Chapter 10
Machine Learning for Patient Stratification and Classification Part 2: Unsupervised Learning with Clustering

Cátia M. Salgado and Susana M. Vieira

Abstract Machine Learning for Phenotyping is composed of three chapters and aims to introduce clinicians to machine learning (ML). It provides a guideline through the basic concepts underlying machine learning and the tools needed to easily implement it using the Python programming language and Jupyter notebook documents. It is divided into three main parts: part 1—data preparation and analysis; part 2—unsupervised learning for clustering and part 3—supervised learning for classification.

Keywords Machine learning · Phenotyping · Data preparation · Data analysis · Unsupervised learning · Clustering · Supervised learning · Classification · Clinical informatics

10.1 Clustering

Clustering is a learning task that aims to decompose a given set of observations into subgroups (clusters) based on data similarity, such that observations in the same cluster are more closely related to each other than observations in different clusters. It is an unsupervised learning task, since it identifies structures in unlabeled datasets, and a classification task, since it can give a label to observations according to the cluster they are assigned to. For a more detailed description of supervised and unsupervised learning please refer to the previous chapter.

This work focuses on the following questions:

- Can we identify distinct patterns even if the class labels are not provided?
- How are the different patterns represented across different outcomes?

In order to address these questions, we will start by providing a description of the basic concepts underlying k-means clustering, which is the most well known and simple clustering algorithm. We will show how the algorithm works using 2D data

C. M. Salgado (✉) · S. M. Vieira
IDMEC, Instituto Superior Técnico, Universidade de Lisboa, Av. Rovisco Pais, Lisboa, Portugal
e-mail: catia.salgado@tecnico.ulisboa.pt

L. A. Celi et al. (eds.), *Leveraging Data Science for Global Health*,
https://doi.org/10.1007/978-3-030-47994-7_10

as an example, perform clustering of time series and use the information gained with clustering to train predictive models. The k-means clustering algorithm is described next.

10.1.1 K-means Clustering Algorithm

Consider a (training) dataset composed of N observations:

$$x_1, x_2, ..., x_N$$

Initialize K centroids $\mu_1, \mu_2, ..., \mu_K$ randomly.
Repeat until convergence:

1. Cluster assignment
   Assign each x_i to the nearest cluster. For every i do:

$$\underset{j}{argmin} \left\| x_i - \mu_j \right\|^2,$$

   where $j = 1, 2, ..., K$.
2. Cluster updating
   Update the cluster centroids μ_j. For every j do:

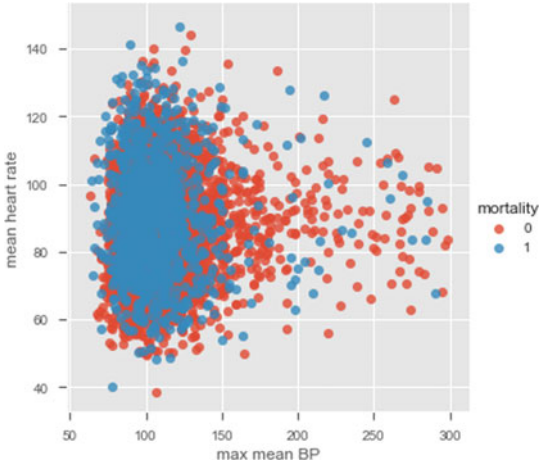
$$\mu_j = \frac{1}{N_j}\left[x_1^j + x_2^j + ... + x_{N_j}^j\right],$$

   where N_j is the number of observations assigned to cluster $j, k = 1, 2, ..., N_j$, and x_k^j represents observation k assigned to cluster j. Each new centroid corresponds to the mean of the observations assigned in the previous step.

10.1.2 Exemplification with 2D Data

Although pairwise plots did not reveal any interesting patterns, some clusters might have emerged after the data were transformed. You can re-run the code for pairwise plots between transformed features, but note that it will be time consuming due to the high dimensionality of the dataset. Features 'max mean BP' and 'mean heart rate' were chosen for illustrative purposes. The dataset is plotted below:

```
In [26]: x1 = 'max mean BP'
         x2 = 'mean heart rate'
         sns.lmplot(x1, x2, data_transf_inv, hue="mortality", fit_reg=False);
```



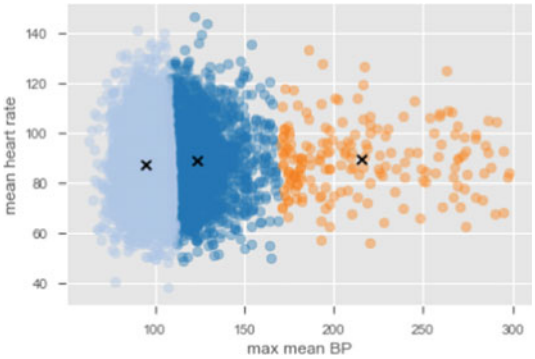
The number of clusters (K) must be provided before running k-means. It is not easy to guess the number of clusters just by looking at the previous plot, but for the purpose of understanding how the algorithm works 3 clusters are used. As usual, the 'random_state' parameter is predefined. Note that it does not matter which value is defined; the important thing is that this way we guarantee that when using the predefined value we will always get the same results.

The next example shows how to perform k-means using 'sklearn'.

```
In [27]: from sklearn.cluster import KMeans

         # set the number of clusters
         K = 3

         # input data to fit K-means
         X = pd.DataFrame.as_matrix(data_transf_inv[[x1,x2]])

         # fit kmeans
         kmeans = KMeans(n_clusters=K, random_state=0).fit(data_transf_inv[[x1,x2]])
```

The attribute 'labels_' gives the labels that indicate to which cluster each observation belongs, and 'cluster_centers_' gives the coordinates of cluster centers representing the mean of all observations in the cluster. Using these two attributes it is possible to plot the cluster centers and the data in each cluster using different colors to distinguish the clusters:

```
In [28]: classes = kmeans.labels_
         centroids = kmeans.cluster_centers_

         # define a colormap
         colormap = plt.get_cmap('tab20')
         for c in range(K):
             centroids[c] = np.mean(X[classes == c], 0)
             plt.scatter(x = X[classes == c,0], y = X[classes == c,1], alpha = 0.4, c =
         colormap(c))
             plt.scatter(x = centroids[c,0], y = centroids[c,1], c = 'black', marker='x')

         plt.xlabel(x1)
         plt.ylabel(x2)
         display.clear_output(wait=True)
```



The algorithm is simple enough to be implemented using a few lines of code. If you want to see how the centers converge after a number of iterations, you can use the code below, which is an implementation of the k-means clustering algorithm step by step.

```
In [29]: # The following code was adapted from http://jonchar.net/notebooks/k-means/
         import time
         from IPython import display

         K = 3

         def initialize_clusters(points, k):
             """Initializes clusters as k randomly selected coordinates."""
             return points[np.random.randint(points.shape[0], size=k)]

         def get_distances(centroid, points):
             """Returns the distance between centroids and observations."""
             return np.linalg.norm(points - centroid, axis=1)

         # Initialize centroids
         centroids = initialize_clusters(X, K)
         centroids_old = np.zeros([K, X.shape[1]], dtype=np.float64)

         # Initialize the vectors in which the assigned classes
         # of each observation will be stored and the
         # calculated distances from each centroid
         classes = np.zeros(X.shape[0], dtype=np.float64)
         distances = np.zeros([X.shape[0], K], dtype=np.float64)

         # Loop until convergence of centroids
         error = 1
         while error>0:

             # Assign all observations to the nearest centroid
             for i, c in enumerate(centroids):
                 distances[:, i] = get_distances(c, X)

             # Determine class membership of each observation
             # by picking the closest centroid
             classes = np.argmin(distances, axis=1)

             # Update centroid location using the newly
             # assigned observations classes
             # Change to median in order to have k-medoids
             for c in range(K):
                 centroids[c] = np.mean(X[classes == c], 0)
                 plt.scatter(x = X[classes == c,0], y = X[classes == c,1], alpha = 0.4, c =
         colormap(c))
                 plt.scatter(x = centroids[c,0], y = centroids[c,1], c = 'black', marker='x')
```

```
        error = sum(get_distances(centroids, centroids_old))
        centroids_old = centroids.copy()

        #pl.text(max1, min2, str(error))
        plt.xlabel(x1)
        plt.ylabel(x2)
        display.clear_output(wait=True)
        display.display(plt.gcf())
        time.sleep(0.01)
        plt.gcf().clear()
```

Please refer to the online material in order to visualize the plot. It shows the position of the cluster centers at each iteration, until convergence to the final centroids. The trajectory of the centers depends on the cluster initialization; because the initialization is random, the centers might not always converge to the same position.

10.1.3 Time Series Clustering

Time series analysis revealed distinct and interesting patterns across survivors and non-survivors. Next, k-means clustering is used to investigate patterns in time series. The goal is to stratify patients according to their evolution in the ICU, from admission to t = 48 h, for every variable separately. Note that at this point we are back to working with time series information instead of constructed features.

For this particular task and type of algorithm, it is important to normalize data for each patient separately. This will allow a comparison between time trends rather than a comparison between the magnitude of observations. In particular, if the data is normalized individually for each patient, clustering will tend to group together patients that (for example) started with the lowest values and ended up with the highest values, whereas if the data is not normalized, the same patients might end up in different clusters because of the magnitude of the signal, even though the trend is similar.

Missing data is filled forward, i.e., missing values are replaced with the value preceding it (the last known value at any point in time). If there is no information preceding a missing value, these are replaced by the following values.

```
In [30]: # Now we are going to pivot the table in order to have rows corresponding to unique
         # ICU stays and columns corresponding to hour since admission. This will be used for
         clustering

         def clustering(variable, ids_clustering, K, *args):
             """Return data for clustering, labels attributted to training observations and
             if *args is provided return labels attributted to test observations"""

             data4clustering = timeseries_data(data_median_hour, variable, filldata = 1)

             # data for clustering is normalized by patient
             # since the data is normalized by patient we can normalize training and test data
         together
             for index, row in data4clustering.iterrows():
                 maxx = data4clustering.loc[index].max()
                 minn = data4clustering.loc[index].min()
                 data4clustering.loc[index] = (data4clustering.loc[index] - minn) / (maxx-minn)
```

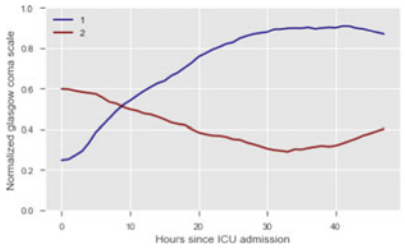
```
    # select data for creating the clusters
    data4clustering_train = data4clustering.loc[ids_clustering].dropna(axis=0)
    print('Using ' + str(data4clustering_train.shape[0]) + ' ICU stays for creating the
clusters')

    # create the clusters
    kmeans = KMeans(n_clusters = K, random_state = 2).fit(data4clustering_train)
    centers = kmeans.cluster_centers_
    labels = kmeans.labels_

    # test the clusters if test data is provided
    labels_test = []
    for arg in args:
        data4clustering_test = data4clustering.loc[arg].set_index(arg).dropna(axis=0)
        labels_test = kmeans.predict(data4clustering_test)
        labels_test = pd.DataFrame(labels_test).set_index(data4clustering_test.index)
        print('Using ' + str(data4clustering_test.shape[0]) + ' ICU stays for cluster
assignment')

    print(str(K) + ' clusters')
    cluster=0
    d = {}
    mortality_cluster = {}

    colormap1 = plt.get_cmap('jet')
    colors = colormap1(np.linspace(0,1,K))

    fig1 = plt.figure(1, figsize=(15,4))
    fig2 = plt.figure(2, figsize=(15,3))

    for center in centers:
        ax1 = fig1.add_subplot(1,2,1)
        ax1.plot(center, color = colors[cluster])

        ax2 = fig2.add_subplot(1,K,cluster+1)
        data_cluster =  data4clustering_train.iloc[labels==cluster]
        ax2.plot(data_cluster.transpose(),  alpha = 0.1, color = 'silver')
        ax2.plot(center, color = colors[cluster])
        ax2.set_xlabel('Hours since admission')
        if cluster == 0:
            ax2.set_ylabel('Normalized ' + variable)
        ax2.set_ylim((0, 1))
        cluster += 1
        data_cluster_mort =
data['mortality'].loc[data_cluster.index].groupby(['icustay']).mean()
        print('Cluster ' + str(cluster) + ': ' + str(data_cluster.shape[0]) + '
observations')
        mortality_cluster[cluster] = sum(data_cluster_mort)/len(data_cluster_mort)*100
        d[cluster] = str(cluster)

    labels = pd.DataFrame(labels).set_index(data4clustering_train.index)

    ax1.legend(d)
    ax1.set_xlabel('Hours since ICU admission')
    ax1.set_ylabel('Normalized ' + variable)
    ax1.set_ylim((0, 1))

    ax3 = fig1.add_subplot(1,2,2)
    x, y = zip(*mortality_cluster.items())
    ax3.bar(x, y, color=colors)
    ax3.set_xlabel('Cluster')
    ax3.set_ylabel('Non-survivors (%)')
    ax3.set_xticks(np.arange(1, K+1, step=1))

    plt.show()

    if args:
        return data4clustering, labels, labels_test
    else:
        return data4clustering, labels
```

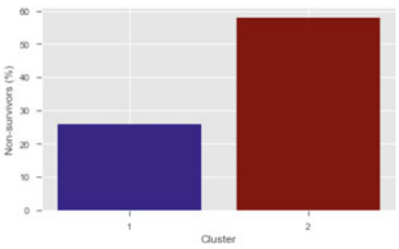
10.1.3.1 Visual Inspection of the Best Number of Clusters for Each Variable

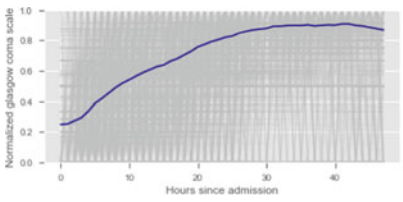
In the next example, k-means clustering is performed for glasgow coma scale (GCS), for a varying number of clusters (K). Only the training data is used to identify the clusters. The figures show, by order of appearance: cluster centers, percentage of non-survivors in each cluster, and cluster centers and training data in each cluster.

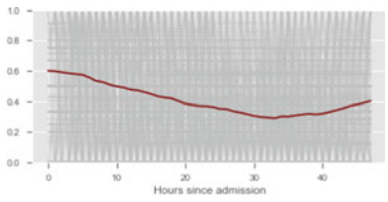
```
In [31]: variable = 'glasgow coma scale'
         clusters = range(2, 6)

         ids_clustering = X_train.index.unique()
         for K in clusters:
             data4clustering, cluster_labels = clustering(variable, ids_clustering, K)
```

```
Using 2110 ICU stays for creating the clusters
2 clusters
Cluster 1: 1215 observations
Cluster 2: 895 observations
```

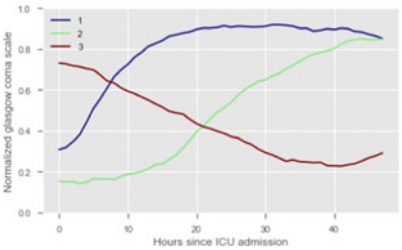


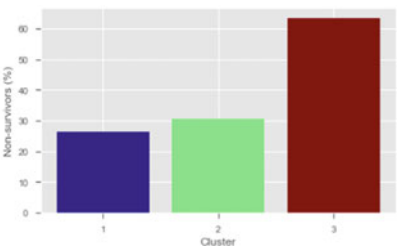


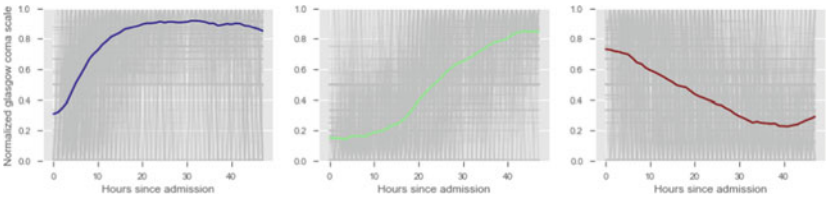




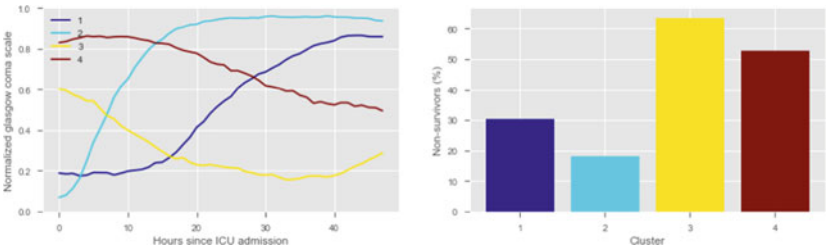

```
Using 2110 ICU stays for creating the clusters
3 clusters
Cluster 1: 806 observations
Cluster 2: 631 observations
Cluster 3: 673 observations
```

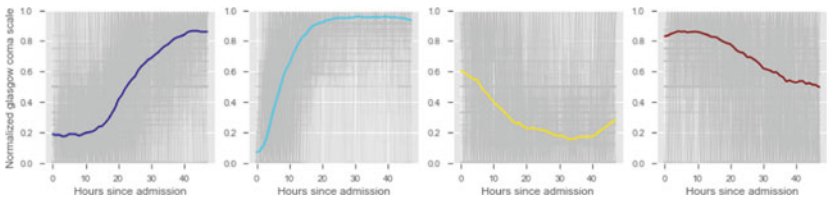




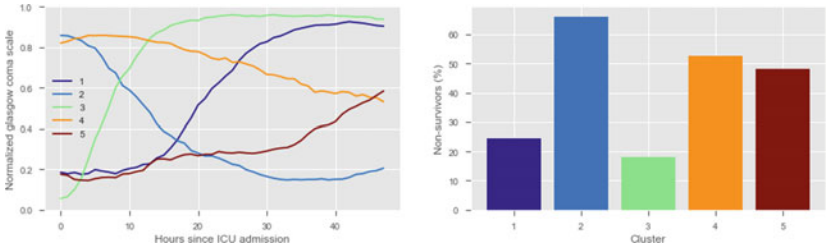


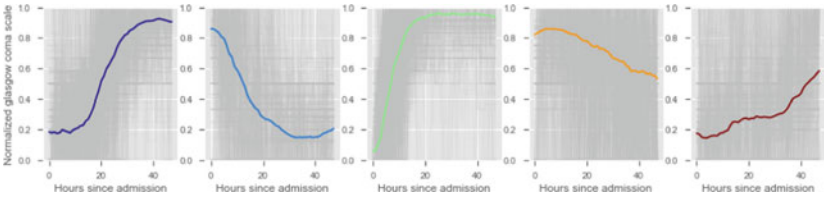

```
Using 2110 ICU stays for creating the clusters
4 clusters
Cluster 1: 579 observations
Cluster 2: 578 observations
Cluster 3: 453 observations
Cluster 4: 500 observations
```






```
Using 2110 ICU stays for creating the clusters
5 clusters
Cluster 1: 471 observations
Cluster 2: 328 observations
Cluster 3: 514 observations
Cluster 4: 464 observations
Cluster 5: 333 observations
```





The goal of plotting cluster centers, mortality distribution and data in each cluster is to visually inspect the quality of the clusters. Another option would be to use quantitative methods, typically known as cluster validity indices, that automatically give the "best" number of clusters according to some criteria (e.g., cluster compactness, cluster separation). Some interesting findings are:

- $K = 2$
  - shows two very distinct patterns, similar to what was found by partitioning by mortality;
  - but, we probably want more stratification.
- $K = 3$
  - 2 groups where GCS is improving with time;
  - 1 group where GCS is deteriorating;
  - yes, this is reflected in terms of our ground truth labels, even though we did not provide that information to the clustering. Mortality$>$ 60% in one cluster versus 30% and 28% in the other two clusters.
- $K = 4$
  - one more "bad" cluster appears.
- $K = 5$
  - Clusters 2 and 4 have similar patterns and similar mortality distribution. GCS is improving with time;
  - Clusters 3 and 5 have similar mortality distribution. GCS is slowly increasing or decreasing with time;
  - Cluster 1 is the "worst" cluster. Mortality is close to 70%.

In summary, every K from 2 to 5 gives an interesting view of the evolution of GCS and its relation with mortality. For the sake of simplicity, this analysis is only shown for GCS. You can investigate on your own the cluster tendency for other variables and decide what is a good number of clusters for all of them. For now, the following K is used for each variable:

```
In [32]: # create a dictionary of selected K for each variable
         Ks = dict([('diastolic BP', 4),
                    ('glasgow coma scale', 4),
                    ('glucose', 5),
                    ('heart rate', 5),
                    ('mean BP', 5),
                    ('oxygen saturation', 3),
```

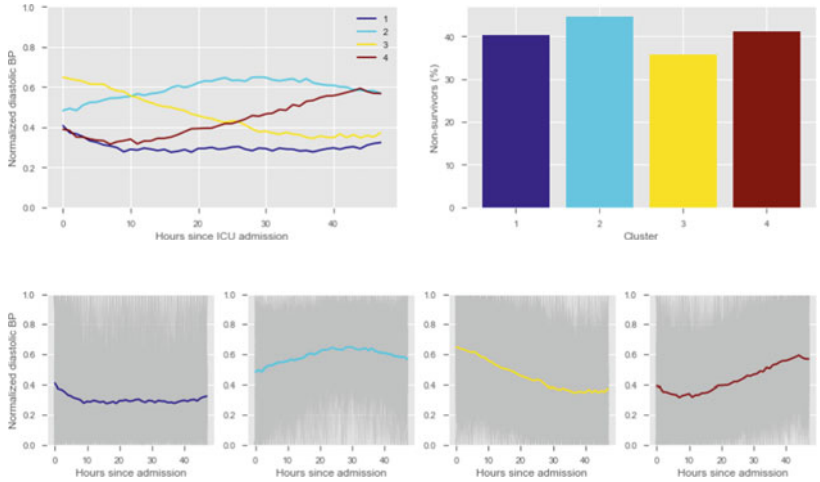
```
                ('respiratory rate', 5),
                ('systolic BP', 4),
                ('temperature', 4),
                ('pH', 4),
            ])
```

10.1.3.2 Training and Testing

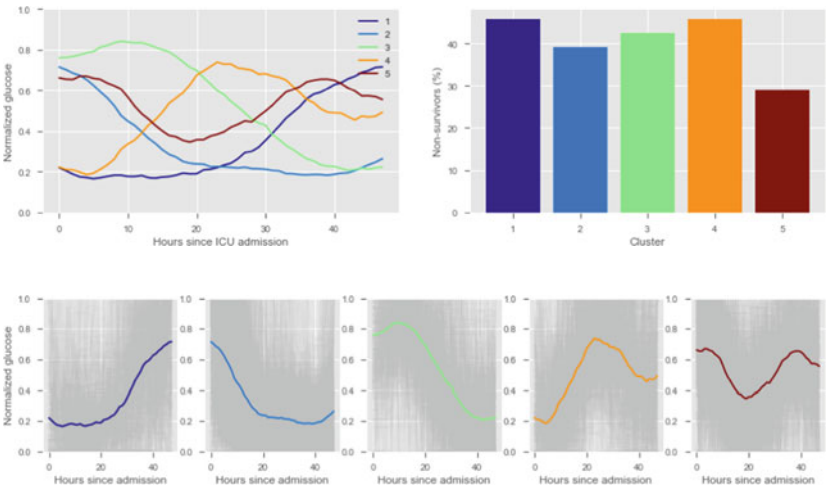
In this work, cluster labels are used to add another layer of information to the machine learning models. During the training phase, clustering is performed on the time series from the training set. Cluster centers are created and training observations are assigned to each cluster. During the test phase, test observations are assigned to one of the clusters defined in the training phase. Each observation is assigned to the most similar cluster, i.e., to the cluster whose center is at a smaller distance. These observations are not used to identify clusters centers.

The next example trains/creates and tests/assigns clusters using the 'clustering' function previously defined. Cluster labels are stored in 'cluster_labels_train' and 'cluster_labels_test'.

```
In [33]: id_train = X_train.index.unique()
         id_test = X_test.index.unique()
         cluster_labels_train = pd.DataFrame()
         cluster_labels_test = pd.DataFrame()

         for feature in variables:
             print(feature)
             K = Ks[feature]
             data4clustering, labels_train, labels_test = clustering(feature, id_train, K,
         id_test)

             labels_test.columns=['CL ' + feature]
             labels_train.columns=['CL ' + feature]

             cluster_labels_train = pd.concat([cluster_labels_train, labels_train],
         axis=1).dropna(axis=0)
             cluster_labels_test = pd.concat([cluster_labels_test, labels_test],
         axis=1).dropna(axis=0)


         for col in cluster_labels_train:
             cluster_labels_train[col] = cluster_labels_train[col].astype('category')

         for col in cluster_labels_test:
             cluster_labels_test[col] = cluster_labels_test[col].astype('category')

diastolic BP
Using 2352 ICU stays for creating the clusters
Using 1387 ICU stays for cluster assignment
4 clusters
Cluster 1: 633 observations
Cluster 2: 414 observations
Cluster 3: 658 observations
Cluster 4: 647 observations
```

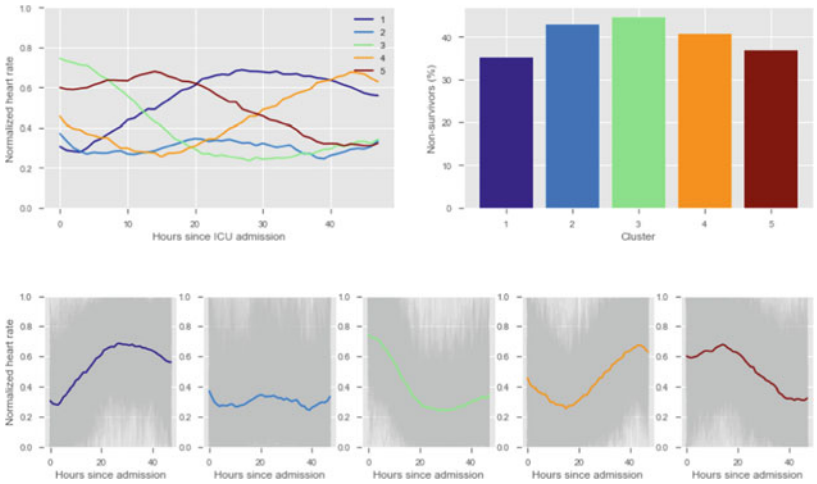



```
glucose
Using 2351 ICU stays for creating the clusters
Using 1387 ICU stays for cluster assignment
5 clusters
Cluster 1: 362 observations
Cluster 2: 716 observations
Cluster 3: 431 observations
Cluster 4: 398 observations
Cluster 5: 444 observations
```

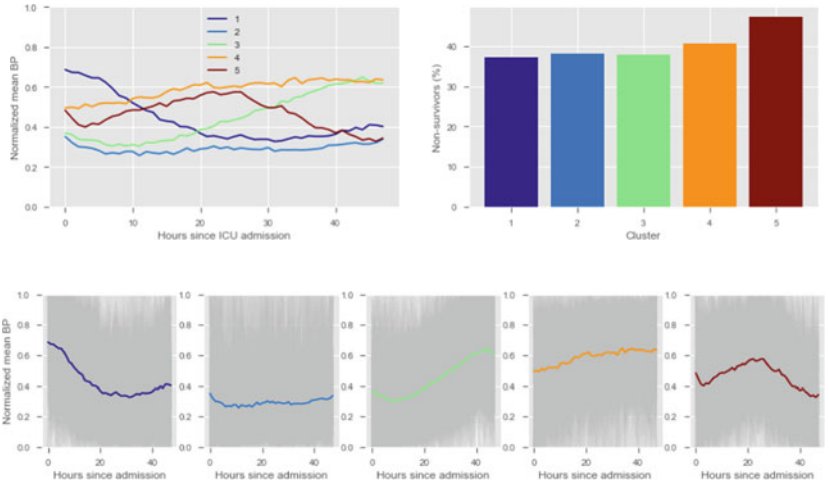



```
heart rate
Using 2350 ICU stays for creating the clusters
Using 1387 ICU stays for cluster assignment
5 clusters
Cluster 1: 455 observations
Cluster 2: 455 observations
```

```
Cluster 3: 452 observations
Cluster 4: 482 observations
Cluster 5: 506 observations
```

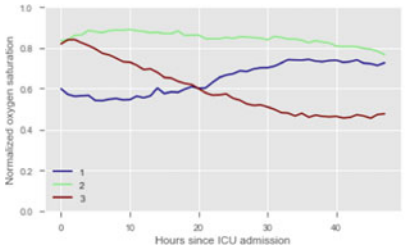


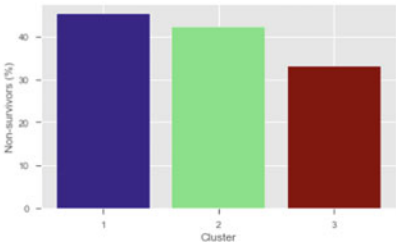

```
mean BP
Using 2352 ICU stays for creating the clusters
Using 1387 ICU stays for cluster assignment
5 clusters
Cluster 1: 505 observations
Cluster 2: 499 observations
Cluster 3: 526 observations
Cluster 4: 381 observations
Cluster 5: 441 observations
```

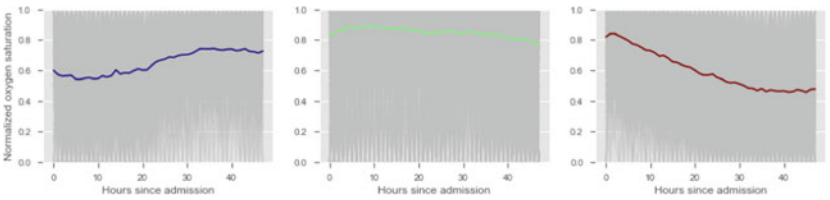



```
oxygen saturation
Using 2344 ICU stays for creating the clusters
Using 1378 ICU stays for cluster assignment
```

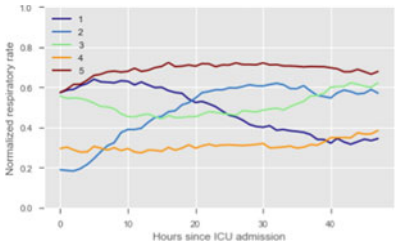
```
3 clusters
Cluster 1: 466 observations
Cluster 2: 1143 observations
Cluster 3: 735 observations
```

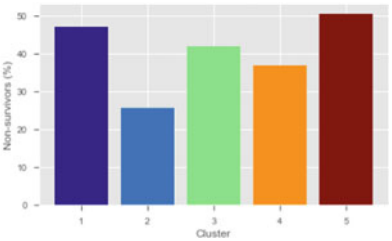


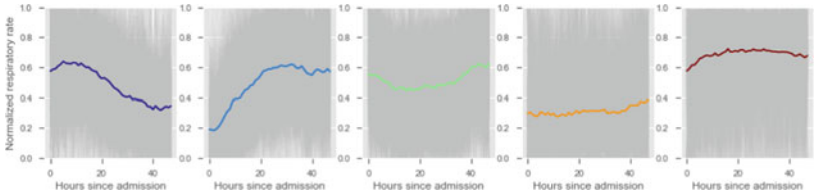





```
respiratory rate
Using 2352 ICU stays for creating the clusters
Using 1387 ICU stays for cluster assignment
5 clusters
Cluster 1: 385 observations
Cluster 2: 500 observations
Cluster 3: 516 observations
Cluster 4: 468 observations
Cluster 5: 483 observations
```

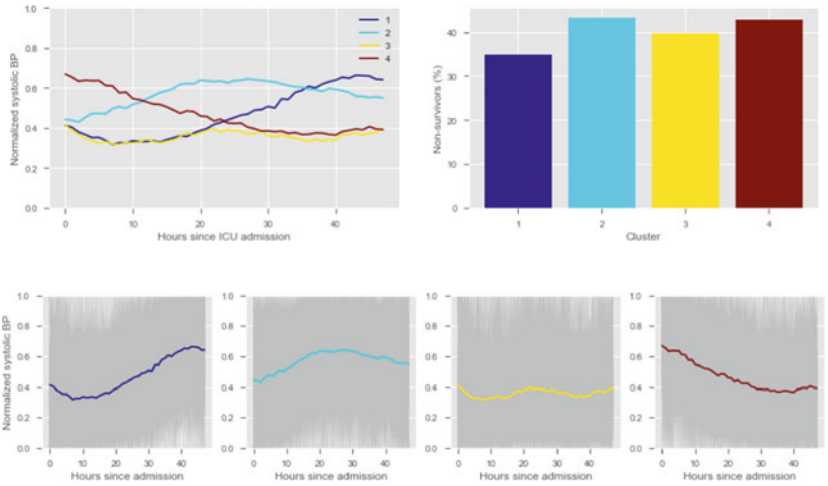




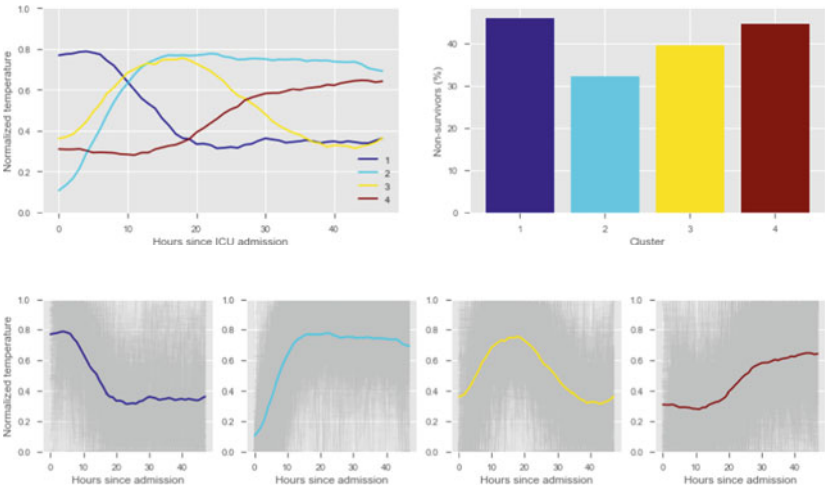



```
systolic BP
Using 2352 ICU stays for creating the clusters
```

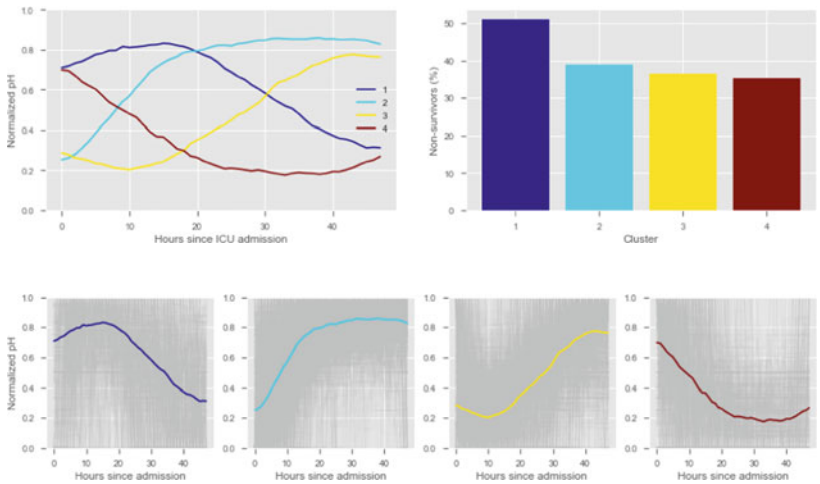
```
Using 1387 ICU stays for cluster assignment
4 clusters
Cluster 1: 653 observations
Cluster 2: 586 observations
Cluster 3: 574 observations
Cluster 4: 539 observations
```




```
temperature
Using 2352 ICU stays for creating the clusters
Using 1387 ICU stays for cluster assignment
4 clusters
Cluster 1: 472 observations
Cluster 2: 695 observations
Cluster 3: 561 observations
Cluster 4: 624 observations
```



```
pH
Using 2326 ICU stays for creating the clusters
Using 1363 ICU stays for cluster assignment
4 clusters
Cluster 1: 472 observations
Cluster 2: 789 observations
Cluster 3: 643 observations
Cluster 4: 422 observations
```



Clustering allowed us to stratify patients according to their physiological evolution during the first 48 h in the ICU. Since cluster centers reflect the cluster tendency, it is possible to investigate the relationship between distinct physiological patterns and mortality and ascertain to if the relationship is expected. For example, cluster 4 and cluster 5 in glucose are more or less symmetric: in cluster 4, patients start with low glucose, which increases over time until it decreases again; in cluster 5, patients start with high glucose, which decreases over time until it increases again. In the first case, mortality is approximately 45% and in the second case it is approximately 30%. Although this is obviously not enough to predict mortality, it highlights a possible relationship between the evolution of glucose and mortality. If a certain patient has a pattern of glucose similar to cluster 4, there may be more reason for concern than if they express the pattern in cluster 5.

By now, some particularities of the type of normalization performed can be noted:

- It hinders interpretability;
- It allows the algorithm to group together patients that did not present significant changes in their physiological state through time, regardless of the absolute value of the observations.

We have seen how clustering can be used to stratify patients, but not how it can be used to predict outcomes. Predictive models that use the information provided by clustering are investigated next. Models are created for the extracted features together with cluster information. This idea is represented in Fig. 10.1.

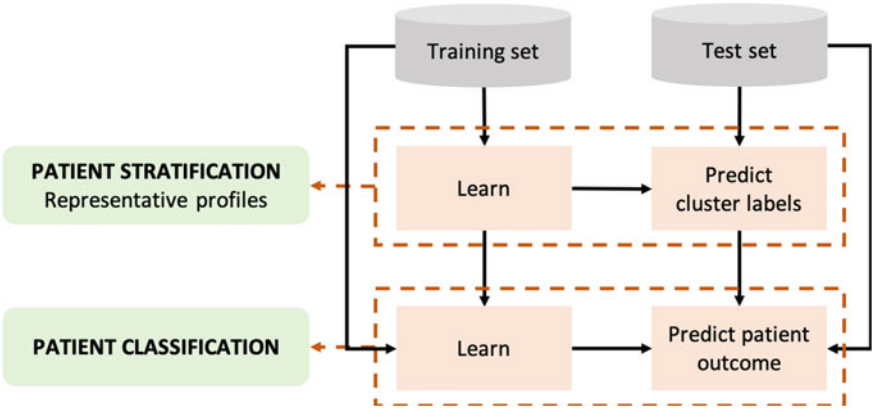


Fig. 10.1 Schematic representation of the machine learning steps

10.1.4 Normalization

Normalization, or scaling, is used to ensure that all features lie between a given minimum and maximum value, often between zero and one. The maximum and minimum values of each feature should be determined during the training phase and the same values should be applied during the test phase.

The next example is used to normalize the features extracted from the time series.

```
In [34]: X_train_min = X_train.min()
         X_train_max = X_train.max()
         X_train_norm = (X_train - X_train_min) / (X_train_max - X_train_min)
         X_test_norm = (X_test - X_train_min) / (X_train_max - X_train_min)
```

Normalization is useful when solving for example least squares or functions involving the calculation of distances. Contrary to what was done in clustering, the data is normalized for all patients together and not for each patient individually, i.e., the maximum and minimum values used for scaling are those found in the entire training set.

The next example uses the 'preprocessing' package from 'sklearn', which performs exactly the same:

```
In [35]: from sklearn import preprocessing

         min_max_scaler = preprocessing.MinMaxScaler()
         X_train_norm_skl = pd.DataFrame(min_max_scaler.fit_transform(X_train))

         # the same normalization operations will be applied to be consistent with the
         # transformation performed on the train data.
         X_test_norm_skl = pd.DataFrame(min_max_scaler.transform(X_test))
```

10.1.5 Concatenate Predicted Clustering Labels with Extracted Features

In the next example, the 'get_dummies' function from 'pandas' is used to get dummy variables for the cluster labels obtained through k-means. The idea is to use binary cluster labels, i.e., features indicating "yes/no belongs to cluster k", as input to the models. This will provide an extra level of information regarding the clinical temporal evolution of the patient in a multidimensional space.

You can add a 'drop_first' parameter to the 'get_dummies' function to indicate if you want to exclude one category, i.e., whether to get k–1 dummies out of k categorical levels by removing the first level. Because we will perform feature selection, this option does not need to be selected.

```
In [37]: # drop_first : bool, default False
         # use drop_first=True to exclude one of the categories

         X_train_clust = pd.get_dummies(cluster_labels_train, prefix_sep=' ')
         y_train_clust =
         pd.DataFrame(data_transf_inv.loc[cluster_labels_train.index]['mortality'])
         X_test_clust = pd.get_dummies(cluster_labels_test, prefix_sep=' ')
         y_test_clust = pd.DataFrame(data_transf_inv.loc[cluster_labels_test.index]['mortality'])

         X_train = pd.concat([X_train_norm, X_train_clust], axis=1).dropna(axis=0)
         y_train = y_train.loc[X_train.index]

         X_test = pd.concat([X_test_norm, X_test_clust], axis=1).dropna(axis=0)
         y_test = y_test.loc[X_test.index]
```

The next example prints the number of observations in the training and test sets, total number of features and a snapshot of the data.

```
In [38]: print('Number of observations in training set: ' + str(X_train.shape[0]))
         print('Number of observations in test set: ' + str(X_test.shape[0]))
         print('Number of features: ' + str(X_train.shape[1]))
         display.display(X_train.head())
```

```
Number of observations in training set: 2110
Number of observations in test set: 1232
Number of features: 85
```

icustay	min diastolic BP	min glasgow coma scale	min glucose	\
200019.0	0.370370	0.250000	0.461864	
200220.0	0.506173	0.250000	0.559322	
200250.0	0.506173	0.333333	0.387712	
200379.0	0.395062	0.000000	0.368644	
200488.0	0.617284	0.500000	0.283898	

icustay	min heart rate	min mean BP	min oxygen saturation	\
200019.0	0.347107	0.630719	0.950	
200220.0	0.752066	0.705882	0.970	
200250.0	0.685950	0.552287	0.960	
200379.0	0.570248	0.513072	0.930	
200488.0	0.561983	0.637255	0.895	

icustay	min respiratory rate	min systolic BP	min temperature	min pH	\
200019.0	0.366667	0.877551	0.760278	0.746479	
200220.0	0.300000	0.761905	0.643836	0.563380	

```
200250.0              0.016667         0.591837         0.739726  0.732394
200379.0              0.300000         0.605442         0.760278  0.788732
200488.0              0.100000         0.605442         0.842466  0.845070

             ...      CL systolic BP 2.0  CL systolic BP 3.0  \
icustay      ...
200019.0     ...                     0.0                 1.0
200220.0     ...                     1.0                 0.0
200250.0     ...                     1.0                 0.0
200379.0     ...                     0.0                 1.0
200488.0     ...                     0.0                 1.0

           CL temperature 0.0  CL temperature 1.0  CL temperature 2.0  \
icustay
200019.0                  1.0                 0.0                 0.0
200220.0                  1.0                 0.0                 0.0
200250.0                  0.0                 0.0                 0.0
200379.0                  0.0                 0.0                 1.0
200488.0                  0.0                 0.0                 1.0

           CL temperature 3.0  CL pH 0.0  CL pH 1.0  CL pH 2.0  CL pH 3.0
icustay
200019.0                  0.0        0.0        0.0        0.0        1.0
200220.0                  0.0        0.0        0.0        1.0        0.0
200250.0                  1.0        1.0        0.0        0.0        0.0
200379.0                  0.0        0.0        0.0        1.0        0.0
200488.0                  0.0        0.0        0.0        1.0        0.0

[5 rows x 85 columns]
```

The dataset is now composed of a mixture of summary statistics obtained through simple operations and clustering. Cluster labels are categorized as 'CL'. For example, 'CL 0.0' corresponds to cluster 1, 'CL 1.0' to cluster 2 and so on.

In the following "Part 3—Supervised Learning", classification models will be created in order to predict mortality.

Acknowledgements This work was supported by the Portuguese Foundation for Science & Technology, through IDMEC, under LAETA, project UID/EMS/50022/2019 and LISBOA-01-0145-FEDER-031474 supported by Programs Operational Regional de Lisboa by FEDER and FCT.