Intellipaat

Feature Engineering

Agenda

01 Introduction to Feature Engineering

02 Different techniques for feature engineering

03 Handling Continuous features

04 Handling Categorical features

05 Feature Selection

06 Common methods for feature selection

Introduction to Feature Engineering

Introduction to Feature Engineering

It is a process of selecting and modifying features from the dataset while creating a predictive model using machine learning.

The two main objectives of Feature engineering are -

Preparing a suitable input dataset that meets the requirements of the machine learning algorithm.

Improving the performance of Machine Learning models

Note Cleaning and organizing the data takes up to 60% of a data scientist's time

Different Techniques for Feature Engineering

Different Techniques for Feature Engineering

Handling Missing Data

Many machine learning algorithms do not accept data with missing values, so handling missing data is critical. Missing values in the dataset might lead to error and poor performance of Machine Learning model.

Different types of common missing values in a dataset

NaN	**-**
null	**N/A**
none	**?**

Different techniques for Feature Engineering

Different ways to handle missing data -

Variable Deletion

Replace missing values with **mean/median/mode**

Variable Deletion

It can be used to remove columns with missing values. This method is suitable when there are lot of missing values present in a column and the column is not important feature

It is worth it to use when missing values are present more than 60 percent in a column.

Replacing missing values with mean/median/mode

There are three main missing value imputation techniques - mean, median and mode. Mean is the average of all values in a set, median is the middle number in a set of numbers sorted by size, and mode is the most common numerical value for two or more sets.

Impute missing values with Mean

- Replace the missing value with mean when there is a symmetric data distribution
- Imputing missing values with mean data can be done on numerical data
- Do not replace missing value with mean value when data is skewed. It may lead to lowering down of model accuracy.

Replacing missing values with mean/median/mode

Impute missing values with Median

- It is good to consider to replace the missing value with median when the data is skewed
- Imputing missing values with median value can be done only on numerical data

Impute missing values with Mode

- It is good to consider to replace the missing value with mode when the data is skewed
- Imputing missing values with mode value can be done on both categorical as well as numerical data

Handling Continuous Features

Handling Continuous Features

Continuous features present in the dataset consists of distinct range of values. Before you train machine learning algorithms, it's critical to deal with continuous features in your dataset.

Some examples of continuous features are salary, experience and prices

Model will not perform well if it is trained with different range of values

Normalization

Normalization

Normalization also known as min-max scaling is used to transform features into a similar scale. It scales the values from range between 0 and 1.

It is useful when features present in dataset are of different scales.

Scikit-learn library provides the MinMaxScaler method to normalize features.

It is beneficial when there are no outliers.

Code

```
# Normalization
from sklearn.preprocessing import MinMaxScaler
import numpy as np
a=np.array([[1000,200,30],[46,900,188],[759,800,999]])

m1=MinMaxScaler(feature_range=(0,1))
c=m1.fit_transform(a)
print(c)
```

```
[[1.         0.         0.        ]
 [0.         1.         0.1630547 ]
 [0.74737945 0.85714286 1.        ]]
```

Standardization

Standardization

It is a scaling technique in which the values are centered around the mean with a unit standard deviation. It makes sure that each feature present in the dataset has a mean of zero and standard deviation as 1

$$z = x\text{-}\mu \,/\, \sigma$$

μ = mean, σ = Standard Deviation, x= observation

Scikit-learn library provides the StandardScaler method to standardise features

Code

```
# standardization - mean value will be zero and standard deviation as 1
from sklearn.preprocessing import StandardScaler
import numpy as np
a=np.array([[1,2],
            [4,5],
            [7,9],
            [10,11]])
scaler=StandardScaler()
scaled_data=scaler.fit_transform(a)
print(scaled_data)

print(scaled_data.mean(axis=0))
print(scaled_data.mean(axis=1))
```

```
[[ 0.98342552 -1.40182605 -0.88500533]
 [-1.37185802  0.86266219 -0.51278481]
 [ 0.38843251  0.53916387  1.39779015]]
[0. 0.]
[1. 1.]
```

Handling Categorical Features

Handling Categorical Features

Categorical features represents the data that can be separated into categories such as country, gender etc.

To be used in most machine learning libraries, non-numerical values must be transformed to integers or floats. The following are some common methods to deal with categorical features -

01 **Label Encoding**

02 **One-hot-encoding**

Label Encoding

Label Encoding converts categorical values of a column into number. Here, each label is given a unique integer based on alphabetical order.

01

LabelEncoder from scikit learn library is used to convert categorical values to binary.

02

The only challenge with this is that it assigns a unique number to each class which lead to the priority issue.

One Hot Encoding

One hot encoding is another typical technique for dealing with categorical variables.

01 It adds new features based on the unique values present in the categorical column

02 It is a process to create dummy variables. Each category is represented as one-hot vector in this technique.

03 OneHotEncoder from scikit learn library is used to create one hot encoding of integer encoded values.

Code

```
from sklearn.preprocessing import LabelEncoder,OneHotEncoder
import numpy as np
import pandas as pd
a=np.array(["Summer","Winter","Rainy","Spring","Summer","Winter","Rainy","Spring"])
le=LabelEncoder()
label_encoding=le.fit_transform(a)
print(label_encoding)
label_encoding=label_encoding.reshape(len(label_encoding),1)
he= OneHotEncoder(sparse=False) # sparse= False will return an array
hot_encoding=he.fit_transform(label_encoding)
print(hot_encoding)
```

```
[2 3 0 1 2 3 0 1]
[[0. 0. 1. 0.]
 [0. 0. 0. 1.]
 [1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]
 [1. 0. 0. 0.]
 [0. 1. 0. 0.]]
```

Feature Selection

Feature Selection

It is a process of selecting the features either manually or automatically which contribute the most to the prediction variable. Irrelevant features present in the data can decrease the machine learning model accuracy

Following are the most compelling reasons to use feature selection -

01 It helps to reduce the complexity of a machine learning model

02 It helps to improve the accuracy of a model

03 It is used to reduce overfitting

04 It allows machine learning model to train faster

Common methods for Feature Selection

Removing features with low variance

sklearn.feature_selection module provides several classes which can be used for feature selection/dimensionality reduction on sample sets. Also it can be used to improve estimator's accuracy scores or enhance their performance on very high-dimensional datasets.

01 VarianceThreshold is a simple baseline approach to feature selection

02 All features whose variance does not meet a certain threshold are removed.

03 It removes all zero-variance features by default, i.e. features with the same value across all samples.

04 Threshold can be select using the formula **Var[X]=p(1-p)** for boolean featues.

Removing features with low variance

```
from sklearn.feature_selection import VarianceThreshold
X = [[0, 0, 1,0], [1, 1, 0, 0], [1, 0, 0, 0], [0, 1, 1, 0], [0, 1, 0, 0], [0, 1, 1, 0]]
sel = VarianceThreshold(threshold=(.7 * (1 - .7)))
sel.fit_transform(X)
```

```
array([[0, 0, 1],
       [1, 1, 0],
       [1, 0, 0],
       [0, 1, 1],
       [0, 1, 0],
       [0, 1, 1]])
```

Univariate feature selection

Statistical tests can assist in the selection of independent features from dataset that have the strongest relationship with the target feature.
Eg- Chi-squared test

01 The **SelectKBest** class in the Scikit-learn library can be used a variety of statistical tests to choose a certain number of features.

02 Here **K** is the number of top features to select

03 To import the SelectKBest class -
from sklearn.feature_selection import SelectKBest

Univariate feature selection

```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
from sklearn.datasets import load_iris

X, y = load_iris(return_X_y=True)
X.shape

new_X = SelectKBest(chi2, k=3).fit_transform(X, y)
new_X.shape
```

```
(150, 4)

(150, 3)
```

Recursive Feature Elimination

Recursive Feature Elimination (RFE) is a feature selection technique that reduces the complexity of a model by selecting important features by eliminating the less important ones.

01 Weaker features are removed one by one until the desired number of features to select is eventually reached.

02 Importance of each feature is obtained by attributes such as coef_, feature_importances_

03 To import RFE from sklearn module -
from sklearn.feature_selection import RFE

Recursive Feature Elimination

```
from sklearn.datasets import make_friedman1
from sklearn.feature_selection import RFE
from sklearn.svm import SVR
X, y = make_friedman1(n_samples=70, n_features=11, random_state=0)
estimator = SVR(kernel="linear")
selector = RFE(estimator, n_features_to_select=6, step=1)
selector = selector.fit(X, y)
selector.support_
selector.ranking_
```

```
array([ True,  True,  True,  True,  True, False, False, False,  True,
        False, False])
```

```
array([1, 1, 1, 1, 1, 3, 6, 2, 1, 5, 4])
```

Feature selection using SelectFromModel

Scikit-Learn library provides SelectFromModel class which is based on Machine Learning model estimation to extract best features

01 SelectFromModel feature selection is based on specific attribute(such as coef_ or feature_importances) threshold.

02 The mean is the threshold by default.

03 To import the SelectFromModel class -
from sklearn.feature_selection import SelectFromModel

Feature selection using SelectFromModel

```
from sklearn.feature_selection import SelectFromModel
from sklearn.linear_model import LogisticRegression
X = [[ 0.89, -1.30,  0.29 ],
      [-2.71, -0.01, -0.89 ],
      [-1.35, -0.47, -2.59 ],
      [ 1.93,  1.50,  0.71 ]]
y = [0, 1, 0, 1]

selector = SelectFromModel(estimator=LogisticRegression()).fit(X, y)
selector.estimator_.coef_
selector.threshold_
selector.get_support()
selector.transform(X)
```

```
array([[-0.33046635,  0.82726407,  0.50087696]])
```

```
0.5528691258331869
```

```
array([False,  True, False])
```

```
array([[-1.3 ],
       [-0.01],
       [-0.47],
       [ 1.5 ]])
```

Sequential Feature Selection

Sequential Feature Selection(SFS) is a greedy algorithm which is used to find the best features by moving either forward or backward based on the cross validation score of an estimator.

01

SFS Forward – It makes a feature selection by starting with zero feature and identifying the one feature that maximizes a cross-validated score when a machine model is trained on this single feature. After selecting the first feature , the process is repeated by adding new features until the desired number of features is reached.

02

SFS Backward – The similar concept is used by SFS-Backward, but it operates the opposite direction. It starts with all the features and removes all the features until the desired number of features are reached.

Correlation Matrix Heatmap

It shows the relationship between the features or target features.

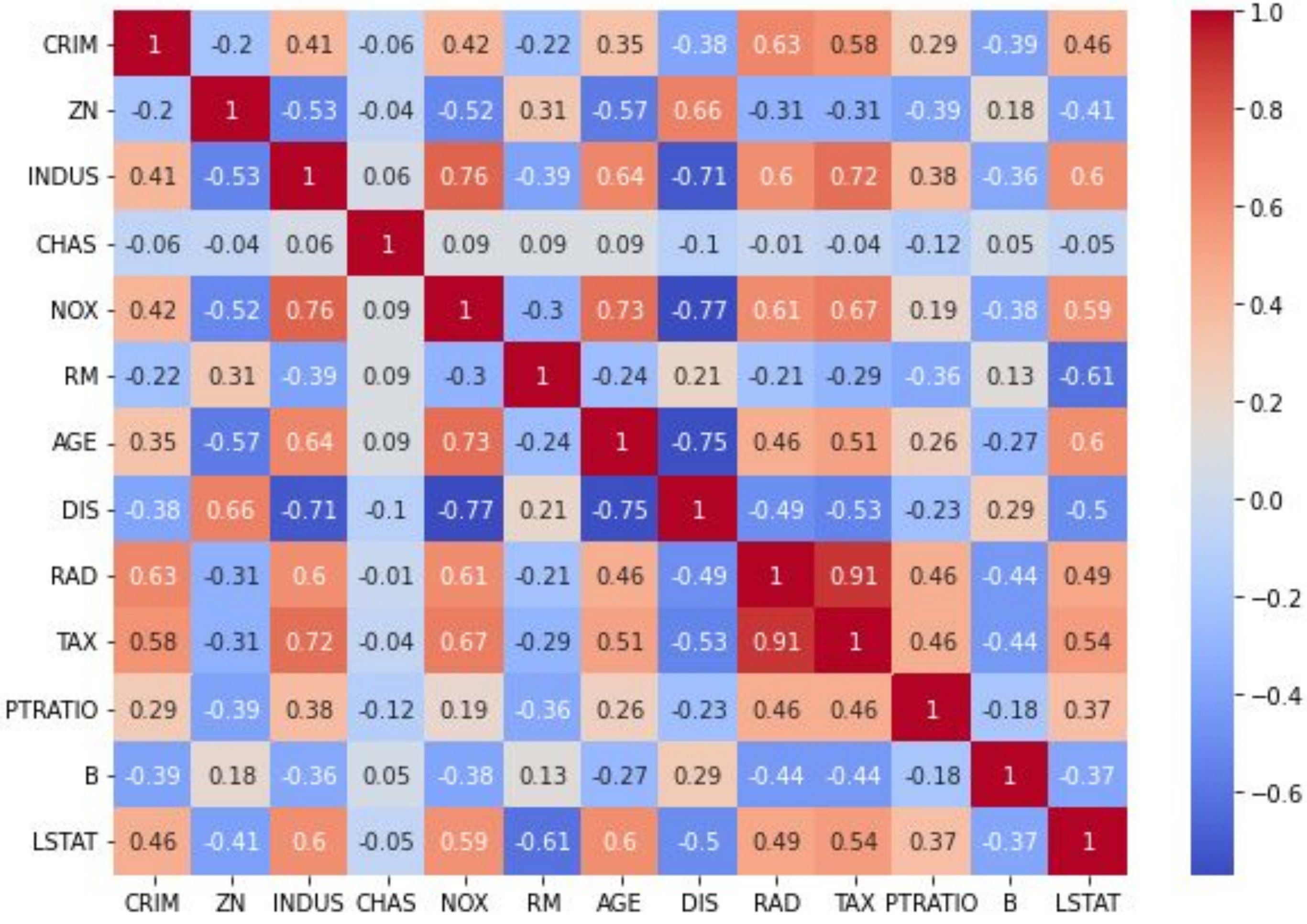
01 Correlation can be positive or negative

02 Pandas library provides the **corr( )** method to find the correlation of all features present in a dataframe

03 Correlation coefficient varies from -1 to1. If the value is 1 it shows there is a strong positive correlation between two features and -1 means features have strong negative correlation

Correlation Matrix Heatmap

```
boston.columns
plt.figure(figsize=(10,7))
ax=sns.heatmap(boston.corr().round(2), annot= True,cmap="coolwarm")
```



Thank You

Contact Us

India: +91-7847955955

US: 1-800-216-8930 (TOLL FREE)

sales@intellipaat.com

24/7 Chat with Our Course Advisor