`

4.1 Signal De-noising

Process measurements are inevitably contaminated with high-frequency noise. If not dealt with appropriately, noise can sometimes result in errors in model parameter estimation or unfavorable characteristics in predicted variables. For example, in Figure 4.1, we can see the noisy fluctuations in a flow measurement signal. These kinds of fluctuations are generally not real process variations and are just an artifact of measurement sensors. If not removed, these may result in noisy fluctuations in predicted variables, let's say product purity, which would be very undesirable.

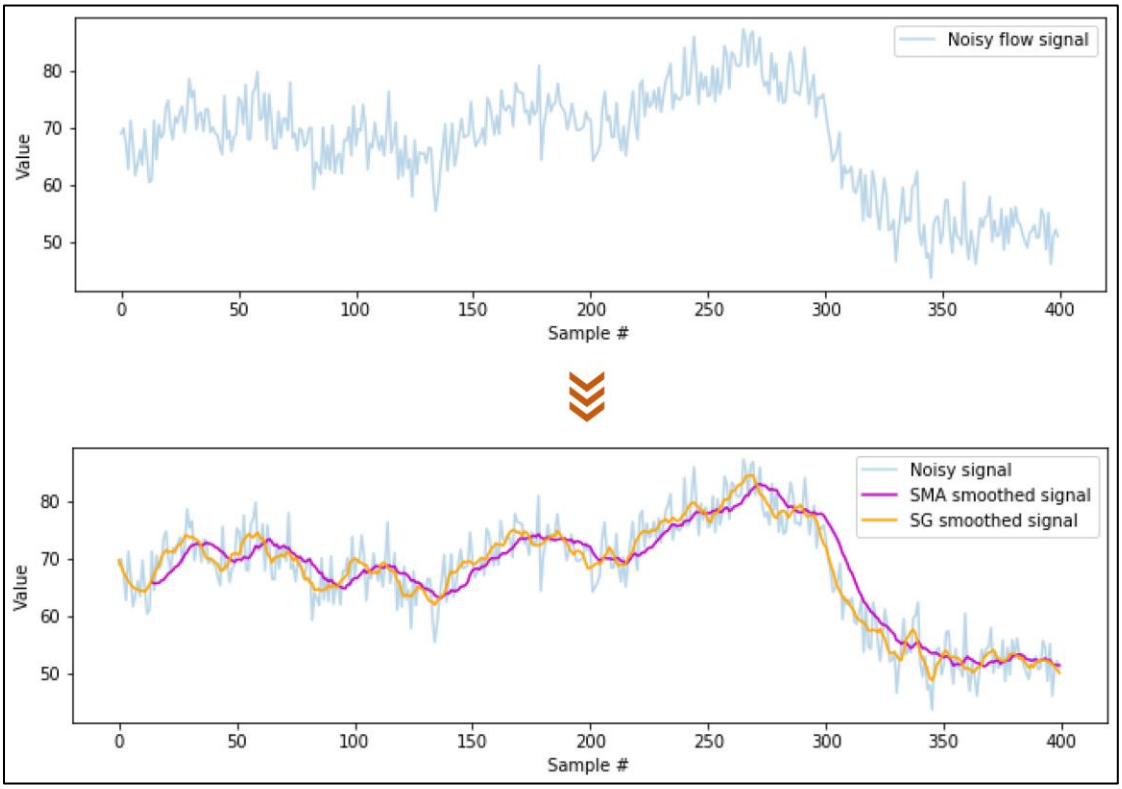

Figure 4.1: Time series signal smoothed by simple moving average (SMA) and SG filters

Two common ways to de-noise process signals is to smooth them using moving window averages and Savitzky-Golay (SG) filtering. Figure 4.1 shows the smoothed flow signals. We can see that crucial process variations have been preserved while noisy fluctuations have been removed! Let's quickly study the two techniques and learn how to implement them.

Moving window average filter

In moving window average, the smoothed value of a variable *x* at time *t* is taken as a weighted combination of previous measurements of *x* in a window of finite or infinite length. In the most

`

common variant, simple moving average (SMA), the combination is a simple average as shown below

$$x(t)_{smoothed} = \frac{\sum_{j=0}^{W-1} x(t-j)_{raw}}{W}$$

Where *W* is the window size. To generate the SMA smoothed signal in Figure 4.1, Pandas was utilized as follows

```
# read noisy flow data
import numpy as np
noisy_signal = np.loadtxt('noisy_flow_signal.csv', delimiter=',')

# denoise using SMA filter
import pandas as pd
windowSize = 15
smoothed_signal_MA = pd.DataFrame(noisy_signal).rolling(windowSize).mean().values # taking mean on a rolling window and converting back to numpy array
```

Alternatives to SMA are LWMA (linearly weighted moving average) and EWMA (exponentially weighted moving average) where the past measurements are weighted in a finite and infinite length window, respectively. The value of *W* in finite window length variants depend on 'how much' smoothing is desired and is guided by the knowledge of the underlying process system (window size should ideally be smaller than the period over which systematic process variations occur) or through trial and error. As would be obvious, larger *W* achieves more smoothing.

SG filter

In SG filtering, the smoothed value at any point *t** is realized by approximating the values of *x* in an odd-sized window centered at *t** with a polynomial of time (t) and then evaluating the polynomial at *t**. Specifically, at point *t**, a m^{th} order polynomial of the following form is fitted and then evaluated as

$$x(t^*)_{smoothed} = \sum_{k=1}^{m} b_k (t^*)^k$$

where b_k are the estimated polynomial coefficients. In Figure 4.1, SG filter was implemented using Scipy with 2nd order polynomials as follows

```
# import savgol_filter from scipy
from scipy.signal import savgol_filter
smoothed_signal_SG = savgol_filter(noisy_signal, window_length = 15, polyorder = 2)
```

`

Note that, under the default setting, the (*W*-1)/2 values at the extreme edges are handled by fitting a polynomial to the *W* values at the edges and evaluating this polynomial to get smoothed values for the (*W*-1)/2 data-points nearest to the edges. You are encouraged to check the official documentation[13] for other available settings.

SG filter enjoys the advantage of preserving the important features of the raw signals better than moving window average filters. SMA has another drawback which is apparent in Figure 4.1 - if we observe closely around sample 300 where a step change in flow occurs, the SMA smoothed signal shows a small time-delay/offset unlike SG smoothed signal which follows the original signal very closely during the step change. This time offset can become problematic for certain problems. SMA, however, enjoys computational advantage as it is faster to compute.

SMA and SG filters belong to the category of low-pass filters which block high frequency component of raw signals and allow low frequencies to pass through. This effectively removes spurious fast transitions, spikes, and noise while retaining the slowly evolving systematic variations. On the other hand, high-pass filters block low frequency components and allow high frequencies. These are often utilized to remove long-term slow drifts from process signals.

There are other advanced methods as well like wavelet filters, LOWESS smoothing which can be explored if SMA or SG filters don't provide satisfactory performance. Irrespective of the method deployed, you should pay due attention tuning the filters to ensure that the smoothed signal is acceptable around edges, transients, and step changes in the original signal.

4.2 Variable Selection / Feature Selection

Imagine that you are tasked with building a regression (or a classifier) model for predicting product quality in a large petrochemical plant. For such complex and highly integrated processes, it would not be surprising if you don't have enough process insights about which variables effect product quality and therefore, you end up with hundreds of variables as potential inputs to your model. The problem with inclusion of irrelevant inputs is that model training (and update) time increases, the model becomes prone to overfitting, and there is a greater requirement of training samples to adequately represent the model input space. Moreover, unnecessarily bulky input set makes physical interpretation of the model difficult.

[13] https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.savgol_filter.html

`

The task of selecting the most relevant subset of inputs/feature is called variable selection / feature selection. A trivially optimal way of variable selection is to explore all possible combinations of input variables and select the one that gives the best performance on the validation dataset. However, this can be computationally infeasible when dealing with large number of input candidates. To circumvent these issues, several variable selection methods have been devised which try to find a good subset of relevant model inputs. Figure 4.2 lists some of these methods that are commonly employed. In the next few subsections, we will learn these and try to understand their strengths and drawbacks.

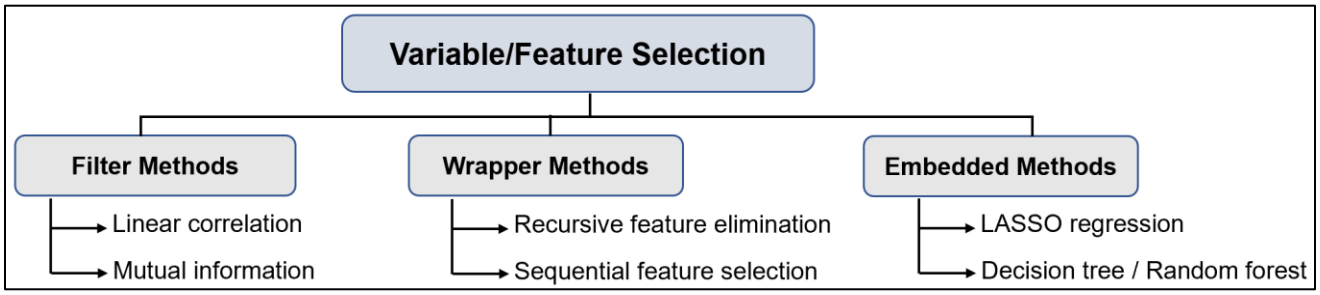

Figure 4.2: An overview of variable selection methods

As you can see, the methods are divided into three categories depending on the employed search strategy for the relevant inputs. Filter methods use statistical measures such as correlation coefficients, mutual information to quantify the relevance of any input w.r.t. the target variable. Wrapper methods utilize the model to quantify the predictive power of different input subsets via metrics like MSE, BIC, etc. In the third category, the embedded methods perform variable selection during the process of model fitting itself. Before we delve deeper into these methods, let's take a quick look at the dataset we will use in this section.

Illustration dataset

The dataset is provided in the file VSdata.csv and has been simulated using a mechanism (with slight modifications) devised by Chong & Jun[14].The mechanism was designed to ensure that the dataset mimics a real manufacturing process with interconnected unit processes. There are 40 input variables out of which 10 (inputs 17 to 26) are known to be relevant while the rest are irrelevant to the target variable. Fitting and validation dataset contain 1000 and 250 samples, respectively. The target samples were generated using the relevant inputs as follows

$$y_{sample} = \sum_{j=17}^{26} \beta_j x_{sample,j} + \varepsilon_{sample}$$

[14] Ching & Jun, Performance of some variable selection methods when multicollinearity is present, Chemometrics and intelligent laboratory systems, 2005

`

where ε_{sample} is measurement noise. All the inputs are generated from a multivariate normal distribution with zero mean and covariance matrix $\boldsymbol{\Gamma}$ which is designed to induce strong correlations among neighboring process inputs.

$$\Gamma_{ij} = 0.9^{|i-j|}, \qquad (i, j = 1,2,\cdots,40)$$

Figure 4.3 shows the target data in the training dataset and the input data. From these plots, it is not immediately clear whether an input is (more) relevant for prediction of the target or not. A multivariate linear regression with all the inputs gives a R^2 value of 0.614 on the validation dataset, while using only 10 relevant inputs results in slightly better R^2 of 0.634. Let us see if our variable selection techniques can correctly find the relevant inputs in the dataset.

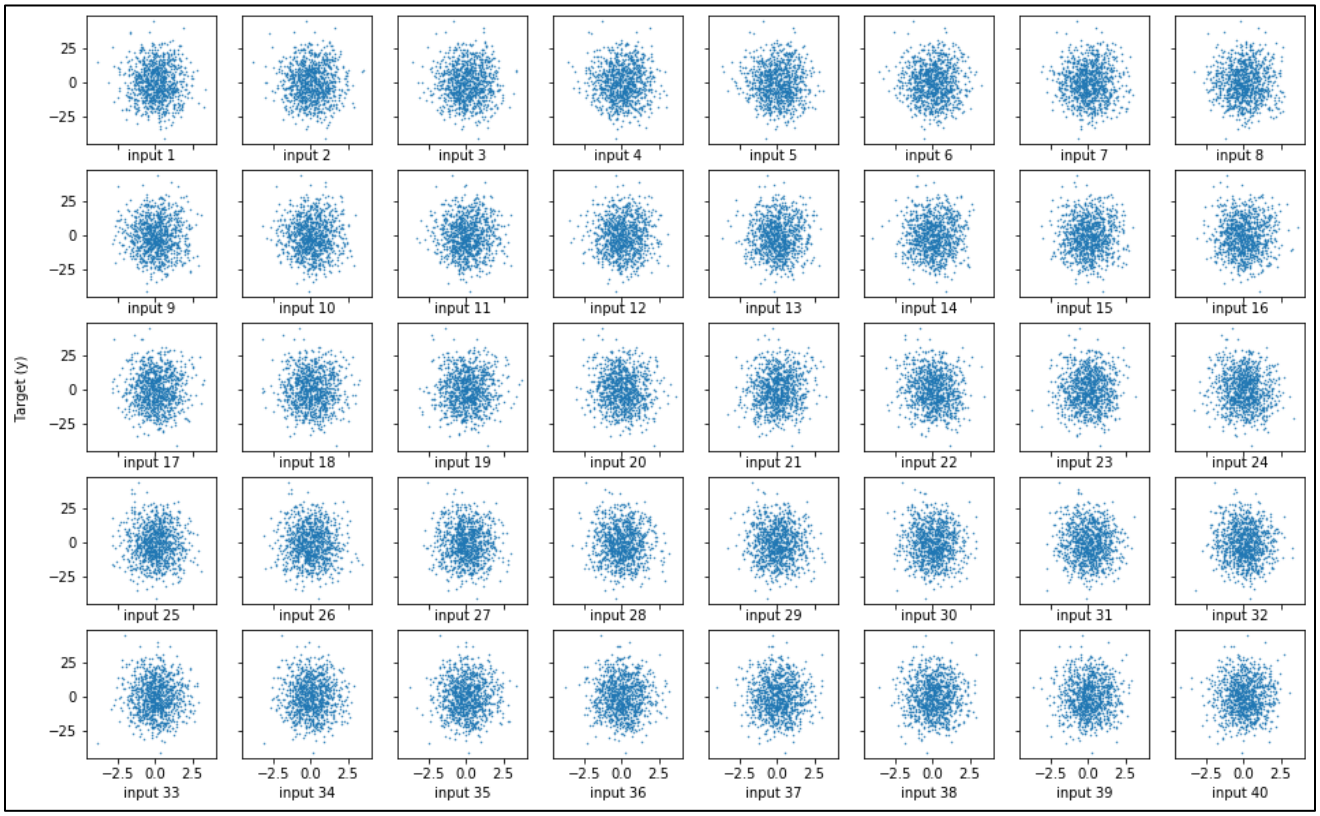

Figure 4.3: Target vs input plots for a simulated manufacturing process dataset

`

Filter methods

In the most common form of filter methods, the relationship between each input and the target is estimated using some statistical measure and then the inputs are ranked according to the estimated relationship strengths. Once ranked, the top ranked variables can be picked. Figure 4.4 shows an overview of the strategy.

Figure 4.4: Filter method strategy for feature selection

Let's look at the tow commonly used statistical measures.

Correlation coefficients

For continuous inputs and target, the common statistical measure is Pearson correlation coefficient which measure the linear correlation between an input and the target and is given by

$$R_{xy} = \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_i (x_i - \bar{x})^2 \sum_i (y_i - \bar{y})^2}}$$

For variable selection, the top *k* variables with highest correlations, where *k* is known beforehand or decided via cross-validation, or all variables with correlations significantly different from zero can be selected.

Mutual information

Correlation coefficients can be deceptive for nonlinear dependencies between target and inputs and can lead to wrong variable selections. The solution is to use 'mutual information (MI)' metric which can efficiently quantify any kind of dependency between variables. Mathematically, MI is given by

$$MI\ (x,y) = \iint p(x,y) log \frac{p(x,y)}{p(x)p(y)} dxdy$$

where denotes $p(x,y)$ joint probability density of variables *x* and *y* and denotes the marginal probability density of x. From process data, these densities can be estimated using *k*-nearest neighbor method (used by Sklearn), KDE, or histograms.

`

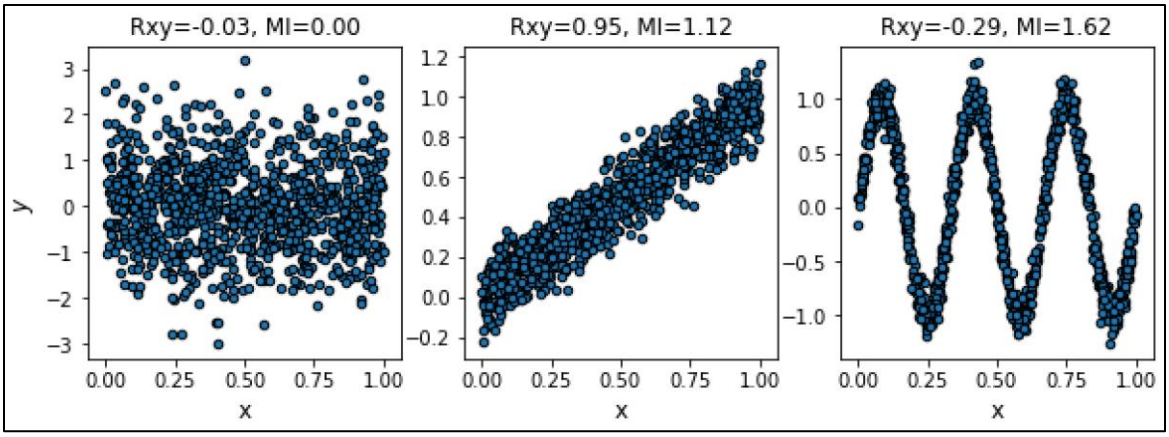

Figure 4.5: Linear correlation and mutual information metrics for zero, linear, and nonlinear dependencies between target and input

Consider the scenarios in Figure 4.3. Pearson correlation coefficient fails to identify the nonlinear (sinusoidal) dependency between target and input, but mutual information is able to determine the dependencies correctly. Colloquially speaking, MI measures the amount of information about a variable (*y*) that is provided by another variable (*x*). Therefore, if a target and an input are independent then no information about *y* can be obtained from knowing *x* and therefore, their MI would be 0. On the other hand, if *y* depends on *x,* then MI > 0 and higher value implies higher dependency.

As you would have noticed, filter methods do not entail building any predictive models and therefore are fast to execute. Filter methods are particularly favored when the number of input candidates are very large. However, the quality of 'relevant' input subset may be low. This is because filter methods consider the inputs in isolation and don't consider the interactions among the inputs. An input that is ranked low may provide significant boost in predictive accuracy when combined with some other input. Therefore, these methods are often used as a first step of variable selection to only remove very low ranked variables and the remaining variables are further screeded via other more sophisticated methods.

Sklearn implementation

Sklearn provides f_regression and mutual_info_regression methods to compute the statistical metrics we need. f_regression computes linear correlation between each input variable and target which is then converted into an F score. F scores come from F-test (you may remember from your statistics classes) which does a statistical check on whether the difference in modeling errors between 2 linear models (one relating target to just a constant and the other to an input and a constant) are significant or just happened by chance. If this does not ring any bell, don't worry. It suffices to know that the ranking obtained through F scores is the same as that obtained using correlation coefficients.

`

f_regression can be combined with SelectKBest transformer to conveniently transform raw input data matrix into one containing relevant inputs only as shown below

```
# read data
import numpy as np
VSdata = np.loadtxt('VSdata.csv', delimiter=',')

# separate X and y
y = VSdata[:,0]
X = VSdata[:,1:]

# compute linear correlation-based scores
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_regression

VSmodel = SelectKBest(f_regression, k=10).fit(X, y)
input_scores = VSmodel.scores_

# find the top ranked inputs
top_k_inputs = np.argsort(input_scores)[::-1][:10] + 1
#  [::-1] reverses the array returned by argsort() and [:n] gives that last n elements

# reduce X to only top relevant inputs
X_relevant = VSmodel.transform(X)
```

For our simulated dataset, inputs 22, 21, 24, 20, 30, 29, 32, 31, 28, 27 got selected as the top 10 relevant variables, i.e., only 4 inputs got correctly picked. We can see that the selection is not that good, primarily due to the multivariable dependence of our target. Let's see if wrapper methods can do any better.

Wrapper methods

Figure 4.6 gives an overview of wrapper approach to feature selection. Here, unlike filter methods, a predictive model is used to find the relevant features. The method begins with an initial selection of feature subset and enters a loop. In each cycle of the loop, a new model is built with the currently selected subset and the decision to include/exclude a feature from the subset is based on model's performance or some model attributes. The specific strategy employed distinguishes the different sequential methods. The loop concludes once the desired number of relevant features are obtained or some criteria on model performance is met. Note that the model performance may be assessed using a validation dataset or fitting dataset itself using BIC/AIC metrics. Let's study a couple of well-known wrapper methods.

`

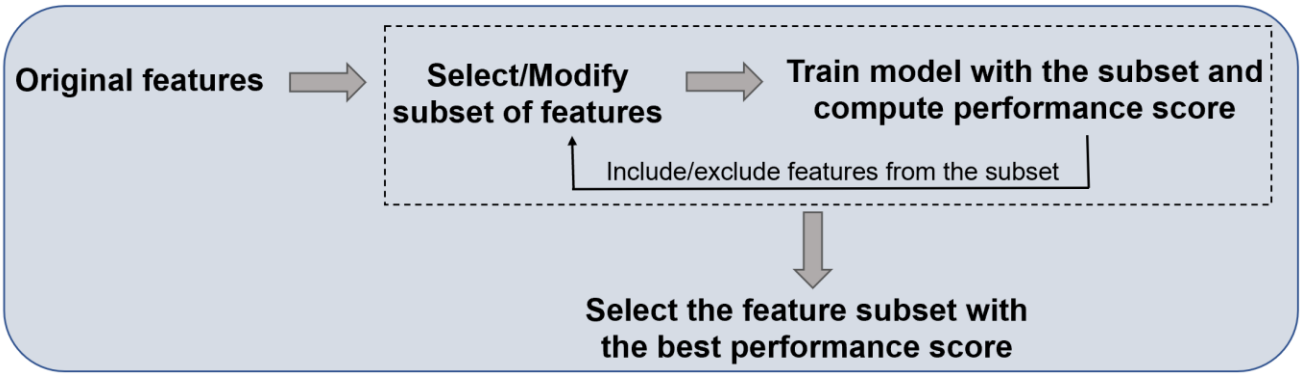

Figure 4.6: Wrapper method strategy for feature selection

Recursive feature elimination (RFE)

In this method, a model is initially built using all the input variables. The importance of each input is quantified using any specific model attribute (such as model coefficients) or through some combination of model attributes. A few of the inputs with the least importances are discarded and the entire process is repeated until the desired number of relevant inputs are obtained or model performance begins to deteriorate.

PLS is a very popular regression method for process systems which we will study in detail in Chapter 5. We are mentioning this here because PLS allows for computation of VIP (variable importance in projection) scores that quantify the importance of each input. PLS VIP[15]-based RFE algorithm can be used to iteratively exclude inputs that contribute the least to the prediction of the target variable.

Sequential feature selection (SFS)

Sequential feature selection technique has two common variants. In forward SFS, the search begins with an empty subset. The input that leads to the best performance when the model is trained on this single input gets added to the subset. Next, the previously selected subset is combined with each of the remaining inputs one at a time. The combination of two inputs that gives the best performance becomes the selected subset. This procedure is repeated till *k* inputs are selected or predictions no longer improve by adding new variables.

The other variant, backward SFS, works similarly as forward SFS, except that it starts with all the inputs. The input whose exclusion leads to the best model performance is excluded from the subset. This procedure is repeated to exclude inputs one by one.

[15] PLSRegression class in sklearn does not have any built-in VIP computation. pyChemometrics package can be used which provides a ChemometricsPLS class with a VIP method

`

Since wrapper methods use models, they outperform filter methods. However, because they execute multiple model fittings, they can become computationally infeasible for very large number of inputs and complex models (such as ANNs). Between forward and backward SFS, backward SFS takes into account the interactions effects of all the variables better than the forward SFS but can entail higher number of model fittings when *k* is much lower than the number of input candidates.

Sklearn implementation

Sklearn provides RFE and SequentialFeatureSelector classes that transform raw input data matrix into relevant subset using any specified estimator/model. Below is an implementation for backward SFS using linear regression model.

```
# scale data
from sklearn.preprocessing import StandardScaler
xscaler = StandardScaler()
X_scaled = xscaler.fit_transform(X)

yscaler = StandardScaler()
y_scaled = yscaler.fit_transform(y[:,None])

# implement backward SFS
from sklearn.feature_selection import SequentialFeatureSelector
from sklearn.linear_model import LinearRegression

BSFS = SequentialFeatureSelector(LinearRegression(), n_features_to_select=10, direction='backward',
cv=5).fit(X_scaled, y_scaled)

# check selected inputs
print('Inputs selected: ', BSFS.get_support(indices=True)+1)

>>> Inputs selected:  [18 19 20 21 22 23 24 25 31 33]

# reduce X to only top relevant inputs
X_relevant = BSFS.transform(X)
```

We can see that wrapper method has performed much better than filter method and has recovered 8 out of 10 relevant inputs! Another impressive thing to highlight here is how SequentialFeatureSelector allows condensing all the complexities of SFS variable selection with cross-validation into just a couple of lines of code!!

`

Embedded methods

Embedded methods make use of algorithms where selection of variables is an inherent part of the model fitting process itself. We already saw one of the methods in the previous chapter, Lasso regression, where irrelevant inputs are eventually removed from model by assigning them zero coefficients. Decision trees and random forests are another set of algorithms which directly provide feature importances after model fitting. We will study them in Chapter 9.

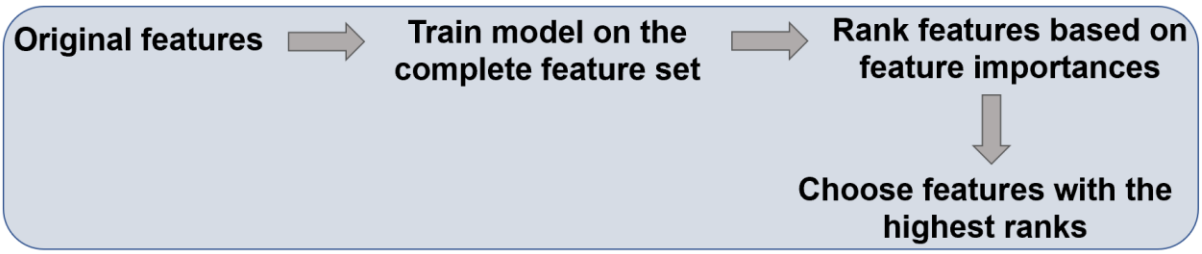

Figure 4.7: Embedded method strategy for feature selection

Embedded methods are computationally less expensive than wrapper methods and work best when the number of samples are much higher than the number of inputs. Below is a quick implementation of Lasso regression for our simulated dataset where we use the model coefficients to judge any input's importance.

Sklearn implementation

We will employ LassoCV model which automatically selects a penalization strength using cross-validation. As shown, Lasso has recovered 8 out of 10 relevant inputs.

```
# fit Lasso model
from sklearn.linear_model import LassoCV
Lasso_model = LassoCV(cv=5).fit(X_scaled, y_scaled)

# find the relevant inputs using model coefficients
top_k_inputs = np.argsort(abs(Lasso_model.coef_))[::-1][:10] + 1
print('Relevant inputs: ', top_k_inputs)

>>> Relevant inputs:  [21 22 20 23 24 19 25 18 33 14]
```

This concludes our look into the variable selection methods. The presence of nonlinearity, input interactions, computational resource availability, and the number of input candidates influence the eventual method selection. You are now familiar with the mechanism of these methods which should help you choose the method most suitable for your problem.

`

Feature extraction and feature selection are similar in the sense that both achieve feature dimension reduction. Feature extraction creates new features by using all the original features and therefore does not help in simplification of original measurement space. The focus is more on dimensionality reduction and removal of correlation among the features. Feature selection, as we have seen, discards irrelevant features altogether. Here the focus is on improved model performance with simpler physical interpretation due to low number of model inputs.

4.3 Outlier Handling

Outliers are abnormal observations that are inconsistent with the majority of the data. Presence of outliers in the training dataset results in a model that does not adequately represent the true process and therefore negatively impacts its prediction accuracy. It is imperative to remove outliers before model fitting or use a modeling algorithm that is robust to outliers.

Several methods are at our disposal - the choice depends on whether we are dealing with univariate or multivariate system, the degree of outlier contamination, Gaussian or non-Gaussian data distribution, etc. Figure 4.8 lists some of these methods that are commonly employed.

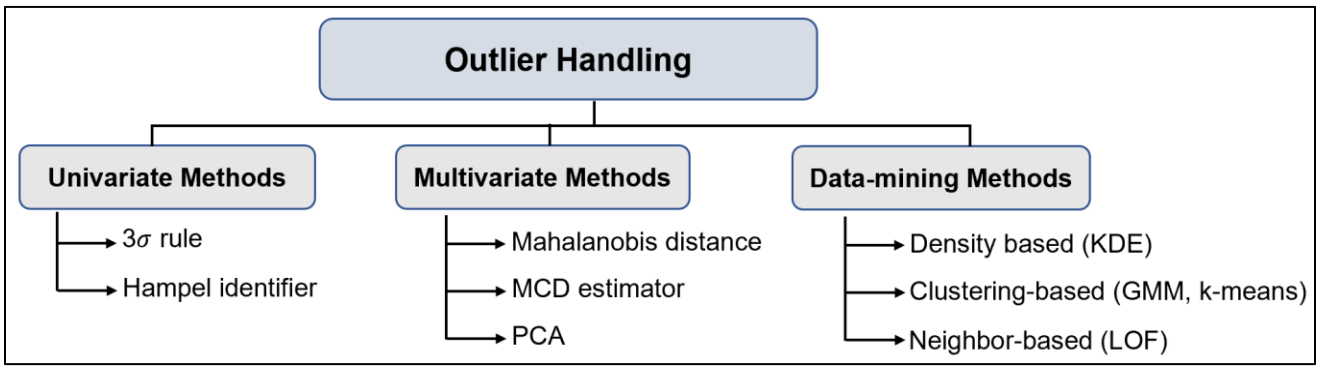

Figure 4.8: An overview of variable selection methods

Univariate methods

Univariate methods clean each variable separately. The 3-sigma rule and Hampel identifier are the popular methods available under this category.

`

3σ rule

The 3-sigma rule works as follows: if μ and σ denote the mean and standard deviation of a variable, then any sample beyond the range $\mu \pm 3\sigma$ are considered as outliers. This rule follows from the properties of a Gaussian-distributed variable for which 99.87% of data lie within the $\mu \pm 3\sigma$ range. However, by using a factor other than 3, this rule is often used for non-Gaussian variables as well.

Hampel identifier

You will agree than the 3σ rule is quite simple and easy to implement. Expectedly, it has some limitations. If a variable is severely contaminated with outliers, then μ and σ can get impacted to such an extent that abnormal samples end up falling within the $\mu \pm 3\sigma$ range and fail to be identified as outliers. The solution is to use Hampel identifier which replaces μ and σ with median and MAD, respectively. For any variable x, Hampel identifier tags an observation as an outlier if it lies outside the range $median(x) \pm 3\sigma_{MAD}$ where $\sigma_{MAD} = 1.4826 * median(|x - median(x)|)$.

We worked with median and MAD in the previous chapter for robust scaling. Using the same dataset, Figure 4.9 now illustrates the effectiveness of Hampel identifier for univariate outlier detection. The 'normal range' computed using 3-sigma rule is inflated enough to include all the data within the normal range! Hampel identifier, however, provides a normal range which is more representative of the normal data and therefore can flag the abnormal samples as outliers. We can see that Hampel identifier provides quite a superior performance with very minimal additional complexity.

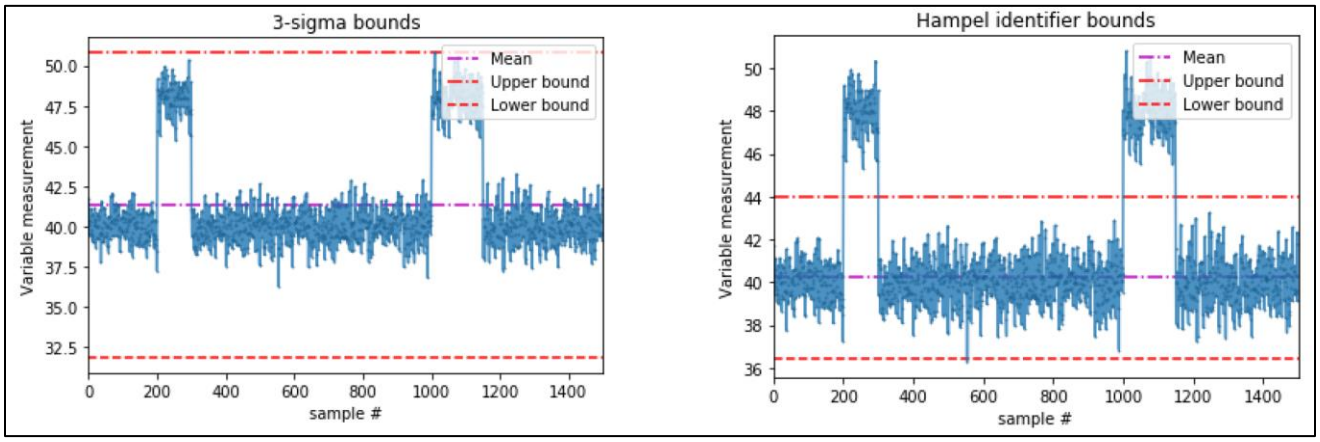

Figure 4.9: Univariate bounds obtained with 3-sigma and Hampel identifier rules for data contaminated with outliers

`

Multivariate methods

In process dataset where variables are highly correlated, outliers can escape detection when variables are 'cleaned' separately. For example, consider Figure 4.10. The red samples are clearly outliers in the 2D space. However, if we look only along the *x1* (or *x2*) axis alone, these abnormal samples lie close to the center of the data distribution, and therefore will not be considered outliers in univariate sense.

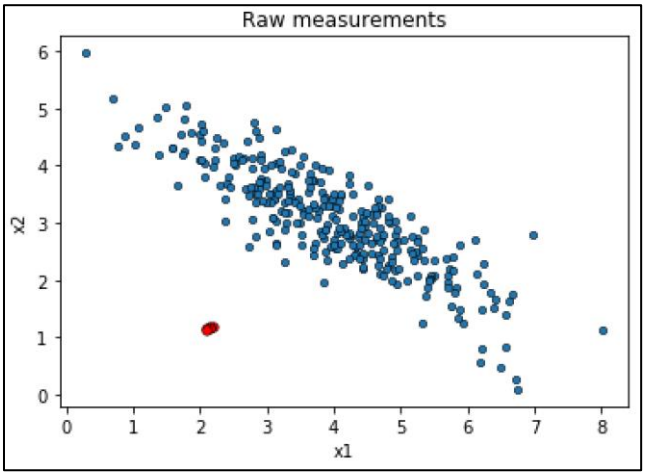

Figure 4.10: Outliers in 2D space difficult to detect via univariate methods

The solution is to use multivariate methods of outlier detection. These methods take into consideration the shape of multivariate data distribution and are often based on using some distance metrics to flag samples that are far away from the center of data distribution. Let's study these methods in detail.

Mahalanobis distance

Mahalanobis distance (MD) is a classical multivariate distance measure that takes into account the covariance/shape of data distribution for computing distances from the center of data. Specifically, MD of any observation $\boldsymbol{x}_n$ is given by

$$MD(\boldsymbol{x}_n) = \sqrt{(\boldsymbol{x}_n - \overline{\boldsymbol{x}})\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}_n - \overline{\boldsymbol{x}})^T}$$

where $\overline{\boldsymbol{x}}$ and $\boldsymbol{\Sigma}$ are mean and covariances of the sampled observations. The presence of $\boldsymbol{\Sigma}$ differentiates MD from Euclidean distance. Computation of MD essentially converts a multivariate outlier detection problem into a univariate problem. For illustration, Figure 4.11 shows the MDs for the dataset from Figure 4.10. We can see that the outlier samples are clearly very distinct from the normal samples and can be easily flagged via univariate techniques.

`

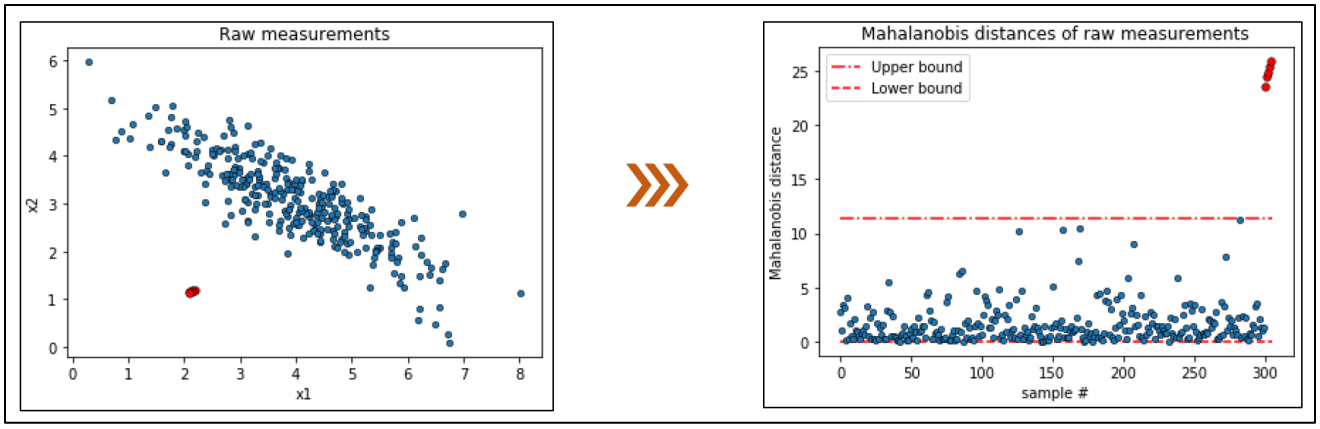

Figure 4.11: Multivariate outlier detection via Mahalanobis distance and Hampel identifier

In this illustration, the shown bounds were obtained via Hampel identifier on the cubic roots of MDs (cubic root taken to make the MDs distribution approximately Gaussian[16]) as shown in the code below

```
# read data
import numpy as np
data_2Doutlier = np.loadtxt('simple2D_outlier.csv', delimiter=',')

# compute Mahalanobis distances and transform into Gaussian distribution using cubic-root
from sklearn.covariance import EmpiricalCovariance
emp_cov = EmpiricalCovariance().fit(data_2Doutlier)
MD_emp_cov = emp_cov.mahalanobis(data_2Doutlier)
MD_cubeRoot = np.power(MD_emp_cov, 0.333)

# find Hampel identifier bounds
from scipy import stats
median = np.median(MD_cubeRoot)
sigma_MAD = stats.median_absolute_deviation(MD_cubeRoot)

upperBound_MD = np.power(median+3*sigma_MAD, 3)
lowerBound_MD = np.power(median-3*sigma_MAD, 3)

# plot Mahalanobis distances with bounds (last 5 samples are the outliers)
plt.figure(), plt.plot(MD_emp_cov[:-5], '.', markeredgecolor='k', markeredgewidth=0.5, ms=9)
plt.plot(np.arange(300,305), MD_emp_cov[-5:], '.r', markeredgecolor='k', markeredgewidth=0.5, ms=11)
plt.hlines(upperBound_MD, 0, 305, colors='r', linestyles='dashdot', label='Upper bound')
plt.hlines(lowerBound_MD, 0, 305, colors='r', linestyles='dashed', label='Lower bound')
```

[16] Wilson & Hilferty, The distribution of chi-square, Proceedings of the National Academy of Sciences of the United States of America, 1931

`

MCD estimator

Consider the data distribution and its Mahalanobis distances in Figure 4.12. Here again a bunch of samples lie away from the normal data, but as shown, the MD-based method fails miserably in detecting all the outliers. So, what happened here?

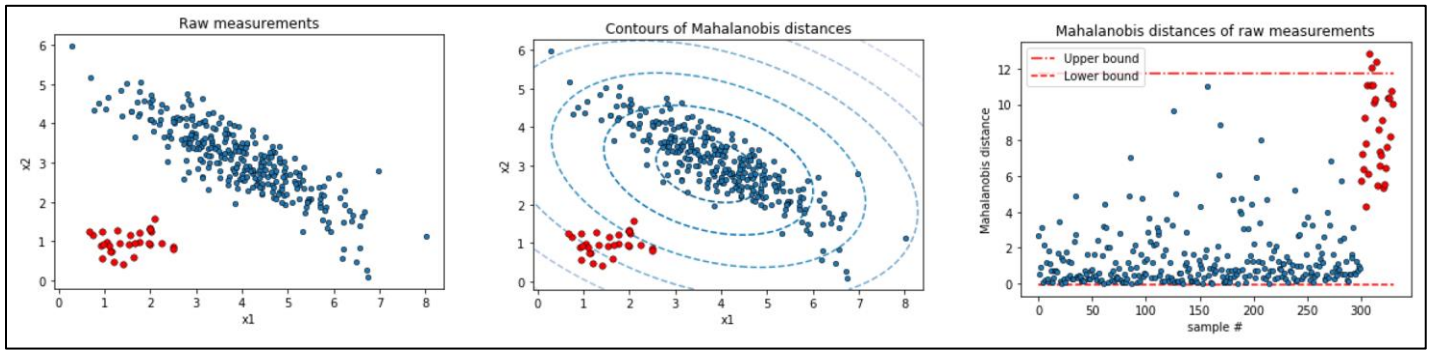

Figure 4.12: : Outliers in 2D space difficult to detect via classical Mahalanobis distance method

What's happening here is that the outliers are distorting the estimated shape/covariance structure of the data distribution. As shown in the middle subplot, the contours of MDs have been inflated to such an extent that MDs of several outliers are similar to those of inliers. This unwanted effect is called masking effect. Moreover, if inliers get classified as outliers due to the impact of outliers, it is called swamping effect.

Previously, in univariate methods, we saw that a robust outlier detection solution can be obtained by utilizing robust estimates of data location and spread. Similar concept exists for multivariate methods. A popular estimator of location (centroid) and scatter (covariance matrix) is MCD (Minimum Covariance Determinant). MCD finds *h* samples out of the whole dataset for which the covariance matrix has lowest determinant, or, simply speaking, the enveloping ellipsoid for a given confidence level has lowest volume among all possible subsets of size *h*. The centroid and covariance of *h* samples are returned as robust estimates.

The MinCovDet class in Sklearn implements a FAST-MCD algorithm which can efficiently analyze large datasets with high dimensionalities. The superior performance of this method is validated in Figure 4.13. The MD contours now fit the inlying samples much better and the outliers are well beyond the computed upper bound. Note that Sklearn provides a EllipticEnvelop class that uses FAST-MCD to directly detect an outlier but requires specification of data contamination level which may not be known beforehand. The code below shows the computation of MCD-based MDs; bound are computed as before.

```
# read data
import numpy as np
```

`

```
data_2Doutlier = np.loadtxt('complex2D_outlier.csv', delimiter=',')

# compute MCD-based Mahalanobis distances
from sklearn.covariance import MinCovDet

MCD_cov = MinCovDet().fit(data_2Doutlier)
MD_MCD = MCD_cov.mahalanobis(data_2Doutlier)
```

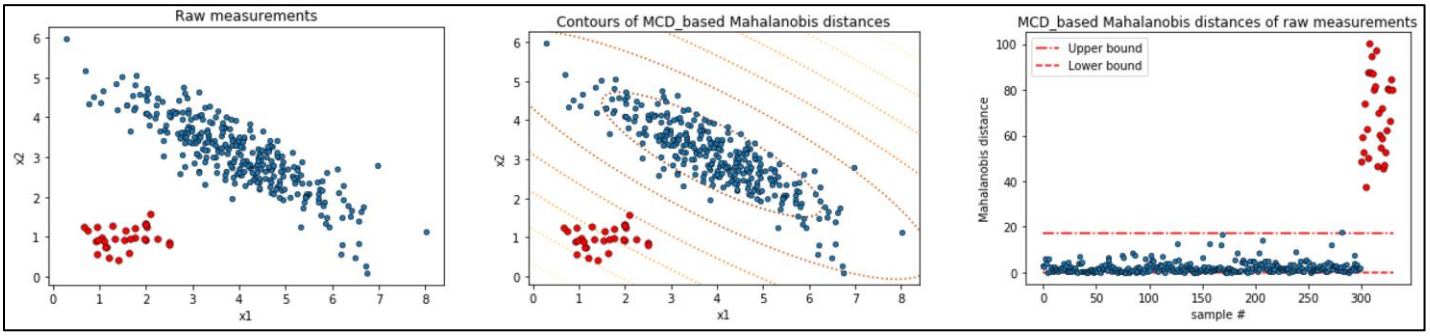

Figure 4.13: Multivariate outlier detection via MCD-based robust Mahalanobis distances

PCA

Not all outliers are bad and warrant removal. For example, consider Figure 4.14. Here, two kinds of outliers are highlighted. While samples *A* tend to break away from the majority correlation, samples *B* follow the correlation but lie far away from the majority of data.

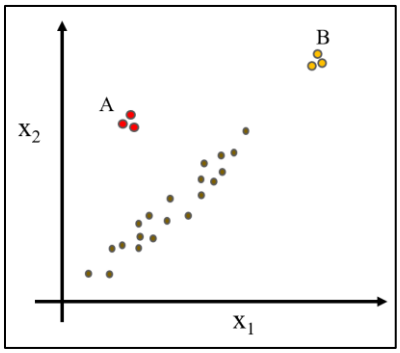

Figure 4.14: Different types of outliers which can be distinguished via PCA

Depending on the context of our problem, we may or may not want to remove samples *B*. The MD-based approach won't distinguish between these two kinds of outliers and therefore, won't work for us if we want to retain samples *B*. In that case, PCA (principal component analysis) can be employed. We will learn PCA in detail in Chapter 5. There you will understand how PCA distinguishes between these two different kinds of outlying observations. Classical PCA is based on sample covariance of data. Therefore, if the given dataset is highly contaminated with outliers, MCD-based covariance can be used in PCA for robust analysis.

`

Data-mining methods

The multivariate methods we studied in previous section make an implicit assumption about Gaussianity or unimodality of data distribution. In real life, it is not rare to encounter situations where these assumptions do not hold. One such data distribution is shown in Figure 4.15, where 3 distinct clusters of data is present with outliers in between them. MD or PCA-based approach will fail here!

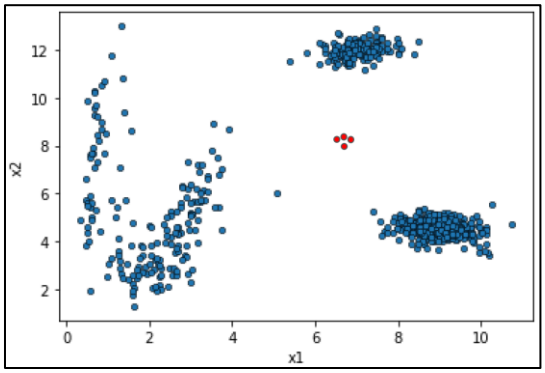

Figure 4.15: Complex multivariate data distribution with outliers

Advanced data-mining methods are employed to handle these complicated situations. The methods can be categorized into density-based, clustering-based, and neighbor-based. Density-based methods (like KDE) rely on estimation of distribution density of the data; observations lying in low-density regions are flagged as outliers. Cluster-based methods (like GMM, Kmeans) find distinct clusters of data; samples belonging to small-sized or low-importance clusters are flagged as outliers. Neighbor-based methods (like KNN, LOF) rely on finding the neighbors of each data-point. Neighboring data-points are used to compute any sample's distance from its k-nearest neighbor or the local density; samples with large distances or low densities are flagged as outliers.

We will study most of these techniques in Part 2 of the book. Once you study their nuances, you will realize that these are very powerful and useful methods but do require careful selection of their intrinsic model parameters such as the number of clusters in Kmeans, number of neighbors in LOF, or the kernel function & its width in KDE.

Time-series or dynamic dataset require different approaches for outlier detection. A common practice is to first model the given dataset using an appropriate time-series modeling technique such as ARIMA or ARX and compute residuals between model prediction and given data. With properly chosen model, residuals will be approximately normally distributed and uncorrelated. Then, any traditional outlier detection method can be applied on the residuals. The outliers will tend to have large residuals and will get flagged.

`

Outlier-resistant robust techniques

The outlier-handling method we studied till now relied on explicit identification of outliers. Once identified, these outliers are removed from model fitting dataset before model parameter estimation. There is another set of methods, called robust methods*+, which directly use outlier-contaminated data for model fitting. These algorithms are designed such that the effect of outliers is reduced by down-weighting them and only inliers primarily determine the model parameters.

Figure 4.16 illustrates an application of robust linear regression using RANSAC algorithm in Sklearn. The robust method is able to successfully ignore the outliers to find the correct fit between target and predictor.

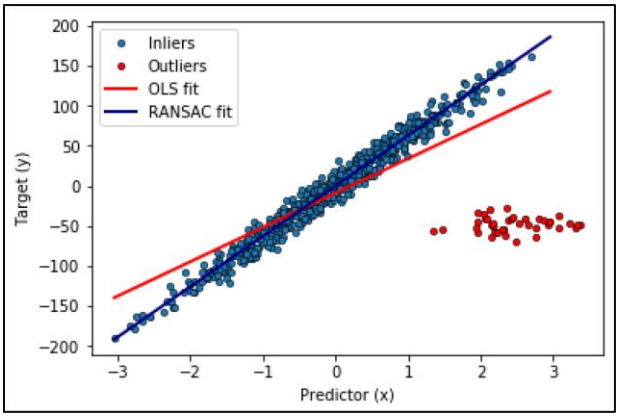

Figure 4.16: Robust model fitting with outlier-infested data

IRPLS (iteratively re-weighted PLS) is another popular robust regression method using PLS. Most of the outlier-resistant robust methods employ iterative algorithm where model parameters are computed iteratively and samples with high residuals identified in each iteration are assigned lower weights.

*Zhu et. al., Review and big data perspectives on robust data mining approaches for industrial process modeling with outliers and missing data, Annual Reviews in Control, 2018

+ Frosch + Bro, Robust methods for multivariate data analysis, Journal of Chemometrics, 2005

This concludes our study of methods for outlier detection. Our objective was to help you understand the pros and cons of different techniques because there is no universal method applicable to all problems. A good understanding of the underlying process and the nature of data can greatly help make the right choice of technique.

`

4.4 Handling Missing Data

Missing data issues arise when values of a variable or multiple variables are missing in one or more observations of the dataset. The samples with missing data can simply be discarded if their number is much smaller than the total number of available samples. However, if this is not the case or there are reasons to believe that discarding samples will negatively impact model fitting, imputation techniques can be employed which attempt to fill-in the missing values of an variable using available data from the same variable or other variables.

A simple imputation strategy is mean imputation where the missing values of a variable are replaced with the mean of all available data for the variable and is illustrated in the code snippet below

```
# Mean imputation
import numpy as np
from sklearn.impute import SimpleImputer

sample_data = [[1, 2, np.nan], [3, 4, 3], [np.nan, 6, 5], [8, 8, 7]]
mean_imputeModel = SimpleImputer(missing_values=np.nan, strategy='mean')

print(mean_imputeModel.fit_transform(sample_data))

>>> [[1. 2. 5.]
     [3. 4. 3.]
     [4. 6. 5.]
     [8. 8. 7.]]
```

Another widely used approach is hot-deck imputation, where missing values of a sample are replaced from the available values of 'similar' samples. Sklearn provides KNNImputer class which imputes missing value with the weighted (or mean) combination from k-nearest neighbors in the fitting dataset. The distance metric between 2 samples is computed using features that neither samples are missing.

```
# KNN imputation
from sklearn.impute import KNNImputer

knn_imputeModel = KNNImputer(n_neighbors=2)
print(knn_imputeModel.fit_transform(sample_data))
```

`

```
>>> [[1.  2.  4. ]
     [3.  4.  3. ]
     [5.5 6.  5. ]
     [8.  8.  7. ]]
```

Although not currently available in Sklearn, more advanced techniques like regression imputation, likelihood-based imputation also exist. The former method imputes the missing values in a variable using a regression model with other correlated variables as predictors. The later method uses expectation-maximation (EM) algorithm for imputing missing values in a probabilistic framework.

As with variable selection and outlier handling methods, it cannot be said beforehand which missing data technique will work best for your problem. A good recourse is to select a few methods based on your process knowledge, build separate models with them, and select the best performing one!

Summary

In this chapter, we studied the techniques commonly employed to handle different types and degree of data contamination. We looked into the issues of measurement noise, irrelevant features, outliers, and missing data. A natural pre-processing workflow entails denoising the dataset first, followed by variable selection and outlier removal. With this, we conclude the first leg of our ML journey. With the fundamentals in place, let’s start the second phase of the journey where we will look into several classical and very useful ML algorithms.