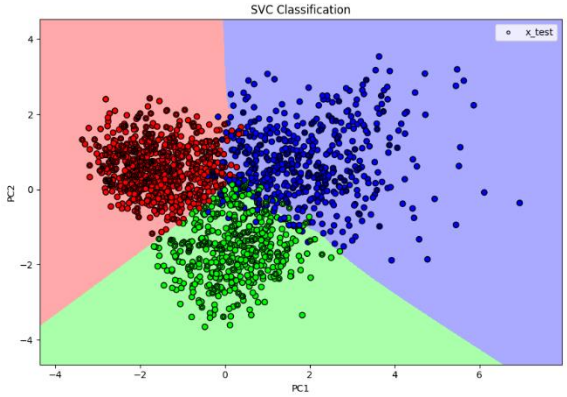


Figure 9. SVM classification mode visualisation using "linear" kernel.

5. Conclusion

In this paper, we compare and analyse various techniques for data pre-processing, clustering and classification. We find that on this weather dataset K-Means performed better than spectral clustering and KNN was only slightly better than SVM for classifying the data. The results concur with many other findings discussed in the literature.

If more time was available, a broader survey on existing methods for all steps of the DM process could have been undertaken, to determine better-suited techniques specifically for the weather data. Most existing research in this field study using DM techniques for prediction and forecasting weather, few research papers cover only classification of weather data. More specifically, it would have been interesting to explore further dimensionality reduction algorithms to see how these impact the results of the clustering and classification algorithms.

Acknowledgements

This study was supported by Prof. Angelov, Director of Research at the School of Computing and Communications at Lancaster University. The following Python libraries were helpful for visualisation and implementation of various methods: Matplotlib, SciPy, NumPy, Scikit-learn, Pandas.

References

[1] D. J. Hand and N. M. Adams, “Data Mining,” Wiley StatsRef: Statistics Reference Online, 2014.

[2] N. Jain and V. Sriastava, “Data Mining Techniques: A Survey Paper,” International Journal of Research in Engineering and Technology , Rajasthan, 2013.

[3] S. Garcia, J. Luengo and F. Herrera, Data Preprocessing in Data Mining, Cham: Springer International Publishing, 2016.

[4] S. Tripathy and S. Hota, “A Survey On Partitioning and Parallel Partitioning Clustering Algorithms,” International Conference on Computing and Control Engineering, Bhubaneswar, 2012.

[5] S. Chakraborty, N. K. Nagwani and L. Dey, “Weather Forecasting using Incremental K-means Clustering,” arXiv, 2014.

[6] U. V. Luxburg, “A tutorial on spectral clustering,” Springer, 2007.

[7] S. Kotsiantis and I. D. Z. P. E. Pintelas, “Machine learning: A review of classification and combining techniques,” Springer, 2006.

[8] A. Y. Ng and M. I. Jordan, “On Discriminative vs. Generative classifiers: A comparison of logistic regression and naive Bayes,” NIPS, 2001.

[9] M. Saculinggan and E. A. Balase, “Empirical Power Comparrison of Goodness of Fit Tests for Normality in The Presence of Outliers,” in *Journal of Physics: Conference Series 435*, Cagayan de Oro, 2012.

[10] B. G. Amidan, T. A. Ferryman and S. K. Cooley, “Data Outlier Detection using the Chebyshev Theorem,” IEEE AC, 2005.

[11] “What Is Principal Component Analysis (PCA) and How It Is Used?,” Sartorius, 18th August 2020. [Online]. Available: https://www.sartorius.com/en/knowledge/science-snippets/what-is-principal-component-analysis-pca-and-how-it-is-used-507186. [Accessed 15th December 2020].

[12] C. Ding, X. He, H. Zha and H. Simon, “Adaptive Dimension Reduction for Clustering High Dimensional Data,” in *Proceedings of International Conference on Data Mining*, 2002.

[13] S. Karamizadeh, S. M. Abdullah, A. A. Manaf, M. Zamani and A. Hooman, “An Overview of Principal Component Analysis,” Journal of Signal and Information Processing, 2013.

[14] S. M. Holland, “Principal Components Analysis (PCA),” University of Georgia, 2019.

[15] X. R and W. D, Clustering, John Wiley & Sons, 2008, pp. 1-12.

[16] D. S, The hardness of k-means clustering, San Diego: Department of Computer Science and Engineering, University of California, 2008.

[17] M. Ahmed, R. Seraj and S. M. S. Islam, “The k-means Algorithm: A Comprehensive Survey and Performance Evaluation,” MDPI Electronics, 2020.

[18] R. L and M. O, “Clustering methods,” Springer, Boston, 2005.

[19] N. S. Altman, “An introduction to kernel and nearest-neighbor nonparametric regression,” The American Statistician, 1992.

[20] A. B. Hassanat, M. A. Abbadi and G. A. Altarawneh, “Solving the Problem of the K Parameter in the KNN Classifier Using an Ensemble Learning Approach,” International Journal of Computer Science and Information Security, Karak, 2014.

[21] I. A. A. Amra and A. Y. A. Maghari, “Students Performance Prediction Using KNN and Naive Bayesian,” in *8th International Conference on Information Technology (ICIT)* , Gaza, 2017.

[22] B. E. Boser, I. M. Guyon and V. N. Vapnik, “A training algorithm for optimal margin classifiers,” in *Proceedings of the fifth annual workshop on Computational learning theory*, 1992.

[23] L. Bottou and C.-J. Lin, “Support Vector Machine Solvers,” Large scale kernel machines, 2007.

[24] J. D. Rodrı´guez, A. Pe´rez and J. A. Lozano, “Sensitivity Analysis of k-Fold Cross Validation in Prediction Error Estimation,” IEEE, 2010.

Appendix A: Pre-processing Results

A.1. Figure 1 - Time-series, Histogram, and quantile-to-quantile plots.

```
ax1 = plt.subplot(grid[0, :])
plt.title("Time-series")
plt.xlabel("Records (2010 - 2019)")
plt.ylabel(features[i])
ax1.plot(x, y , 'g', ls='dotted')
plt.subplots_adjust(hspace=0.75, wspace=0.35)

ax2 = plt.subplot(grid[1, :-1])
plt.title("Histogram")
plt.xlabel("Frequency (f)")
plt.ylabel(features[i])
plt.subplots_adjust(bottom=0.19)
ax2.hist(data_feature, color="blue", edgecolor="none",
         bins=int(len(data_feature)/18), orientation=u'vertical')

ax3 = plt.subplot(grid[1, 1])
plt.title("Q-Q plot")
stats.probplot(data_feature, dist="norm", plot=ax3)
plt.show()
```

A.2. Figure 2 - Bar graph illustrating variance distribution amongst principal components.

```
plt.axhline(y=1/n_principal_components, c='r', linestyle="--")
plt.bar(range(1,len(explained_var_ratio)+1), explained_var_ratio)
plt.xticks(range(n_principal_components+1))
plt.ylabel("Explained Variance")
plt.xlabel("Principal Components")
plt.legend(('cut-off',),loc="upper right")
```

A.3. Figure 3 - Biplot to show PCA results and variable loadings together.

```
plt.figure(figsize=(12,10))
xs = pca_result[:,0]
ys = pca_result[:,1]
n = loadings.shape[0]
scalex = 1.0/(xs.max() - xs.min())
scaley = 1.0/(ys.max() - ys.min())
plt.scatter(xs*scalex,ys*scaley, s=5)
for i in range(n):
    plt.arrow(0,0, loadings[i,0], loadings[i,1], color='r', alpha=0.5)
    plt.text(loadings[i,0]*1.15, loadings[i,1]*1.15, labels[i], color ='g', ha='center', va='center')
    plt.xlabel("Principal Component 1")
    plt.ylabel("Principal Component 2")
plt.show()
```

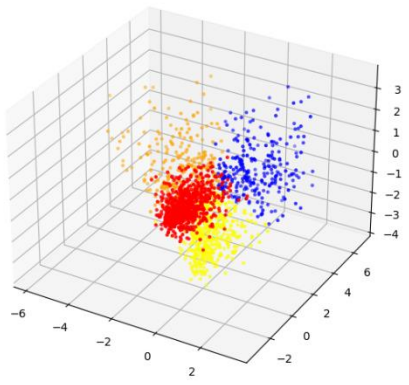
Appendix B: Clustering Results

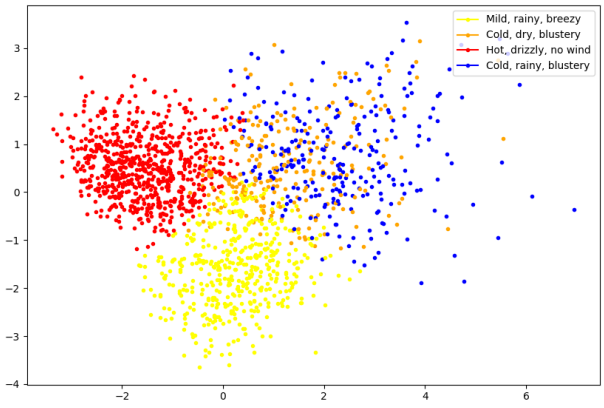
B.1. Figure 4 - Perform eigen decomposition on the affinity matrix and identify eigenvalues with the largest magnitude.

```
L = csgraph.laplacian(spectral.affinity_matrix_, normed=True)
n_components = spectral.affinity_matrix_.shape[0]
eigVal, eigVect = eigsh(L, k=n_components, which="LM",sigma=1.0,maxiter=1000)

fig, ax1 =  plt.subplots(1, 1, figsize=(10,8))
plt.scatter(np.arange(len(eigVal)), eigVal)
ax1 = plt.gca()
ax1.add_patch(Rectangle((2.9, 0.15),1.2,0.11, fill=None, alpha=1, color='r'))
plt.title("Eigengap Hueristic for optimal k clusters")
plt.xlabel("Smallest 10 Eigenvalues of Laplacian")
plt.ylabel("Eigenvalues")
plt.xlim(0,10)
plt.ylim(0,0.6)
plt.grid()
plt.show()
```

B.2. Figure 5 - Function to plot clusters on 2D and 3D plots to visualise and interpret data more clearly, shown by the graphs beneath.

```
def plot_clusters(cluster_labels, data):
    labels=["Hot, drizzly, no wind","Mild, rainy, breezy",
        "Cold, dry, blustery","Cold, rainy, blustery"]
    plt.figure(figsize=(10,8))
    colours = [colors[i] for i in cluster_labels]

    ''' 2D plot with labels '''
    for i in range(len(data)):
        plt.plot(data[i,0], data[i,1], c=colors[cluster_labels[i]], label=labels[cluster_labels[i]], marker='.')
    handles, labels = plt.gca().get_legend_handles_labels()
    label = OrderedDict(zip(labels, handles))
    plt.legend(label.values(), label.keys(), loc="upper right")

    ''' 3D plot '''
    fig = plt.figure(figsize=(10,8))
    ax2 = fig.add_subplot(111, projection='3d')
    ax2.scatter(data[:,2], data[:,0], data[:,1],
           c=colours, marker='.', label=labels[cluster_labels[i]])
    plt.show()
```





B.3. Figure 4 - , A function to plot silhouette's alongside elbow and eigengap heuristic plots.

```
def plot_sil_elbow(data, n_clust, cluster_labels, s_score, e_score):
    fig, (ax1,ax2) = plt.subplots(1,2,figsize=(15,5))
    ax1.set_xlim([-0.1,1])
    ax1.set_ylim([0, len(data) + (n_clust+1) *10])

    ''' First, plot silhouette '''
    sil_val = silhouette_samples(data, cluster_labels)
    lower = 10
    for i in range(n_clust):
        i_cluster_sil_val = sil_val[cluster_labels == i]
        i_cluster_sil_val.sort()
        size_cluster_i = i_cluster_sil_val.shape[0]
        upper = lower + size_cluster_i

        ''' Fill and colour the silhouette's '''
        color = cm.nipy_spectral(float(i)/n_clust)
        ax1.fill_betweenx(np.arange(lower, upper), 0, i_cluster_sil_val,
                          facecolor=color, edgecolor=color, alpha=0.7)
        ax1.text(-0.05, lower +0.5 * size_cluster_i, str(i))
        lower = upper + 10

        ax1.set_title("Silhouette plot")
        ax1.set_xlabel("Silhouette coefficient values")
        ax1.set_ylabel("Cluster label")
        ax1.axvline(x=s_score, color="red", linestyle="--")

        ax1.set_yticks([])
        ax1.set_xticks([-0.1, 0, 0.2, 0.4, 0.6, 0.8, 1])

        ''' Second, plot the elbow '''
        ax2.plot(np.arange(2,12), e_score, color='b')

        ''' Highlight the optimum amount of clusters with red
            unfilled rectangle. '''
        ax2 = plt.gca()
        ax2.add_patch(Rectangle((3.5, 42),1,10, fill=None, alpha=1, color='r'))

        ax2.set_yticks([20, 40, 60, 80, 100])
        ax2.set_title("Elbow plot")
        ax2.set_xlabel("Number of Clusters")
        ax2.set_ylabel(" % of Explained Variance")

    plt.show()
```

B.4. Figure 6 - Parallel plot to interpret cluster by seeing where data points sit across all PCs.

```
def plot_parallel(centroids):

    centroids = pd.DataFrame(centroids, columns=['PC1','PC2','PC3'])
    centroids['clusters'] = ['C0','C1','C2','C3']

    with plt.style.context(("ggplot", "seaborn")):
        fig = plt.figure(figsize=(10,6))
        pd.plotting.parallel_coordinates(centroids, 'clusters', color=colors)

    plt.show()
```

Appendix C: Classification Results

C.1. Figure 7 – Simple plot showing the corresponding error rate for each k value.

```
error_rate = []
for i in range(1, 40):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(x_train, y_train)
    pred_index = knn.predict(x_test)
    error_rate.append(np.mean(pred_index != y_test))

    plt.figure(figsize=(10,8))
    plt.plot(range(1, 40), error_rate, c='r', marker='o')
    plt.title("Error rate for K Value - "+str(model))
    plt.xlabel("K Value")
    plt.ylabel("Mean Error")
    plt.show()
```

C.2. Figure 8 & 9 – Function to visualise any classification model, support vectors can also be plotted for SVM when the code is uncommented.

```
def visualize_classification(data_set, labels, x_test ,model, title):
    Y = labels
    X = data_set[:, :2]

    # Create color maps
    light = ListedColormap(['#FFAAAA', '#AAFFAA', '#AAAAFF'])
    bold = ListedColormap(['#FF0000', '#00FF00', '#0000FF'])

    clf = model
    clf.fit(X,Y)

    # plot the decision boundary.
    x_min, x_max = X[:,0].min() - 1, X[:,0].max() + 1
    y_min, y_max = X[:,1].min() - 1, X[:,1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, .01), np.arange(y_min, y_max, .01))
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])

    # Put results in color plot
    Z = Z.reshape(xx.shape)
    plt.figure(1, figsize=(10,8))
    plt.pcolormesh(xx, yy, Z, cmap=light, alpha=0.8)

    # plot the data and circle out the test data
    plt.scatter(X[:,0], X[:,1], c=Y, cmap=bold, edgecolor='k',)
    plt.scatter(x_test[:,0], x_test[:,1], s=10, facecolors='none',
                zorder=10, edgecolors='k', label="x_test")

    # plot the support vectors
    #sup_vect = clf.support_vectors_
    #plt.scatter(sup_vect[:,0], sup_vect[:,1], color='orange')

    plt.title(title)
    plt.legend()
    plt.xlabel("PC1")
    plt.ylabel("PC2")

    plt.xlim(xx.min(), xx.max())
    plt.ylim(yy.min(), yy.max())
```

C.3. Additional SVM visualisations for radial basis function and polynomial kernels.