Multivariate Data Visualization with Animated Plotting

Fabio Ricardo Oliveira Bento
Programa de Pós-graduação em
Engenharia Elétrica
Universidade Federal do Espírito
Santo
Vitória - Espírito Santo - Brasil
fabio.obento@gmail.com

***Abstract*— Many pattern recognition problems involve thousands or even millions of features for each training sample. This scenario can not only make training extremely slow, but can also make it difficult to get a good solution. This problem is often referred to as the curse of dimensionality. Fortunately, in real-world problems, it is often possible to reduce the number of features considerably, turning a potential intractable problem into a tractable one. Apart from speeding up training, dimensionality reduction is also extremely useful for data visualization. In order to reduce the number of features to be processing, in the present work will apply both feature selection and feature extraction. In our experimental study will be used Recursive Feature Elimination (RFE) algorithm for feature selection. For feature extraction Principal Component Analysis(PCA), t-Distributed Stochastic Neighbor Embedding(t-SNE) and Radviz were used. The aforementioned techniques were applied on the Tennessee Eastman(TE) process dataset, and was obtained animated plotting visualization of multivariate data.**

I. INTRODUCTION

Reducing the number of dimensions down to two (or three) makes it possible to plot a condensed view of a high-dimensional training set on a graph and often gain some important insights by visually detecting patterns, such as clusters. Moreover, data visualization is essential to communicate your conclusions to people who are not data scientists, in particular decision makers who will use your results. Today Python boasts of a large number of powerful visualisation tools like Plotly [1], Bokeh [2], Altair [3] to name a few. These libraries are able to achieve state of the art animations and interactiveness. Nonetheless, the aim of this article is to highlight the Animations aspect, and we are going to look at some of the ways of doing that. Matplotlib [4] is a Python plotting library and also the most popular one. Most of the people start their Data Visualisation journey with Matplotlib. One can generate plots, histograms, power spectra, bar charts, error charts, scatterplots, etc easily with Matplotlib. It also integrates seamlessly with libraries like Pandas [5] and Seaborn [6] to create even more sophisticated visualisations. Animations are an interesting way of demonstrating a phenomenon. We as humans are always enthralled by animated and interactive charts rather than the static ones. Animations make even more sense when depicting time series data like sensors readings on an industrial plant, stock prices over the years, climate change over the past decade, seasonalities and trends since we can then see how a particular parameter behaves with time. Matplotlib's `animation` base class [7] deals with the animation part. It provides a framework around which the animation functionality is built. There are two main interfaces to achieve that:

- `FuncAnimation` makes an animation by repeatedly calling a function.
- `ArtistAnimation`: Animation using a fixed set of Artist objects.

However, out of the two, `FuncAnimation` is the most convenient one to use and, therefore it was used in this work. More information about then can be seen in the documentation [7]. Scikit-learn[8] provides the `sklearn.module`, whose classes can be used for feature selection/dimensionality reduction on high-dimensional datasets. Scickit-learn is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy[9]. In the present work the `RFE` class from this module was used, which implements the recursive feature elimination. For the purpose of dimensionality reduction were used the classes `sklearn.decomposition.PCA` and `sklearn.manifold.TSNE`. Aditionally the `plotting.radviz` Pandas implementation of Radviz multidimensional plot in 2D was used.

The present report was organized in six sections. The sections II and III are just some theoretical notes studied along the work. The section IV briefly describes the TE dataset. The final results of animations are in section V. Section VI presents some final discussions on multivariate animated data visualization.

This technical report and the full source code written are available at https://github.com/fabiobento/TE-multvariate-timeseries-view. Please refer to the this repository and feel free to download, reproduce the results and reuse the code[10].

II. FEATURES SELECTION

Datasets may be of a huge variety of sizes. Sometimes they have very few samples, and sometimes extremely large. In the case large datasets with multiple features, it becomes very complex and time-consuming to train our machine learning algorithms. Sometimes it might be even lead to overfitting and underfitting. One possible way to overcome these problems is applying feature selection. Often in high dimentional datasets(having multiple features) there are some irrelevant features which do not contribute in any way to predict the output,instead these features just increases the complexity of the algorithm. Therefore, such features cause a number of problems, which in turn prevents the process of efficient predictive modeling:

- Unnecessary resource allocation for these features.
- These features act as a noise for which the machine learning model can perform terribly poorly.
- The machine model takes more time to get trained.

One first approach in trying to overcome the issues described sofar is feature selection. Feature Selection can be basically described as technique to extract the most important features, having most profound impact on predicting output. There are many feature selection techniques,some of the most important ones are mentioned below.

A. Filter methods

Filter method relies on the general uniqueness of the data to be evaluated and pick a feature subset, not including any mining algorithm. Filter methods are generally used as a data preprocessing step. These methods provides a rank on the basis of statistical scores, which tend to determine the features' correlation with the output variable. Please refer to the Table I for defining correlation coefficients for different types of data (in this case continuous and categorical).

Feature\Response	Continuous	Categorical
Continuous	Pearson's Correlation	LDA
Categorical	Anova	Chi-Square

TABLE I
CORRELATION COEFFICIENTS FOR DIFFERENT TYPES OF DATA

B. Wrapper methods

As you can see in the Figure 1 , a wrapper method needs one machine learning algorithm and uses its performance as evaluation criteria. In this methods we use subset of features and then train a model[11].They "wrap" the feature selection around a specific classifier and select a subset of features based on the classifier's accuracy using cross-validation. This method is computationally very expensive as it becomes an exhaustive search problem. Some typical examples of wrapper methods are forward feature selection, backward feature elimination, recursive feature elimination([12][13]):

- **Forward Selection:** The procedure starts with an empty set of features(reduced set). The best of the original features is determined and added to the reduced set. At each subsequent iteration, the best of the remaining original attributes is added to the set.
- **Backward Elimination**: The procedure starts with the full set of attributes. At each step, it removes the worst attribute remaining in the set.
- **Combination of forward selection and backward elimination:** The stepwise forward selection and backward elimination methods can be combined so that, at each step, the procedure selects the best attribute and removes the worst from among the remaining attributes.
- **Recursive Feature elimination**: Recursive feature elimination performs a greedy search to find the best performing feature subset. It iteratively creates models and determines the best or the worst performing feature at each iteration. It constructs the subsequent models with the left features until all the features are explored. It then ranks the features based on the order of their elimination. In the worst case, if a dataset contains N number of features RFE will do a greedy search for 2N combinations of features.

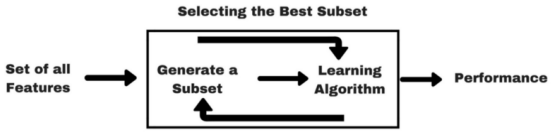

Fig. 1. Wrapper-based feature selection methods description.

C. Embedded methods

Embedded methods uses inbuilt machine learning algorithms to find best performing features(Figure 2). For example, random forest,decision tree,lasso regression,ridge regression. Embedded methods are iterative in a sense that takes care of each iteration of the model training process and carefully extract those features which contribute the most to the training for a particular iteration. Regularization methods are the most commonly used embedded methods which penalize a feature given a coefficient threshold. This is why Regularization methods are also called penalization methods that introduce additional constraints into the optimization of a predictive algorithm (such as a regression algorithm) that bias the model toward lower complexity (fewer coefficients). Examples of regularization algorithms are the LASSO, Elastic Net, Ridge Regression.

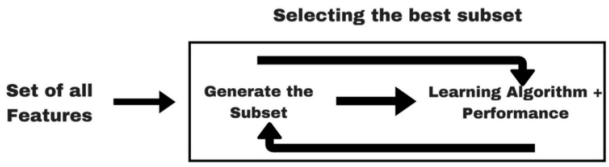

Fig. 2. Embedded methods for feature selection methods description.

D. Differentiation between filter and wrapper methods

It might get confusing at times to differentiate between filter methods and wrapper methods in terms of their functionalities. Let's take a look at what points they differ from each other.

Filter methods do not incorporate a machine learning model in order to determine if a feature is good or bad whereas wrapper methods use a machine learning model and train it the feature to decide if it is essential or not.

Filter methods are much faster compared to wrapper methods as they do not involve training the models. On the other hand, wrapper methods are computationally costly, and in the case of massive datasets, wrapper methods are not the most effective feature selection method to consider.

Filter methods may fail to find the best subset of features in situations when there is not enough data to model the statistical correlation of the features, but wrapper methods can always provide the best subset of features because of their exhaustive nature.

Using features from wrapper methods in your final machine learning model can lead to overfitting as wrapper methods already train machine learning models with the features and it affects the true power of learning. But the features from filter methods will not lead to overfitting in most of the cases

You may have already understood the worth of feature selection in a machine learning pipeline and the kind of services it provides if integrated. But it is very important to understand at exactly where you should integrate feature selection in your machine learning pipeline.

Simply speaking, you should include the feature selection step before feeding the data to the model for training especially when you are using accuracy estimation methods such as cross-validation. This ensures that feature selection is performed on the data fold right before the model is trained. But if you perform feature selection first to prepare your data, then perform model selection and training on the selected features then it would be a blunder.

If you perform feature selection on all of the data and then cross-validate, then the test data in each fold of the cross-validation procedure was also used to choose the features, and this tends to bias the performance of your machine learning model.

E. Features Selection with Scikit-Learn - RFE

The recursive REF method is implemented in Scikit-Learn library in the `sklearn.feature_selection.RFE` object[13].

```
from sklearn.feature_selection import RFE
# Create the RFE object and rank each feature:
estimator = SVC(kernel="linear", C=1)
selector = RFE(estimator=estimator,
    n_features_to_select=2,step=1)
selector.fit(X, y)
features_ranking = selector.ranking_
# Discover what are de "n" best features
n = 2
selected_features = np.ravel(np.where(
    features_ranking < 3))
```

Listing 1. Features Selection with Scikit-Learn - RFE

The `ranking` method of the `RFE` class(instantiated with the name "selector" in Listing 1 above,) return a ranking of the `n` best features that represent the current dataset.

III. FEATURE EXTRACTION

In this section we will discuss feature extraction and goes through two popular dimensionality reduction techniques: PCA and t-SNE. Additionally the Radviz , a multivariate data visualization algorithm, will be briefly described. Feature extraction involves reducing the number of resources required to describe a large set of data. It employs methods of constructing combinations of the variables, while still describing the data with sufficient accuracy.

A. Principal Component Analysis - PCA

Principal Component Analysis (PCA) is by far the most popular dimensionality reduction algorithm[14] First it identifies the hyperplane that lies closest to the data, and then it projects the data onto it, just like in Figure 3.

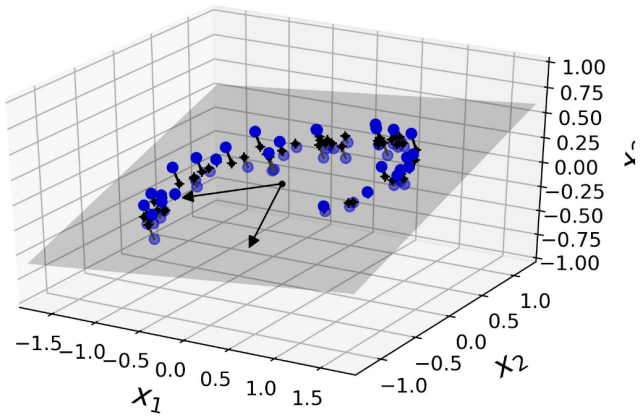

Fig. 3. A 3D dataset lying close to a 2D subspace.

1) Preserving Variance: Before project the training set onto a lower-dimensional hyperplane, first its necessary to choose the right hyperplane. For example, a simple 2D dataset is represented on the left of Figure 4, along with three different axes (i.e., one-dimensional hyperplanes). On the right is the result of the projection of the dataset onto each of these axes. As you can see, the projection onto the solid line preserves the maximum variance, while the projection onto the dotted line preserves very little variance, and the projection onto the dashed line preserves an intermediate amount of variance. It seems reasonable to select the axis that preserves the maximum amount of variance, as it will most likely lose less information than the other projections. Another way to justify this choice is that it is the axis that minimizes the mean squared distance between the original dataset and its projection onto that axis. This is the rather simple idea behind PCA.

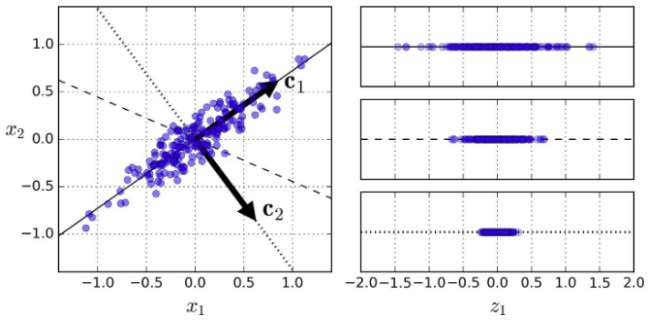

Fig. 4. Selecting the subspace onto wich to project

2) Principal Components: PCA identifies the axis that accounts for the largest amount of variance in the training set. In Figure 4, it is the solid line. It also finds a second axis, orthogonal to the first one, that accounts for the largest amount of remaining variance. In this 2D example there is no choice: it is the dotted line. If it were a higher-dimensional dataset, PCA would also find a third axis, orthogonal to both previous axes, and a fourth, a fifth, and so on (as many axes as the number of dimensions in the dataset). The unit vector that defines the i^{th} axis is called the i^{th} principal component (PC). In Figure 4, the 1^{st} PC is c_1 and the 2^{nd} PC is c_2 . In Figure 3 the first two PCs are represented by the orthogonal arrows in the plane, and the third PC would be orthogonal to the plane (pointing up or down). The principal components of a training set can be found with a standard matrix factorization technique called Singular Value Decomposition (SVD) that can decompose the training set matrix X into the matrix multiplication of three matrices $U\Sigma V^T$, where V contains all the principal components that we are looking for, as shown in Equation 1.

$$V = \begin{pmatrix} | & | & | \\ c_1 & c_2 & c_n \\ | & | & | \end{pmatrix} \tag{1}$$

PCA assumes that the dataset is centered around the origin, although, Scikit-Learn's PCA classes take care of centering the data.

3) Projecting Down to d Dimensions: Once identified all the principal components, the next step is to reduce the dimensionality of the dataset down to d dimensions by projecting it onto the hyperplane defined by the first d principal components. Selecting this hyperplane ensures that the projection will preserve as much variance as possible. For example, in Figure 3 the 3D dataset is projected down to the 2D plane defined by the first two principal components, preserving a large part of the dataset's variance. As a result, the 2D projection looks very much like the original 3D dataset. To project the training set onto the hyperplane, you can simply compute the matrix multiplication of the training set matrix X by the matrix W_d , defined as the matrix containing the first d principal components (i.e., the matrix composed of the first d columns of V), as shown in Equation 2.

$$X_{d-proj} = XW_d \tag{2}$$

4) Using Scikit-Learn for PCA: Scikit-Learn's PCA class implements PCA using SVD decomposition[15]. The following code applies PCA to reduce the dimensionality of the dataset down to two dimensions (note that it automatically takes care of centering the data):

```
from sklearn.decomposition import PCA
    pca = PCA(n_components = 2)
    X2D = pca.fit_transform(X)
```

Listing 2. Using Scikit-Learn for PCA

5) Explained Variance Ratio: Another very useful piece of information is the *explained variance ratio* of each principal component, available via the `explained_variance_ratio_` variable. It indicates the proportion of the dataset's variance that lies along the axis of each principal component. For example, let's look at the explained variance ratios of the first two components of the 3D dataset represented in Figure 3:

```
>>> pca.explained_variance_ratio_
array([0.84248607, 0.14631839])
```

Listing 3. Explained Variance Ratio with Scikit-Learn

This output reports that 84.2% of the dataset's variance lies along the first axis, and 14.6% lies along the second axis. This leaves less than 1.2% for the third axis, so it is reasonable to assume that it probably carries little information.

6) Selecting the proper Number of Dimensions: Instead of arbitrarily choosing the number of dimensions to reduce down to, it is generally preferable to choose the number of dimensions that add up to a sufficiently large portion of the variance (e.g., 95%). Unless, of course, you are reducing dimensionality for data visualization—in that case you will generally want to reduce the dimensionality down to 2 or 3. The following code computes PCA without reducing dimensionality, then computes the minimum number of dimensions required to preserve 95% of the training set's variance:

```
    pca = PCA()
    pca.fit(X_train)
    cumsum = np.cumsum(pca.
    explained_variance_ratio_)
    d = np.argmax(cumsum >= 0.95) + 1
```

Listing 4. Selecting the proper Number of Dimensions

You could then set `n_components=d` and run PCA again. However, there is a much better option: instead of specifying the number of principal components you want to preserve, you can set `n_components` to be a float between `0.0` and `1.0` , indicating the ratio of variance you wish to preserve:

```
    pca = PCA(n_components=0.95)
    X_reduced = pca.fit_transform(X_train)
```

Listing 5. Calculating the ratio of variance you wish to preserve

Yet another option is to plot the explained variance as a function of the number of dimensions (simply plot cumsum; see Figure 5). There will usually be an elbow in the curve, where the explained variance stops growing fast. You can think of this as the intrinsic dimensionality of the dataset. In this case, you can see that reducing the dimensionality down

to about 100 dimensions wouldn't lose too much explained variance.

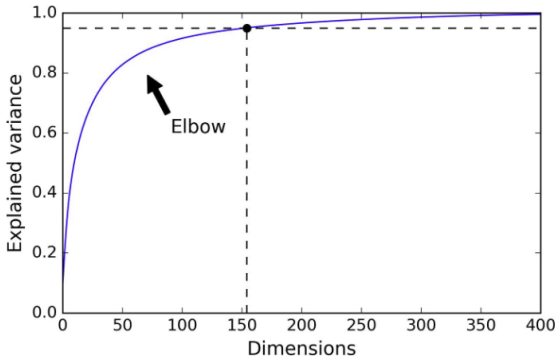

Fig. 5. Explained variance as a function of the number of dimensions

B. t-Distributed Stochastic Neighbor Embedding

This method reduces dimensionality while trying to keep similar instances close, and dissimilar instances apart[16]. It is mostly used for visualization, in particular to visualize clusters of instances in high-dimensional space.

1) Brief description of t-SNE: More specifically, t-SNE converts similarities between data points to joint probabilities and tries to minimize the Kullback-Leibler divergence between the joint probabilities of the low-dimensional embedding and the high-dimensional data. t-SNE has a cost function that is not convex, i.e. with different initializations we can get different results. It is highly recommended to use another dimensionality reduction method (e.g. PCA for dense data or TruncatedSVD for sparse data) to reduce the number of dimensions to a reasonable amount (e.g. 50) if the number of features is very high. This will suppress some noise and speed up the computation of pairwise distances between samples.
So far we have discussed that PCA is a good choice for dimensionality reduction and visualization for datasets with a large number of variables. But t-SNE might be a more suitable technique in cases where we search for patterns in a non-linear way. There are mainly two types of approaches we can use to map the data points:

- Local approaches : They maps nearby points on the manifold to nearby points in the low dimensional representation.
- Global approaches : They attempt to preserve geometry at all scales, i.e. mapping nearby points on manifold to nearby points in low dimensional representation as well as far away points to far away points.

t-SNE is one of the few algorithms which is capable of retaining both local and global structure of the data at the same time It calculates the probability similarity of points in high dimensional space as well as in low dimensional space. High-dimensional Euclidean distances between data points are converted into conditional probabilities that represent similarities[17]:

$$p_{j|i} = \frac{e^{\frac{-\left\|x_i - x_j\right\|^2}{2\times\sigma_i^2}}}{\sum_{k\neq i} e^{\frac{-\|x_i - x_k\|^2}{2\times\sigma_i^2}}} \tag{3}$$

where x_i and x_j are data points, $||x_i - x_j||$ represents the Euclidean distance between these data points, and i is the variance of data points in high dimensional space.
For the relatively lower-dimensional data points y_i and y_j corresponding to the high-dimensional data points x_i and x_j, it is possible to compute a similar conditional probability using:

$$q_{j|i} = \frac{e^{-\left\|y_i - y_j\right\|^2}}{\sum_{k\neq i} e^{-\|y_i - y_k\|^2}} \tag{4}$$

where $||y_i - y_j||$ represents the Euclidean distance between y_i and y_j. After calculating both the probabilities, it minimizes the difference between both the probabilities

2) Using Scikit-Learn for t-SNE: The t-SNE in Scikit-learn is available trough The `sklearn.manifold` module, which either implements others data embedding techniques. As usual, its very straightforward to use it.

```
from sklearn.manifold import TSNE
tsne = TSNE(compnts= 2, iter=30).fit_transform(X)
```

Listing 6. Feature extraction with t-SNE in Scikit-Learn

The `compnts` parameters defines the number of components in the transformed data. It is highly recommended to use another dimensionality reduction method (e.g. PCA for dense data or TruncatedSVD for sparse data) to reduce the number of dimensions to a reasonable amount (e.g. 50) if the number of features is very high. This will suppress some noise and speed up the computation of pairwise distances between samples. For more tips see Laurens van der Maaten's FAQ [18], and consult the guide at [19].

C. Radial Visualization (Radviz)

Radial visualizations are based on a spring tension minimization algorithm[20]. The features of the dataset are equally spaced on a unit circle and the instances are dropped into the center of the circle. The features then 'pull' the instances towards their position on the circle in proportion to their normalized numerical value for that instance.

1) Using Pandas for RadViz: In Python, the Radviz is available in Pandas library, in its *RadViz Visualizer* class[21]. RadViz is, indeed, a multivariate data visualization algorithm that plots each feature dimension (uniformly) around the circumference of a circle.Then it plots points on the interior of the circle such that the point normalizes its values on the axes from the center to each arc. This mechanism allows as many dimensions as will easily fit on a circle, greatly expanding the dimensionality of the visualization. Again, it is very simple to use.

```
import pandas as pd
rad_viz = pd.plotting.radviz(df, 'target')
```

Listing 7. Using Pandas for RadViz

IV. DESCRIPTION OF TENNESSEE EASTMAN (TE) PROCESS DATASET

The TE process is a simulation model of a chemical industrial system described by Downs and Vogel [22] . The TE process dataset is extensively used by researchers as a fault detection benchmark. Beside the produced data of normal condition, the model can generate samples from 21 faulty situations. Both normal and faulty samples have 52 variables(features) and are produced at a sampling time of 3 min. Specifically for the present work, we used the simulator to generate four simulation cases files, one for each of the (arbitrarily) chosen faults 1, 2, 4 and 6. Each file has 536 normal samples and 1606 faulty samples, in a total of 2142 samples.

V. RESULTS

This technical report and the full source code written are available at https://github.com/fabiobento/TE-multvariate-timeseries-view. Please refer to the this repository and feel free to download, reproduce the results e reuse the code[10].

In the implmentation, the TE data was loaded and normalized though mean removal and variance scaling.The animation of time series was implemented with the `animation.FuncAnimation` object from Matplotlib. A instance of this object is created for each animation. At each time step it receives as input (among others parameters):

- A initialization function to plot the background of each frame function at each time step (`init`)
- A animation function this is called sequentially(`animate`).

For features selection with RFE, the best features were selected, and the final result at the end of animation is in Figure 6. Note that the four different faults was clearly discriminated with only the two features selected. The first

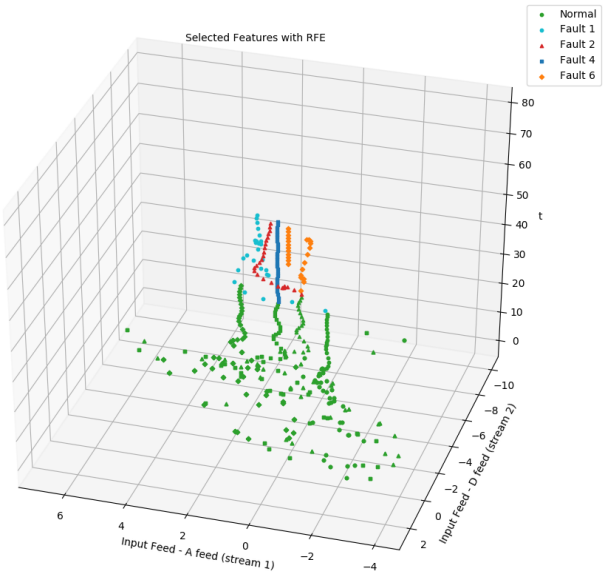

Fig. 6. Temporal evolution of selected features from TE dataset with RFE

method applied for features extraction was PCA. The two principal components time series are in Figure 7. The original aspect of data were preserved, with clear separation between classes of failures.

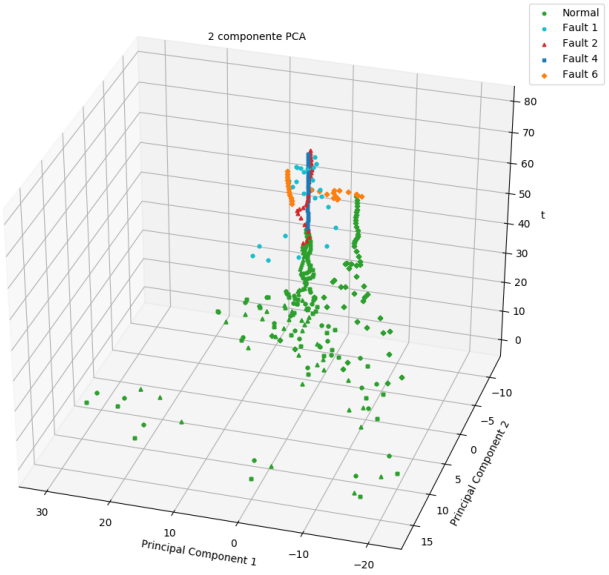

Fig. 7. Temporal evolution of extracted features(1^{st} and 2^{nd} components) from TE dataset with PCA

In the result for the t-SNE technique(Figure 8), there is some definite separation between the classes that are labeled. In the RadViz graph result(Figure 9) , we also can see

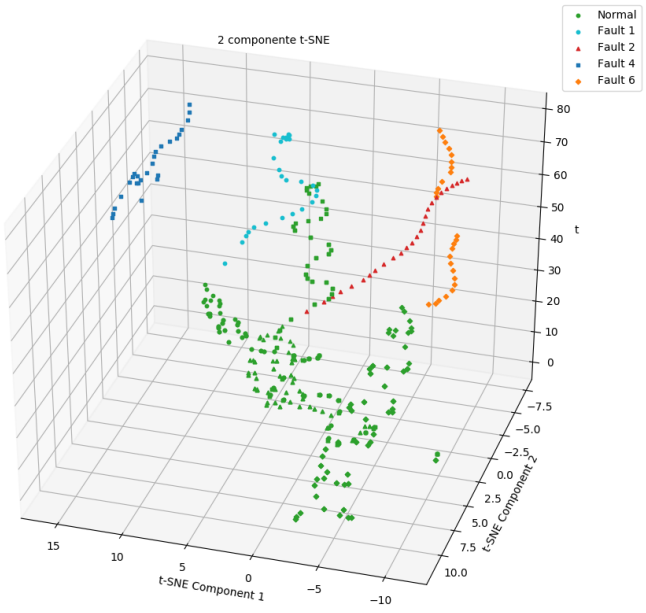

Fig. 8. Temporal evolution of extracted features(1^{st} and 2^{nd} components) from TE dataset with t-SNE

that there is some separation between the classes. Moreover, it appears that the same features previous selected with RFE(”Input Feed - A feed (stream 1)” and ”Input Feed - D feed (stream 2)’ seems to be the more predictive features,

given how strongly dots are being ‘pulled’ toward that part of the circle.

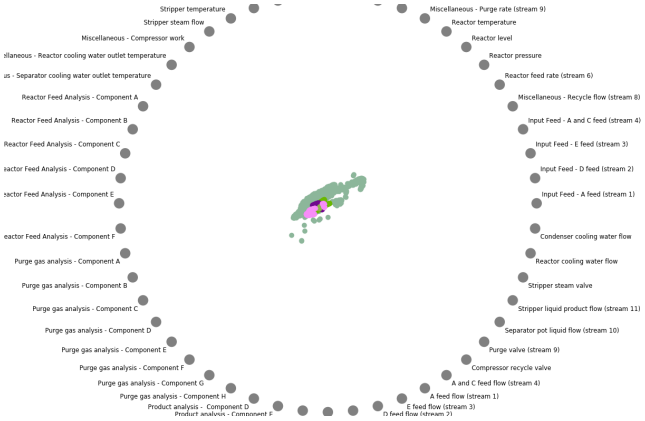

Fig. 9. Multivariate data visualization from TE dataset with Radial Visualization(RadViz)

VI. FINAL DISCUSSIONS

It was observed that animations help to highlight certain features of the visualisation which otherwise cannot be communicated easily with static charts. Having said that it is also important to keep in mind that unnecessary and overuse of visualisations can sometimes complicate things. Every feature in data visualisation should be used judiciously to have the best impact. Therefore, the importance of feature selection and dimensionality reduction for data visualization, resides in its ability to increase the effectiveness of results communication and responsiveness of interaction with the final user.

REFERENCES

[1] (2019) Modern Analytic Apps for the Enterprise plotly. [Online]. Available: https://plot.ly/

[2] (2019) Welcome to Bokeh bokeh 1.2.0 documentation. [Online]. Available: https://bokeh.pydata.org/en/latest/

[3] (2019) Altair - Declarative Visualization in Python altair 3.1 documentation. [Online]. Available: https://altair-viz.github.io/

[4] (2019) Matplotlib - Python Plotting matplotlib 3.1.1 documentation. [Online]. Available: https://matplotlib.org/

[5] (2019) Python Data Analysis Library - pandas python data analysis library. [Online]. Available: https://pandas.pydata.org/

[6] M. Waskom. (2019) seaborn: statistical data visualization seaborn 0.9.0 documentation. [Online]. Available: https://seaborn.pydata.org/

[7] (2019) matplotlib.animation matplotlib 3.1.0 documentation. [Online]. Available: https://matplotlib.org/3.1.0/api/animation_api.html

[8] (2019) scikit-learn: machine learning in Pythonscikit-learn 0.21.2 documentation. [Online]. Available: https://scikit-learn.org/

[9] (2019) Scipy.org/. [Online]. Available: https://scipy.org/

[10] F. R. O. Bento. (2019) Multivariate Data Visualization with Animated Plotting of Tennessee Eastman Dataset. [Online]. Available: https://github.com/fabiobento/TE-multvariate-timeseries-view.git

[11] T. M. Phuong, Z. Lin, and R. B. Altman, “Choosing snps using feature selection,” in *2005 IEEE Computational Systems Bioinformatics Conference (CSB’05)*. IEEE, 2005, pp. 301–309.

[12] S. Khalid, T. Khalil, and S. Nasreen, “A survey of feature selection and feature extraction techniques in machine learning,” in *2014 Science and Information Conference*, Aug 2014, pp. 372–378.

[13] (2019) sklearn-feature selection-RFE. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.RFE.html

[14] A. Geron, *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. OŔeilly Media, Incorporated, 2019.

[15] (2019) sklearn.decomposition.PCA - scikit-learn 0.21.2 documentation. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html

[16] L. v. d. Maaten and G. Hinton, “Visualizing data using t-sne,” *Journal of machine learning research*, vol. 9, no. Nov, pp. 2579–2605, 2008.

[17] S. Jaj. (2019) Comprehensive Guide on t-SNE algorithm with implementation in R Python. [Online]. Available: https://www.analyticsvidhya.com/blog/2017/01/t-sne-implementation-r-python/

[18] L. van der Maaten. (2019) t-SNE - Laurens van der Maaten. [Online]. Available: https://lvdmaaten.github.io/tsne/

[19] (2019) sklearn.manifold.TSNE - scikit-learn 0.21.2 documentation. [Online]. Available: https://scikit-learn.org/stable/modules/manifold.html#t-sne

[20] L. Zhou and D. Weiskopf, “Multivariate visualization of particle data,” *The European Physical Journal Special Topics*, vol. 227, no. 14, pp. 1741–1755, 2019.

[21] (2019) RadViz Visualizer - yellowbrick 0.9.1 documentation plotly. [Online]. Available: https://www.scikit-yb.org/en/latest/api/features/radviz.html

[22] J. J. Downs and E. F. Vogel, “A plant-wide industrial process control problem,” *Computers & chemical engineering*, vol. 17, no. 3, pp. 245–255, 1993.