Rakamin Academy

HOME CREDIT

Virtual Internship Experience

Machine Learning

Data Cleaning,
Feature Engineering,
Normalization and Standardization.

What is Data Cleaning?

Data cleaning is the process of preparing data for analysis by removing or modifying data that is incorrect, incomplete, irrelevant, duplicated, or improperly formatted. But, as we mentioned above, it isn't as simple as organizing some rows or erasing information to make space for new data.

Data cleaning is a lot of muscle work. There's a reason data cleaning is the most important step if you want to create a data-culture, let alone make airtight predictions. It involves:

- Fixing spelling and syntax errors
- Standardizing data sets
- Correcting mistakes such as empty fields
- Identifying duplicate data points

It's said that the majority of a data scientist's time is spent on cleaning, rather than machine learning. In fact, 45% of data scientist's time is spent on preparing data.

And to us, that makes sense—if there's data that doesn't belong in your dataset, you aren't going to get accurate results. And with so much data these days, usually combined from multiple sources, and so many critical business decisions to make, you want to be extra sure that your data is clean.

Why is Data Cleaning so Important?

Businesses have a plethora of data. But not all of it is accurate or organized. When it comes to machine learning, if data is not cleaned thoroughly, the accuracy of your model stands on shaky grounds.

We've talked about how no-code simplifies the traditional machine learning process. What is typically a 10-step process instantly becomes a much simpler route with platforms like Obviously AI.

But the traditional and no-code process still requires the same important first step: connecting to your data. And all that time you've saved by opting for a no-code tool won't matter if those large stacks of data aren't properly organized, formatted, or accurate.

Preparing your data helps you maintain quality and makes for more accurate analytics, which increases effective, intelligent decision-making.

These are the kinds of benefits you'll see:

- Better decision making
- Boost in revenue
- Save time
- Increase productivity
- Streamline business practices

I've Already Cleaned My Data Myself. Can I Start Predicting?

The short answer is no. Unless you are trained in data science, or already know how to prepare your dataset, we typically advise against jumping right in.

The best way to explain this is with an analogy. Most people know how to drive a car. But not everyone knows how to drive a race car. Driving a regular car and driving a race car are two very different things. But

because they have the same fundamentals (press gas, brake, turn wheel), there are those who think that they'll be able to successfully drive a race car at the racetrack.

Race cars, however, put out raw power and it is largely up to the driver to filter that force as the car is driven. So, while you could absolutely go out on that racetrack and drive that race car, chances are, you won't be able to drive it well or get the most out of your experience. You might even crash.

The same thing can be said about data. Everyone knows how to operate an excel spreadsheet. But oftentimes, the dataset in that spreadsheet isn't set up for building machine learning models. Let's say you're trying to predict housing prices. You have a lot of data on sellers: their demographics, the amount they sold their house for, etc. You might also have data that appears to be irrelevant to what you want to predict. But that outlier may be crucial to your predictions. And machine learning will catch that.

This is why we always advise meeting with our team first before getting started on your first predictions and what we mean when we say that data cleaning is more than just formatting spreadsheets. And typically, we find that most people that go through onboarding have the most success when they see how data needs to be prepped.

How to Clean Your Data

Once you know what to look out for, it's easier to know what to look for when prepping your data. While the techniques used for data cleaning may vary depending on the type of data you're working with, the steps to prepare your data are fairly consistent.

Here are some steps you can take to properly prepare your data.

1. Remove duplicate observations

Duplicate data most often occurs during the data collection process. This typically happens when you combine data from multiple places, or receive data from clients or multiple departments. You want to remove any instances where duplicate data exists.

From a probabilistic perspective, you can think of duplicate data as adjusting the priors for a class label or data distribution. This may help an algorithm like Naive Bayes if you wish to purposefully bias the priors. Typically, this is not the case and machine learning algorithms will perform better by identifying and removing rows with duplicate data.
From an algorithm evaluation perspective, duplicate rows will result in misleading performance. For example, if you are using a train/test split or k-fold cross-validation, then it is possible for a duplicate row or rows to appear in both train and test datasets and any evaluation of the model on these rows will be (or should be) correct. This will result in an optimistically biased estimate of performance on unseen data.
If you think this is not the case for your dataset or chosen model, design a controlled experiment to test it. This could be achieved by evaluating model skill with the raw dataset and the dataset with duplicates removed and comparing performance. Another experiment might involve augmenting the dataset with different numbers of randomly selected duplicate examples.
You also want to remove any irrelevant observations from your dataset. This is where your data doesn't fit into the specific problem you're trying to analyze. This will help you make your analysis more efficient.

2. Filter unwanted outliers

Outliers are unusual values in your dataset. They're significantly different from other data points and can distort your analysis and violate assumptions. Removing them is a subjective practice and depends on what you're trying to analyze. Generally speaking, removing unwanted outliers will help improve the performance of the data you're working with.
Remove an outlier if:

- You know that it's wrong. For example, if you have a really good sense of what range the data should fall in, like people's ages, you can safely drop values that are outside of that range.
- You have a lot of data. Your sample won't be hurt by dropping a questionable outlier.

- You can go back and recollect. Or, you can verify the questionable data point.

Remember: just because an outlier exists, doesn't mean it is incorrect. Sometimes an outlier will help, for instance, prove a theory you're working on. If that's the case, keep the outlier.

3. Fix structural errors

Structural errors are things like strange naming conventions, typos, or incorrect capitalization. Anything that is inconsistent will create mislabeled categories.

A good example of this is when you have both "N/A" and "Not Applicable." Both are going to appear in separate categories, but they should both be analyzed as the same category.

4. Fix missing data

Make sure that any data that's missing is filled in. A lot of algorithms won't accept missing values. You may either drop the observations that have missing values, or you may input the missing value based on other observations.

5. Validate your data

Once you've thoroughly prepped your data, you should be able to answer these questions to validate it:

- Does your data make complete sense now?
- Does the data follow the relevant rules for its category or class?
- Does it prove/disprove your working theory?

6. Delete Columns That Contain a Single Value

Columns that have a single observation or value are probably useless for modeling. Here, a single value means that each row for that column has the same value.

Columns that have a single value for all rows do not contain any information for modeling. Depending on the choice of data preparation and modeling algorithms, variables with a single value can also cause errors or unexpected results.

You can detect rows that have this property using the unique() NumPy function that will report the number of unique values in each column.

7. Consider Columns That Have Very Few Values

In the previous section, we saw that some columns in the example dataset had very few unique values. For example, there were columns that only had 2, 4, and 9 unique values. This might make sense for ordinal or categorical variables. In this case, the dataset only contains numerical variables. As such, only having 2, 4, or 9 unique numerical values in a column might be surprising.

We can refer to these columns or predictors as near-zero variance predictors, as their variance is not zero, but a very small number close to zero. These columns may or may not contribute to the skill of a model. We can't assume that they are useless to modeling.

Depending on the choice of data preparation and modeling algorithms, variables with very few numerical values can also cause errors or unexpected results. For example, I have seen them cause errors when using power transforms for data preparation and when fitting linear models that assume a "sensible" data probability distribution.

To help highlight columns of this type, you can calculate the number of unique values for each variable as a percentage of the total number of rows in the dataset.

Another approach to the problem of removing columns with few unique values is to consider the variance of the column. Recall that the variance is a statistic calculated on a variable as the average squared difference of values on the sample from the mean. The variance can be used as a filter for identifying columns to be removed from the dataset. A column that has a single value has a variance of 0.0, and a column that has very few unique values will have a small variance value.

The VarianceThreshold class from the scikit-learn library supports this as a type of feature selection. An instance of the class can be created to specify the "threshold" argument, which defaults to 0.0 to remove columns with a single value.

It can then be fit and applied to a dataset by calling the fit_transform() function to create a transformed version of the dataset where the columns that have a variance lower than the threshold have been removed automatically.

```
# example of apply the variance threshold
from pandas import read_csv
from sklearn.feature_selection import VarianceThreshold
# define the location of the dataset
path = 'https://raw.githubusercontent.com/jbrownlee/Datasets/master/oil-spill.csv'
# load the dataset
df = read_csv(path, header=None)
# split data into inputs and outputs
data = df.values
X = data[:, :-1]
y = data[:, -1]
print(X.shape, y.shape)
# define the transform
transform = VarianceThreshold()
# transform the input data
X_sel = transform.fit_transform(X)
print(X_sel.shape)
```

Running the example first loads the dataset, then applies the transform to remove all columns with a variance of 0.0.

The shape of the dataset is reported before and after the transform, and we can see that the single column where all values are the same has been removed. We can define a sequence of thresholds from 0.0 to 0.5 with a step size of 0.05, e.g. 0.0, 0.05, 0.1, etc.

Summary

Data cleaning is an extremely vital step for any business that is data-centric. Businesses that take proper care of their datasets are rewarded with high-quality predictions and are able to make leaps ahead of their competition. With clean and organized data, you can predict anything. Obviously AI has a team of data scientists that become an extension of your team helping you make your datasets machine learning-ready.

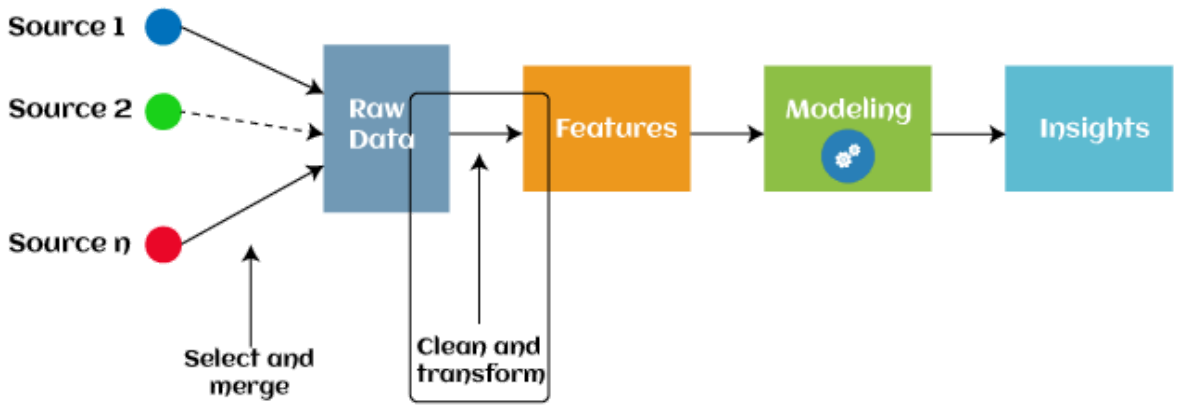
Feature Engineering

Introduction

Feature engineering is the pre-processing step of machine learning, which is used to transform raw data into features that can be used for creating a predictive model using Machine learning or statistical Modeling. The main goal of Feature engineering is to get the best result from the algorithm. Engineering in machine learning aims to improve the performance of models. In this topic, we will understand the details about feature engineering in Machine learning. But before going into details, let's first understand what features are? And What is the need for feature engineering?

What is Feature Engineering?

Feature engineering is the pre-processing step of machine learning, which extracts features from raw data. It helps to represent an underlying problem to predictive models in a better way, which as a result, improves the accuracy of the model for unseen data. The predictive model contains predictor variables and an outcome variable, and while the feature engineering process selects the most useful predictor variables for the model.



Why should we use Feature Engineering in data science?

In Data Science, the performance of the model is dependent on data preprocessing and data handling. Suppose if we build a model without Handling data, we get an accuracy of around 70%. By applying the Feature engineering on the same model there is a chance to increase the performance from 70% to more.

Simply, by using Feature Engineering we improve the performance of the model.

automated feature engineering is also used in different machine learning software that helps in automatically extracting features from raw data. Feature engineering in ML contains mainly four processes: Feature Creation, Transformations, Feature Extraction, and Feature Selection.

These processes are described as below:

1. Feature Creation: Feature creation is finding the most useful variables to be used in a predictive model. The process is subjective, and it requires human creativity and intervention. The new features are created by mixing existing features using addition, subtraction, and ration, and these new features have great flexibility.
2. Transformations: The transformation step of feature engineering involves adjusting the predictor variable to improve the accuracy and performance of the model. For example, it ensures that the model is flexible to take input of the variety of data; it ensures that all the variables are on the same scale, making the model easier to understand. It improves the model's accuracy and ensures that all the features are within the acceptable range to avoid any computational error.
3. Feature Extraction: Feature extraction is an automated feature engineering process that generates new variables by extracting them from the raw data. The main aim of this step is to reduce the volume of data so that it can be easily used and managed for data modeling. Feature extraction methods include cluster analysis, text analytics, edge detection algorithms, and principal components analysis (PCA).

4. Feature Selection: While developing the machine learning model, only a few variables in the dataset are useful for building the model, and the rest features are either redundant or irrelevant. If we input the dataset with all these redundant and irrelevant features, it may negatively impact and reduce the overall performance and accuracy of the model. Hence it is very important to identify and select the most appropriate features from the data and remove the irrelevant or less important features, which is done with the help of feature selection in machine learning. "Feature selection is a way of selecting the subset of the most relevant features from the original features set by removing the redundant, irrelevant, or noisy features."

Below are some benefits of using feature selection in machine learning:

- It helps in avoiding the curse of dimensionality.
- It helps in the simplification of the model so that the researchers can easily interpret it.
- It reduces the training time.
- It reduces overfitting hence enhancing the generalization.

Need for Feature Engineering in Machine Learning

In machine learning, the performance of the model depends on data pre-processing and data handling. But if we create a model without pre-processing or data handling, then it may not give good accuracy. Whereas, if we apply feature engineering on the same model, then the accuracy of the model is enhanced. Hence, feature engineering in machine learning improves the model's performance. Below are some points that explain the need for feature engineering:

- Better features mean flexibility.
  In machine learning, we always try to choose the optimal model to get good results. However, sometimes after choosing the wrong model, still, we can get better predictions, and this is because of better features. The flexibility in features will enable you to select the less complex models. Because less complex models are faster to run, easier to understand and maintain, which is always desirable.

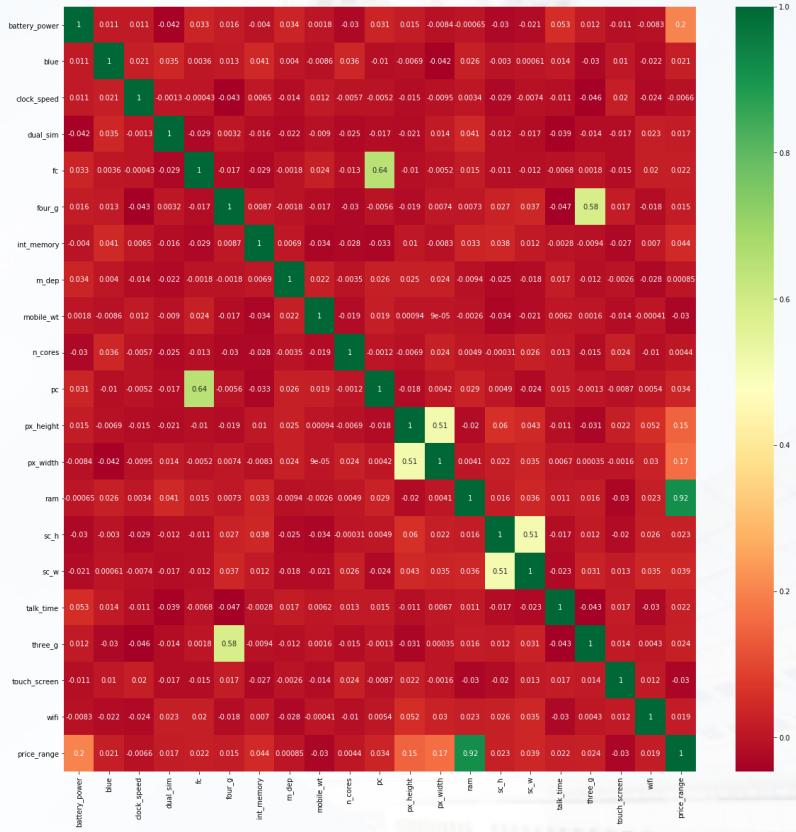
- Better features mean simpler models.
  If we input the well-engineered features to our model, then even after selecting the wrong parameters (Not much optimal), we can have good outcomes. After feature engineering, it is not necessary to work hard for picking the right model with the most optimized parameters. If we have good features, we can better represent the complete data and use it to best characterize the given problem.
- Better features mean better results.
  As already discussed, in machine learning, the data we will provide will get the same output. So, to obtain better results, we must use better features.

Feature selection

Feature selection is nothing but a selection of required independent features. Selecting the important independent features which have more relation with the dependent feature will help to build a good model. There are some methods for feature selection:

2.1 Correlation Matrix with Heatmap

Heatmap is a graphical representation of 2D (two-dimensional) data. Each data value is represented in a matrix.
Firstly, plot the pair plot between all independent features and dependent features. It will give the relation between dependent and independent features. The relation between the independent feature and the dependent feature is less than 0.2 then choose that independent feature for building a model.



2.2 Univariate Selection

In this, Statistical tests can be used to select the independent features which have the strongest relationship with the dependent feature. SelectKBest method can be used with a suite of different statistical tests to select a specific number of features.

```
In [43]: from sklearn.feature_selection import SelectKBest
         from sklearn.feature_selection import chi2
         bestfeatures = SelectKBest(score_func=chi2, k=10)
         fit = bestfeatures.fit(x,y)
         dfscores = pd.DataFrame(fit.scores_)
         dfcolumns = pd.DataFrame(x.columns)
         featureScores = pd.concat([dfcolumns,dfscores],axis=1)
         featureScores.columns = ['Specs','Score']
         featureScores
```

```
In [47]: featureScores
```

Out[47]:

	Specs	Score
0	City_Code	0.122643
1	Region_Code	74.805013
2	Accomodation_Type	0.756115
3	Reco_Insurance_Type	3.965972
4	Upper_Age	2.612166
5	Lower_Age	1.572930
6	Is_Spouse	0.632296
7	Health Indicator	0.453390
8	Holding_Policy_Duration	7.662646
9	Holding_Policy_Type	0.836189
10	Reco_Policy_Cat	1894.032997
11	Reco_Policy_Premium	9575.065324
12	diff_age	0.039347

Which feature has the highest score will be more related to the dependent feature and choose those features for the model.

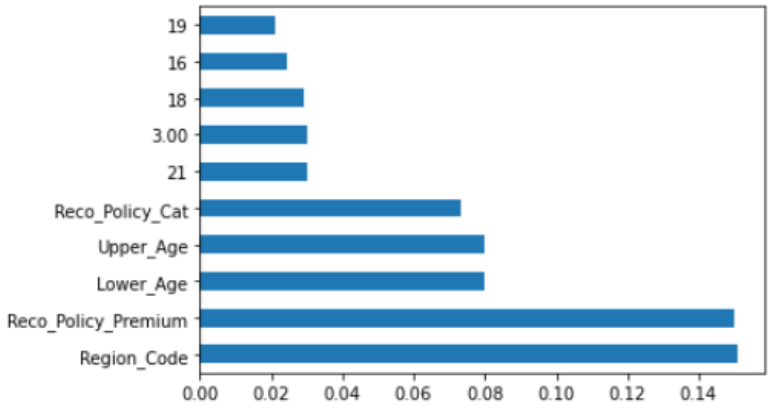
2.3 ExtraTreesClassifier method

In this method, the ExtraTreesClassifier method will help to give the importance of each independent feature with a dependent feature. Feature importance will give you a score for each feature of your data, the higher the score more important or relevant to the feature towards your output variable.

```
In [189]: from sklearn.ensemble import ExtraTreesClassifier
          import matplotlib.pyplot as plt
          model = ExtraTreesClassifier()
          model.fit(x,y)
          print(model.feature_importances_)
```

```
Out[189]: ExtraTreesClassifier()
```

```
In [192]: feat_importances = pd.Series(model.feature_importances_, index=x.columns)
          feat_importances.nlargest(10).plot(kind='barh')
          plt.show()
```



Handling Missing Values

In some datasets, we got the NA values in features. It is nothing but missing data. By handling this type of data there are many ways:

- In the missing value places, to replace the missing values with mean or median to numerical data and for categorical data with mode.

```
In [80]: train['Health'].fillna(train['Health'].mean(),inplace=True)
         train['Holding'].fillna(train['Holding'].median(),inplace=True)
         train['Holding_P'].fillna(train['Holding_P'].mode()[0],inplace=True)
```

- Drop NA values entire rows.

```
df.dropna(how = 'all')
```

- Drop NA values entire features. (it helps if NA values are more than 50% in a feature)

```
df.drop(['A'], axis=1, inplace=True)
```

- Replace NA values with 0.

```
df_result = df.fillna(0)
```

If you choose to drop options, there is a chance to lose important information from them. So better to choose to replace options.

Handling imbalanced data

Why need to handle imbalanced data? To reduce overfitting and underfitting problems. suppose a feature has a factor level 2 (0 and 1). It consists of 1's 5% and 0's 95%. It is called imbalanced data.

The shape of the training dataset:

0: 190430 transactions

1: 330 transactions

By preventing this problem there are some methods:

4.1 Under-sampling majority class

Under-sampling the majority class will resample the majority class points in the data to make them equal to the minority class.

```
In [ ]: from imblearn.under_sampling import RandomUnderSampler
        sam = RandomUnderSampler(random_state=0)
        X_resampled_under, y_resampled_under =sam.fit_resample(X_train, y_train)
```

Oversampling Minority class by duplication

Oversampling minority class will resample the minority class points in the data to make them equal to the majority class.

```
In [ ]: from imblearn.over_sampling import RandomOverSampler
        oversam = RandomOverSampler(sampling_strategy='minority')
        X_over, y_over = oversam.fit_resample(X, y)
```

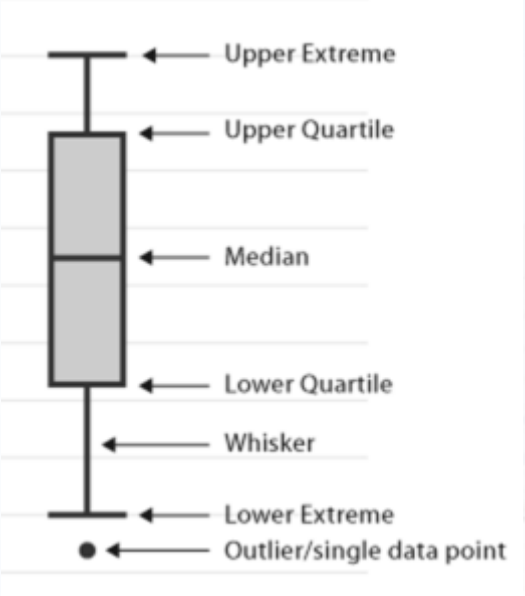
Oversampling minority class using Synthetic Minority Oversampling Technique (SMOTE)

In this method, synthetic samples are generated for the minority class and equal to the majority class.

```
In [ ]: from imblearn.over_sampling import SMOTE
        smot = SMOTE(sampling_strategy='minority')
        X_smot, y_smot = smot.fit_resample(X, y)
```

Handling outliers

Firstly, calculate the skewness of the features and check whether they are positively skewed, negatively skewed, or normally skewed. Another method is to plot the boxplot to features and check if any values are out of bounds or not. if there, they are called outliers.



how to handle these outliers:

first, calculate quantile values at 25% and 75%

```
Q1=mba['gmat'].quantile(0.25)     #to find quantile values
Q3=mba['gmat'].quantile(0.75)
```

- next, calculate the Interquartile range IQR = Q3 – Q1
- next, calculate the upper extreme and lower extreme values

  lower extreme=Q1 – 1.5 * IQR

  upper extreme=Q3– 1.5 * IQRe

```
lower_extreme=Q1-1.5*IQR
upper_extreme=Q3+1.5*IQR
```

- Lastly, check the values will lie above the upper extreme or below the lower extreme. if it presents then remove them or replace them with mean, median, or any quantile values.
- Replace outliers with mean
- Replace outliers with quantile values

```
In [ ]: out1=mba[(mba['gmat']<lower_extreme)].values
        out2=mba[(mba['gmat']>upper_extreme)].values
        mba["gmat"].replace(out1,lower_extreme,inplace=True)
        mba["gmat"].replace(out2,upper_extreme,inplace=True)
```

- Drop outliers

```
  ]: out=mba[(mba['gmat']<lower_extreme)|(mba['gmat']>upper_extreme)].index
     mba.drop(out,inplace=True)
```

Binning

Binning is nothing but any data value within the range is made to fit into the bin. It is important in your data exploration activity. We typically use it to transform continuous variables into discrete ones. Suppose if we have the AGE feature continuous and we need to divide the age into groups as a feature then it will be useful.

```
bin_data = data[['culmen_length_mm']]
bin_data['culmen_length_bin'] = pd.cut(data['culmen_length_mm'], bins=[0, 40, 50, 100], \
                                       labels=["Low", "Mid", "High"])
bin_data
```

	culmen_length_mm	culmen_length_bin
0	39.10000	Low
1	39.50000	Low
2	40.30000	Mid
3	43.92193	Mid
4	36.70000	Low
...	...	...
339	43.92193	Mid
340	46.80000	Mid
341	50.40000	High
342	45.20000	Mid
343	49.90000	Mid

Encoding:

Why will this apply? Because in datasets we may contain object data types. For building a model we need to have all features in integer data types. so, Label Encoder and OneHotEncoder are used to convert object data type to integer datatype.

- **Label Encoding**

```
In [ ]: from sklearn.preprocessing import LabelEncoder,OneHotEncoder
        label_1=LabelEncoder()
        dataset['Height']=label_1.fit_transform(dataset['Height'])
```

After applying label encoding, the Height column is converted into:

Height
Tall
Medium
Short

Height
0
1
2

After applying label encoding then apply the column transformer method to convert labels to 0 and 1

```
In [ ]: from sklearn.compose import ColumnTransformer
        ct=ColumnTransformer([("Height",OneHotEncoder(),[1])],remainder='passthrough')

In [ ]: dataset=ct.fit_transform(dataset)
```

- **One Hot Encoding:**

By applying get_dummies we convert directly categorical to numerical

```
dataset['Height']=pd.get_dummies(dataset['Height'], prefix_sep='_', dummy_na=False, columns=None, sparse=False,drop_first=False)
```

Conclusion

Although feature engineering helps in increasing the accuracy and performance of the model, there are also other methods that can increase prediction accuracy. Moreover, from the above-given techniques, there are many more available techniques of feature engineering, but we have mentioned the most commonly used techniques.

Dimensionality Reduction

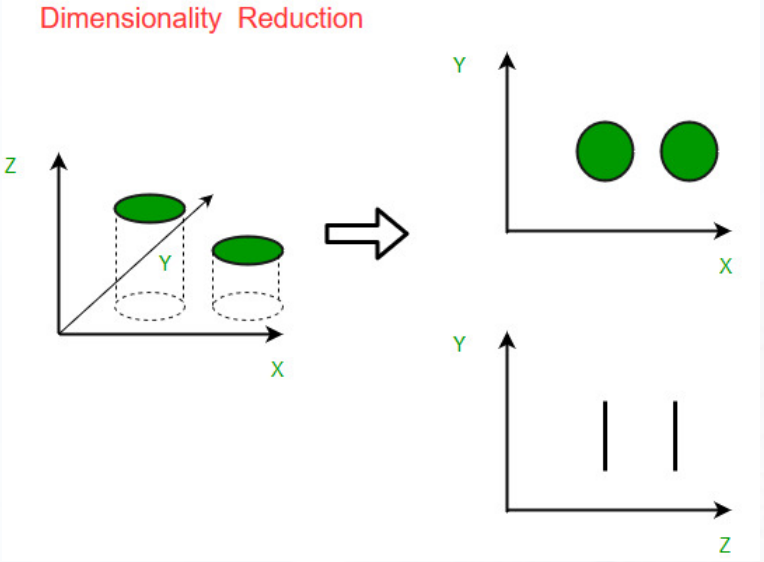
What is Dimensionality Reduction?

In machine learning classification problems, there are often too many factors on the basis of which the final classification is done. These factors are basically variables called features. The higher the number of features, the harder it gets to visualize the training set and then work on it. Sometimes, most of these features are correlated, and hence redundant. This is where dimensionality reduction algorithms come into play. Dimensionality reduction is the process of reducing the number of random variables under consideration, by obtaining a set of principal variables. It can be divided into feature selection and feature extraction.

Why is Dimensionality Reduction important in Machine Learning and Predictive Modeling?

An intuitive example of dimensionality reduction can be discussed through a simple email classification problem, where we need to classify whether the email is spam or not. This can involve a large number of features, such as whether or not the e-mail has a generic title, the content of the e-mail, whether the email uses a template, etc. However, some of these features may overlap. In another condition, a classification problem that relies on both humidity and rainfall can be collapsed into just one underlying feature, since both of the aforementioned are correlated to a high degree. Hence, we can reduce the number of features in such problems. A 3-D classification problem can be hard to visualize, whereas a 2-D one can be mapped to a simple 2 dimensional space, and a 1-D problem to a simple line. The below figure

illustrates this concept, where a 3-D feature space is split into two 1-D feature spaces, and later, if found to be correlated, the number of features can be reduced even further.



There are two components of dimensionality reduction:

- Feature selection: In this, we try to find a subset of the original set of variables, or features, to get a smaller subset which can be used to model the problem. It usually involves three ways:
    1. Filter
    2. Wrapper
    3. Embedded
- Feature extraction: This reduces the data in a high dimensional space to a lower dimension space, i.e. a space with lesser no. of dimensions.

Methods of Dimensionality Reduction

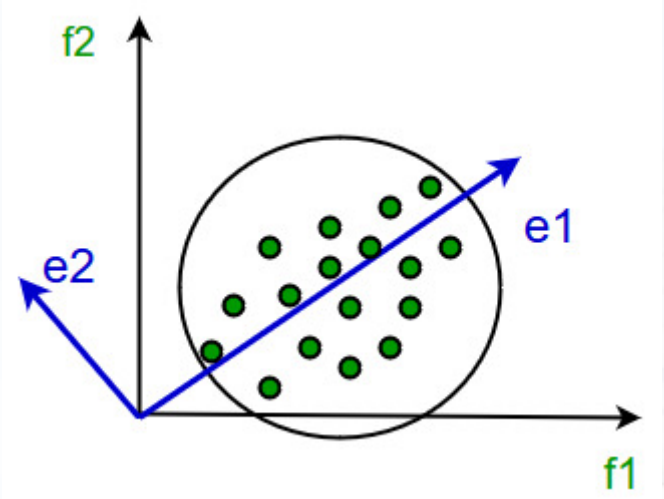
The various methods used for dimensionality reduction include:

- Principal Component Analysis (PCA)
- Linear Discriminant Analysis (LDA)
- Generalized Discriminant Analysis (GDA)

Dimensionality reduction may be both linear or nonlinear, depending upon the method used. The prime linear method, called Principal Component Analysis, or PCA, is discussed below.

Principal Component Analysis

This method was introduced by Karl Pearson. It works on a condition that while the data in a higher dimensional space is mapped to data in a lower dimension space, the variance of the data in the lower dimensional space should be maximum.



It involves the following steps:

- Construct the covariance matrix of the data.
- Compute the eigenvectors of this matrix.
- Eigenvectors corresponding to the largest eigenvalues are used to reconstruct a large fraction of variance of the original data.

Hence, we are left with a lesser number of eigenvectors, and there might have been some data loss in the process. But, the most important variances should be retained by the remaining eigenvectors.

Advantages of Dimensionality Reduction

- It helps in data compression, and hence reduced storage space.
- It reduces computation time.
- It also helps remove redundant features, if any.

Disadvantages of Dimensionality Reduction

- It may lead to some amount of data loss.
- PCA tends to find linear correlations between variables, which is sometimes undesirable.
- PCA fails in cases where mean and covariance are not enough to define datasets.

- We may not know how many principal components to keep- in practice, some thumb rules are applied.

Normalization and Standardization

The Scale of Your Data Matters

Machine learning models learn a mapping from input variables to an output variable. As such, the scale and distribution of the data drawn from the domain may be different for each variable. Input variables may have different units (e.g. feet, kilometers, and hours) that, in turn, may mean the variables have different scales.

Differences in the scales across input variables may increase the difficulty of the problem being modeled. An example of this is that large input values (e.g. a spread of hundreds or thousands of units) can result in a model that learns large weight values. A model with large weight values is often unstable, meaning that it may suffer from poor performance during learning and sensitivity to input values resulting in higher generalization error.

This difference in scale for input variables does not affect all machine learning algorithms. For example, algorithms that fit a model that use a weighted sum of input variables are affected, such as linear regression, logistic regression, and artificial neural networks (deep learning). Also, algorithms that use distance measures between examples or exemplars are affected, such as k-nearest neighbors and support vector machines. There are also algorithms that are unaffected by the scale of numerical input variables, most notably decision trees and ensembles of trees, like random forest. It can also be a good idea to scale the target variable for regression predictive modeling problems to make the problem easier to learn, most notably in the case of neural network models. A target variable with a large spread of values, in turn, may result in large error gradient values causing weight values to change dramatically, making the learning process unstable. So, Scaling input and output variables is a critical step in using neural network models.

What is Normalization?

Normalization is a scaling technique in which values are shifted and rescaled so that they end up ranging between 0 and 1. It is also known as Min-Max scaling. Here's the formula for normalization:

$$X' = \frac{X - X_{min}}{X_{max} - X_{min}}$$

Here, Xmax and Xmin are the maximum and the minimum values of the feature respectively.

- When the value of X is the minimum value in the column, the numerator will be 0, and hence X' is 0
- On the other hand, when the value of X is the maximum value in the column, the numerator is equal to the denominator and thus the value of X' is 1
- If the value of X is between the minimum and the maximum value, then the value of X' is between 0 and 1

Normalization using sklearn

To normalize your data, you need to import the MinMaxScalar from the sklearn library and apply it to our dataset.

```
# data normalization with sklearn
from sklearn.preprocessing import MinMaxScaler

# fit scaler on training data
norm = MinMaxScaler().fit(X_train)

# transform training data
X_train_norm = norm.transform(X_train)

# transform testing dataabs
X_test_norm = norm.transform(X_test)
```

Let's see how normalization has affected our dataset:

	Item_Weight	Item_Fat_Content	Item_Visibility	Item_MRP	Outlet_Establishment_Year	Outlet_Size	Tier 2	Tier 3	Supermarket Type1	Supermarket Type2
count	6818.000000	6818.000000	6818.000000	6818.000000	6818.000000	6818.000000	6818.000000	6818.000000	6818.000000	6818.000000
mean	0.493029	0.355676	0.595849	0.463485	0.464787	0.415151	0.326049	0.395571	0.651071	0.111616
std	0.252066	0.478753	0.175109	0.263444	0.349556	0.299176	0.468800	0.489009	0.476667	0.314917
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.282525	0.000000	0.479154	0.263566	0.208333	0.000000	0.000000	0.000000	0.000000	0.000000
50%	0.494352	0.000000	0.613084	0.470673	0.416667	0.500000	0.000000	0.000000	1.000000	0.000000
75%	0.681453	1.000000	0.730614	0.650280	0.916667	0.500000	1.000000	1.000000	1.000000	0.000000
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

All the features now have a minimum value of 0 and a maximum value of 1.

What is Standardization?

Standardization is another scaling technique where the values are centered around the mean with a unit standard deviation. This means that the mean of the attribute becomes zero and the resultant distribution has a unit standard deviation.
Here's the formula for standardization:

$$X' = \frac{X - \mu}{\sigma}$$

μ is the mean of the feature values and σ is the standard deviation of the feature values. Note that in this case, the values are not restricted to a particular range.

Standardization using sklearn

To standardize your data, you need to import the StandardScalar from the sklearn library and apply it to our dataset.

```
# data standardization with  sklearn
from sklearn.preprocessing import StandardScaler

# copy of datasets
X_train_stand = X_train.copy()
X_test_stand = X_test.copy()
```

```
# numerical features
num_cols                                                    =
['Item_Weight','Item_Visibility','Item_MRP','Outlet_Establishment_Year']

# apply standardization on numerical features
for i in num_cols:

    # fit on training data column
    scale = StandardScaler().fit(X_train_stand[[i]])

    # transform the training data column
    X_train_stand[i] = scale.transform(X_train_stand[[i]])

    # transform the testing data column
    X_test_stand[i] = scale.transform(X_test_stand[[i]])
```

You would have noticed that I only applied standardization to my numerical columns and not the other One-Hot Encoded features. Standardizing the One-Hot encoded features would mean assigning a distribution to categorical features. You don't want to do that!

But why did I not do the same while normalizing the data? Because One-Hot encoded features are already in the range between 0 to 1. So, normalization would not affect their value. Right, let's have a look at how standardization has transformed our data:

	Item_Weight	Item_Fat_Content	Item_Visibility	Item_MRP	Outlet_Establishment_Year	Outlet_Size	Tier 2	Tier 3	Supermarket Type1	Supermar Typ
count	6.818000e+03	6818.000000	6.818000e+03	6.818000e+03	6.818000e+03	6818.000000	6818.000000	6818.000000	6818.000000	6818.000(
mean	1.704754e-16	0.355676	2.342737e-16	1.233002e-16	2.051747e-17	0.830302	0.326049	0.395571	0.651071	0.111(
std	1.000073e+00	0.478753	1.000073e+00	1.000073e+00	1.000073e+00	0.598352	0.468800	0.489009	0.476667	0.314
min	-1.956094e+00	0.000000	-3.402972e+00	-1.759459e+00	-1.329746e+00	0.000000	0.000000	0.000000	0.000000	0.000(
25%	-8.351767e-01	0.000000	-6.664603e-01	-7.589239e-01	-7.337084e-01	0.000000	0.000000	0.000000	0.000000	0.000(
50%	5.250371e-03	0.000000	9.843446e-02	2.728679e-02	-1.376709e-01	1.000000	0.000000	0.000000	1.000000	0.000(
75%	7.475728e-01	1.000000	7.696596e-01	7.091011e-01	1.292819e+00	1.000000	1.000000	1.000000	1.000000	0.000(
max	2.011410e+00	1.000000	2.308161e+00	2.036689e+00	1.531234e+00	2.000000	1.000000	1.000000	1.000000	1.000(

The numerical features are now centered on the mean with a unit standard deviation. Awesome!

The Big Question – Normalize or Standardize?

Normalization vs. standardization is an eternal question among machine learning newcomers. Let me elaborate on the answer in this section.

- Normalization is good to use when you know that the distribution of your data does not follow a Gaussian distribution. This can be useful in algorithms that do not assume any distribution of the data like K-Nearest Neighbors and Neural Networks.
- Standardization, on the other hand, can be helpful in cases where the data follows a Gaussian distribution. However, this does not have to be necessarily true. Also, unlike normalization, standardization does not have a bounding range. So, even if you have outliers in your data, they will not be affected by standardization.

However, at the end of the day, the choice of using normalization or standardization will depend on your problem and the machine learning algorithm you are using. There is no hard and fast rule to tell you when to normalize or standardize your data. You can always start by fitting your model to raw, normalized and standardized data and compare the performance for best results.

It is a good practice to fit the scaler on the training data and then use it to transform the testing data. This would avoid any data leakage during the model testing process. Also, the scaling of target values is generally not required.

References :

- Jason Brownlee, How to perform Data Cleaning for Machine Learning with Python , from https://machinelearningmastery.com/basic-data-cleaning-for-machine-learning/ [acces:13/04/2022].
- Obviously.ai ,Data Cleaning : The most important step in Machine Learning, from https://www.obviously.ai/post/data-cleaning-in-machine-learning#:~:text=Data%20enrichment%2C%20data%20preparation,formatted%20data%20within%20a%20dataset. [Accesed: 13/04/2022].
- Jason Brownlee, How to Use StandardScaler and MinMaxScaler Transform in Python. From https://machinelearningmastery.com/standardscaler-and-minmaxscaler-transforms-in-python/ [Accesed : 13/04/2022].
- Aniruddha, Feature Scaling for Machine Learning : Understanding the Difference Between Normalization vs Standardization. From https://www.analyticsvidhya.com/blog/2020/04/feature-scaling-machine-learning-normalization-standardization/ [Accesed: 13/04/2022].
- Elluru Pavan, “Step by Step process of feature engineering for Machine Learning Algorithm in Data Science” from https://www.analyticsvidhya.com/blog/2021/03/step-by-step-process-of-feature-engineering-for-machine-learning-algorithms-in-data-science/ [accessed: 15/04/2022]
- JavaTpoint, “Feature Engineering for Machine Learning” from https://www.javatpoint.com/feature-engineering-for-machine-learning [Acessed: 15/04/2022].
- Geeksforgeeks, “Introduction to Dimensional Reduction”, from https://www.geeksforgeeks.org/dimensionality-reduction/ [Acessed : 15/04/2022].