Experiment 03 - Data Modelling

| Roll No. |  |
| --- | --- |
| Name |  |
| Class | D15-A |
| Subject | DS using Python Lab |
| LO Mapped | LO2: Analyse the data using different statistical techniques and visualise the outcome using different types of plots. |
|  |  |

**Aim**:

To perform Data Modelling for a selected dataset

**Introduction**:

In the context of Machine Learning, the split of our modelling dataset into training and testing samples is probably one of the earliest pre-processing steps that we need to undertake. The creation of different samples for training and testing helps us evaluate model performance.

In machine learning, there are several ways of data partitioning for experimentation. The most popular ways are typically referred to as training/test partitioning or cross-validation. The training/test partitioning typically involves the partitioning of the data into a training set and a test set in a specific ratio, e.g., 70% of the data are used as the training set and 30% of the data are used as the test set. This data partitioning can be done randomly or in a fixed way (e.g. the first 70% of the instances in the data set are assigned to the training set and the rest to the test set).

The fixed way is typically avoided (except when order matters) as it may introduce systematic differences between the training set and the test set, which leads to sample representativeness issues. To avoid such systematic differences, the random assignment of instances into training and test sets is typically used.

**Why do we need train and test samples**

A very common issue when training a model is overfitting. This phenomenon occurs when a model performs really well on the data that we used to train it but it fails to generalise well to new, unseen data points. There are numerous reasons why this can happen — it could be due to the noise in data or it could be that the model learned to predict specific inputs rather than the predictive parameters that could help it make correct predictions. Typically, the higher the complexity of a model the higher the chance that it will be overfitted.

On the other hand, underfitting occurs when the model has poor performance even on the data that was used to train it. In most cases, underfitting occurs because the model is not suitable for the problem you are trying to solve. Usually, this means that the model is less complex than required in order to learn those parameters that can be proven to be predictive.

Creating different data samples for training and testing the model is the most common approach that can be used to identify these sorts of issues. In this way, we can use the training set for training our model and then treat the testing set as a collection of data points that will help us evaluate whether the model can generalise well to new, unseen data.

The simplest way to split the modelling dataset into training and testing sets is to assign 2/3 data points to the former and the remaining one-third to the latter. Therefore, we train the model using the training set and then apply the model to the test set. In this way, we can evaluate the performance of our model. For instance, if the training accuracy is extremely high while the testing accuracy is poor then this is a good indicator that the model is probably overfitted.

Note that splitting the dataset into training and testing sets is not the only action that could be required in order to avoid phenomenons such as overfitting. For instance, if both the training and testing sets contain patterns that do not exist in real world data then the model would still have poor performance even though we wouldn’t be able to observe it from the performance evaluation.

On a second note, you should be aware that there are certain situations where you should consider creating an extra set called the validation set. The validation set is usually required when apart from model performance we also need to choose among many models and evaluate which model performs better.

**Partition the dataset into Training and Testing set**

We can create training and testing samples by the following ways:

1. Using scikit-learn (aka sklearn) train\_test\_split()
2. Using numpy ‘s randn() function
3. Using pandas' sample() function

**Using pandas**

The first option is to use pandas DataFrames’ method sample().

We initially create the training set by taking a sample with a fraction of 0.7 from the overall rows in the pandas DataFrame. Note that we also define random\_state which corresponds to the seed, so that results are reproducible. Subsequently, we create the testing set by simply dropping the corresponding indices from the original DataFrame which are now included in the training set.

training\_data = df.sample(frac=0.7, random\_state=25)

testing\_data = df.drop(training\_data.index)

**Using scikit-learn**

The second option — and probably the most commonly used — is the use of sklearn ‘s method called train\_test\_split().

We can create both the training and testing sets in a one-liner by passing to train\_test\_split() the modelling DataFrame along with the fraction of the examples that should be included in the testing set. As before, we also set a random\_state so that the results are reproducible, that is every time we run the code, the same instances will be included in the training and testing sets respectively. The method returns a tuple with two DataFrames containing the training and testing examples.

from sklearn.model\_selection import train\_test\_split

training\_data, testing\_data = train\_test\_split(df, test\_size=0.3, random\_state=25)

**Using numpy**

Finally, a less commonly used way of creating testing and training samples is with numpy ‘s method randn().

We first create a mask which is a numpy array that contains boolean values that were computed by comparing a random float number in the range between 0 and 1 with the fraction we want to keep for the training set. Subsequently, we create the training and testing samples by filtering the DataFrame accordingly.

import numpy as np

mask = np.random.rand(len(df)) =< 0.7

training\_data = df[mask]

testing\_data = df[~mask]

**For our Car Features and MSRP dataset:**

from sklearn.model\_selection import train\_test\_split

# Split df into X and y

y = df['MSRP']

X = df.drop('MSRP', axis=1)

# Train-test split (Serial Split)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, train\_size=0.7, shuffle=False)

# Train-test split (Random Split)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, train\_size=0.7, shuffle=True, random\_state=1)

**Analysis using Plots**

A frequency distribution shows how often each different value in a set of data occurs. A histogram is the most commonly used graph to show frequency distributions. It looks very much like a bar chart, but there are important differences between them.

Use a histogram when:

* The data are numerical
* You want to see the shape of the data’s distribution, especially when determining whether the output of a process is distributed approximately normally

Before drawing any conclusions from your histogram, be sure that the process was operating normally during the time period being studied. If any unusual events affected the process during the time period of the histogram, your analysis of the histogram shape likely cannot be generalised to all time periods.

**Typical Histogram Shapes And What They Mean:**

Normal Distribution

A common pattern is the bell-shaped curve known as the "normal distribution." In a normal or "typical" distribution, points are as likely to occur on one side of the average as on the other. Note that other distributions look similar to the normal distribution. Statistical calculations must be used to prove a normal distribution.

It's important to note that "normal" refers to the typical distribution for a particular process. For example, many processes have a natural limit on one side and will produce skewed distributions. This is normal—meaning typical—for those processes, even if the distribution isn’t considered "normal."



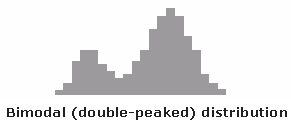
Skewed Distribution

The skewed distribution is asymmetrical because a natural limit prevents outcomes on one side. The distribution’s peak is off centre toward the limit and a tail stretches away from it. For example, a distribution of analyses of a very pure product would be skewed, because the product cannot be more than 100 percent pure. Other examples of natural limits are holes that cannot be smaller than the diameter of the drill bit or call-handling times that cannot be less than zero. These distributions are called right- or left-skewed according to the direction of the tail.



Double-Peaked or Bimodal

The bimodal distribution looks like the back of a two-humped camel. The outcomes of two processes with different distributions are combined in one set of data. For example, a distribution of production data from a two-shift operation might be bimodal, if each shift produces a different distribution of results. Stratification often reveals this problem.



Plateau or Multimodal Distribution

The plateau might be called a “multimodal distribution.” Several processes with normal distributions are combined. Because there are many peaks close together, the top of the distribution resembles a plateau.



Edge Peak Distribution

The edge peak distribution looks like the normal distribution except that it has a large peak at one tail. Usually this is caused by faulty construction of the histogram, with data lumped together into a group labelled “greater than.”



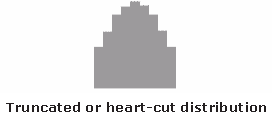
Comb Distribution

In a comb distribution, the bars are alternately tall and short. This distribution often results from rounded-off data and/or an incorrectly constructed histogram. For example, temperature data rounded off to the nearest 0.2 degree would show a comb shape if the bar width for the histogram were 0.1 degree.



Truncated or Heart-Cut Distribution

The truncated distribution looks like a normal distribution with the tails cut off. The supplier might be producing a normal distribution of material and then relying on inspection to separate what is within specification limits from what is out of spec. The resulting shipments to the customer from inside the specifications are the heart cut.



Dog Food Distribution

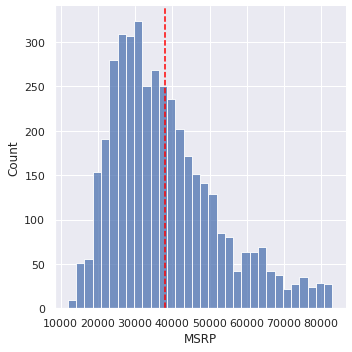
The dog food distribution is missing something—results near the average. If a customer receives this kind of distribution, someone else is receiving a heart cut and the customer is left with the “dog food,” the odds and ends left over after the master’s meal. Even though what the customer receives is within specifications, the product falls into two clusters: one near the upper specification limit and one near the lower specification limit. This variation often causes problems in the customer’s process.



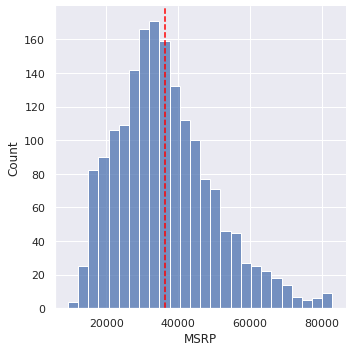
**For our Car Features and MSRP dataset:**

Serial Split:

Training dataset:



Testing dataset:

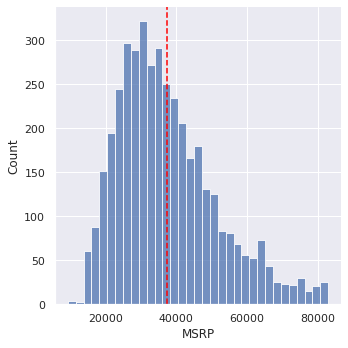


Inference:

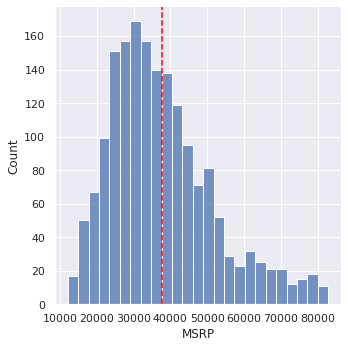
* The distribution curves of training and testing data sets show a lot of difference for every interval.
* The curves for both have high differences between the mean values, proving the dataset samples are not similar.

Random Split:

Training dataset:



Testing dataset:



Inference:

* The distribution curves of training and testing data sets are skewed to the right. (Right-Skewed Distribution Curves)
* The curves for both are very similar with the mean value being almost the same, proving the dataset samples are similar too.

**State Hypothesis**

Hypothesis testing is a part of statistical analysis, where we test the assumptions made regarding a population parameter.

It is generally used when we were to compare:

* a single group with an external standard
* two or more groups with each other

**Null Hypothesis**:

Null hypothesis is a statistical theory that suggests there is no statistical significance between the populations.

It is denoted by H0 and read as H-naught.

**Alternative Hypothesis**: An Alternative hypothesis suggests there is a significant difference between the population parameters. It could be greater or smaller. Basically, it is the contrast of the Null Hypothesis.

It is denoted by Ha or H1.

**Level of significance:**

Denoted by alpha or α. It is a fixed probability of wrongly rejecting a True Null Hypothesis. For example, if α=5%, that means we are okay to take a 5% risk and conclude there exists a difference when there is no actual difference.

**Critical Value:**

Denoted by C and it is a value in the distribution beyond which leads to the rejection of the Null Hypothesis. It is compared to the test statistic.

**Test Statistic:**

Denoted by t and is dependent on the test that we run. It is a deciding factor to reject or accept Null Hypothesis.

**For our Car Features and MSRP dataset:**

Sample 1: y\_train, having Mean = μ1

Sample 2: y\_test, having Mean = μ2

Null Hypothesis (H0) : μ1 == μ2

Alternative Hypothesis (H1) : μ1 != μ2

**Use t-test to confirm the hypothesis**

The t test tells you how significant the differences between groups are; In other words it lets you know if those differences (measured in means) could have happened by chance.

The t score is a ratio between the difference between two groups and the difference within the groups. The larger the t score, the more difference there is between groups. The smaller the t score, the more similarity there is between groups.

Every t-value has a p-value to go with it. A p-value is the probability that the results from your sample data occurred by chance. P-values are from 0% to 100%. They are usually written as a decimal. For example, a p value of 5% is 0.05. Low p-values are good; They indicate your data did not occur by chance.

**For our Car Features and MSRP dataset:**

For serial split:

from scipy import stats

t\_value,p\_value=stats.ttest\_ind(y\_train, y\_test, equal\_var=True)

print('Test statistic is %f'%float("{:.6f}".format(t\_value)))

print('p-value for two tailed test is %f'%p\_value)



alpha = 0.05

if p\_value<=alpha:

print('Reject H0. The mean of training and testing datasets are not equal.')

else:

print('We fail to reject H0. The mean of training and testing datasets are equal.')



For random split:

from scipy import stats

t\_value,p\_value=stats.ttest\_ind(y\_train, y\_test, equal\_var=True)

print('Test statistic is %f'%float("{:.6f}".format(t\_value)))

print('p-value for two tailed test is %f'%p\_value)



alpha = 0.05

if p\_value<=alpha:

print('Reject H0. The means of training and testing datasets are not equal.')

else:

print('We fail to reject H0. The means of training and testing datasets are equal.')



**Inference:**

The split dataset of train and test are similar distributions as their means are equal, as proven in the above hypothesis.

**Conclusion**:

Thus, we have learnt what splitting of dataset into training and testing samples mean, and learnt various ways to do this. We also learnt and performed a two-sample T-Test on the hypothesis that the train and test datasets are similar distributions.