Session – Summary

Data Preparation – Session 2

The first stage of data preparation is called data cleaning which we covered in the first session. Another aspect of data preparation is called feature engineering, where the data set is made more informative by manipulating the features in the data. Feature engineering is as important as data cleaning but should always be done after data cleaning. In this session, you learnt the feature engineering step of the data preparation process.

After the data is cleaned, it can be made more informative by feature engineering which has the following subparts as discusses in this session –

1. Feature Extraction
2. Feature Transformation
3. Feature Standardisation and Conversion
4. Dimensionality Reduction

Feature Extraction

Feature extraction refers to the process where more meaningful features are defined based on the existing ones in the data set. The new features you extract should relate more to the business problem you are trying to solve.

Suppose you want to predict the sales of a superstore for September 2016. However, if the sales data is given to you on a daily basis i.e. sales per day, then you would extract a new feature from this data set which would be average sales per month.

In R the commands for feature extraction depend on the type of feature you want to extract-

- To calculate the average you can use the mean() function.
- To change a feature's unit i.e. cm to m
  - df$colname <- df$colname/number
  - df$colname <- df$colname*number
- To divide a feature into categories

To extract good features, you must have good data and business understanding.

Feature Transformation

It sometimes happens that the dependent variables or the features exhibit some trend. This trend is generally figured out by plotting the dependent variables against the target variable as a scatter plot and looking at the spread of the points. The scatter plot is then analysed to figure out that trend. These trends are not good for analyses as they make the variables and their underlying relationships harder to interpret. Therefore, if a variable shows an underlying trend when plotted against the target variable, then it should be made linear. Transformations such as logarithmic, square root, square, cube, inverse, etc. are generally used to get rid of the respective trends.

Where to use transformations:

- If the data right skewed, then log or square root transformation can be used.
- For left skewed data 'square' transformation is used.
- Inverse is used to make small data points bigger and big data points small.

In R, the following commands can be used for transforming variables:

- Log Transformation - df$colname <- log(data, 2)
  - The 2 represents natural log i.e. log to the base 2.
- Square root transformation - df$colname <- sqrt(df$colname)

Feature Standardisation and Conversion

Feature Standardisation:

When you have numeric features in a data set then often it happens that their range varies a lot from each other. One variable might take small values, and the other might take large values. If you apply a machine learning algorithm on such a data set then the variable which has large values might have a disproportionate effect on the algorithm in some cases. Therefore, it is imperative that the features are standardised to a similar scale so as to nullify any disproportionate effects. This is known as feature standardisation. In this session, you learnt that the most common way of standardising a feature is by z score normalisation. Z scores are basically normal scores which you learnt in statistics as well. Generally they lie between -3 and 3. Feature normalisation also reduces the computational time of certain algorithms.

For a column V containing values v1,…,vn, the z score normalisation for any value v is given as

- Z-score = [v – mean(V)] / std. dev(V)

In R, the following command can be used for scaling variables

- df<- scale(df[, colnumber]) or,
- df$colname <- scale(df$colname)

Feature Conversion:

Feature Conversion refers to the process where either numeric features are converted into categorical variables or categorical variables are converted into numeric variables. You would have to do the later more often. Feature are converted keeping the business problem and the type of algorithm being used in mind. Algorithm specific feature conversion will be covered along with the algorithms, but generally, some algorithms do not work on categorical data and in that case converting categorical data into numeric data becomes necessary.

Categorical to numeric conversion can be done in two ways.

- Straight up factor encoding: Categories are directly encoded as factors. If a variable has four categories then here they will be represented as 1,2,3,4 directly.
- Dummy encoding: Rather than directly encoding categories as factors, dummy variables are created which can take values 0 or 1. For a variable having n categories, n-1 dummy variables are created.

Dummy encoding is always better than straight up factor encoding because in dummy encoding we do not lose the relative difference between the categories. In straight up factor encoding the categories become 1,2,3,4 etc. which for the algorithm has a difference of 1 unit each but in reality, might have unequal gaps. Dummy encoding takes care of this problem because it tells us if a category is present or not by 0 and 1 encoding.

In R, the following command can be used for converting categorical data into numerical

- Straight up factor encoding
  - df$colname <- as.factor(df$colname)
- Dummy encoding
  - dummy<-as.data.frame(model.matrix(colname-1, data= data)

The model.matrix command is used for converting categorical variables into dummy variables. The ‘-1’ adjacent to the column name in the model.marix command, tells the command to lose the intercept column it gives by default. The object dummy here would still have columns equal to the number of categories. Therefore, while merging dummy variables to the final data frame, you will have to delete any one column.

Numeric to Categorical Conversion:

The most famous way of converting a numeric variable into a categorical variable is called equal-width binning. Here, the variable is divided into bins of equal width and frequency is assigned to that bins based on the number of data points in each one of them.

If you want to divide the variable into N bins then;

- Bins = B-A/N
- Where B is the largest value, and A is the smallest value.

In R, the following command can be used to perform equal-width binning

- bins<- table(discretize(x, categories=5)
- where x is df$colname ie. the variable and categories denote the number of categories you want to divide it into.

Dimensionality Reduction

Dimensionality reduction is a useful technique while handling large data sets containing hundreds or thousands of variables. As the name suggests, it helps us reduce the number of variables in the data set. Dimensionality reduction can be achieved through a technique called Principal Component Analysis (PCA).

Principal Component Analysis:

Principal component analysis transforms a set of possible correlated variables into a smaller number of uncorrelated variables called principal factors. Those principal factors are such that they explain the entire data set. Principal factors are then plotted against each other, and the factors explaining the maximum variability are then chosen for the analysis. Principal factors are easier to analyse and interpret. PCA can only be applied to numeric data.

The objectives of PCA are:

- Reduce the number of variables i.e. reduce the dimensionality of the data
- And to identify new underlying meaningful variables.

.

In R, the following commands can be used perform PCA:

- pca=prcomp(df, scale. = T, center = T)
- transData = as.data.frame(pca$x)
- plot(pca, type = "l")

The first command prcomp() will calculate an object of class prcomp which along with principal factors will have a few other components. Always remember to keep the argument scale and centre as TRUE if the numeric data is not normalised.

The second command extracts the principal factors and stores them as a data frame in an abject transData.

The third command plots the principal factors among each other and the principal factors showing the maximum dip show the maximum variability in the data. The factors accounting for the maximum variability are then selected for analysis.

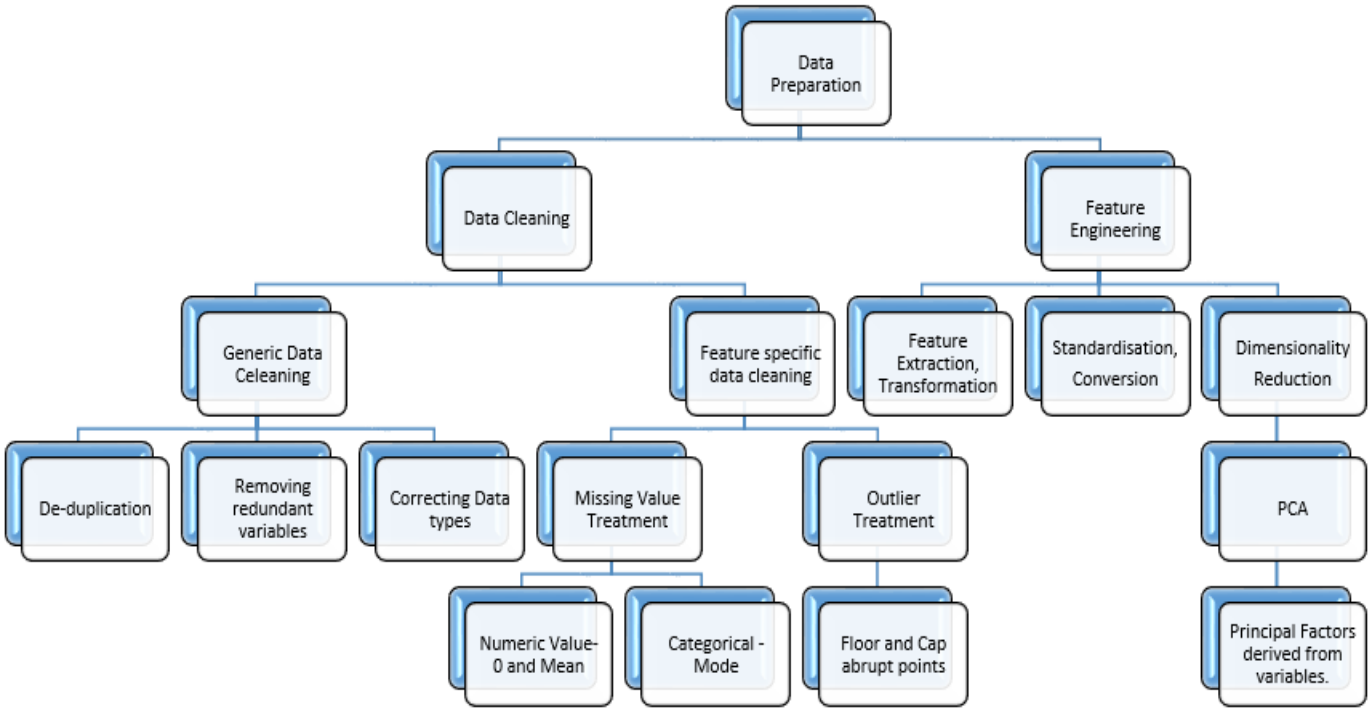

Figure 3.1