**G. H. Raisoni Institute of Engineering &Management, Pune**

**DEPARTMENT: ARTIFICIAL INTELLIGENCE**

**ACADEMIC YEAR: 2020-21**

**CLASS: S.Y. BTech SEMESTER: IV**

**Subject Name: Data Pre-processing Laboratory**

**LAB MANUAL**

|  |  |
| --- | --- |
| **Sr. No.** | **Name of the Experiments** |
| 1 | Implementation of Basic Python Libraries |
| 2 | Find out missing data in dataset |
| 3 | Perform the Categorization of dataset |
| 4 | Execute feature scaling on given dataset |
| 5 | Implement normalization on dataset |
| 6 | Perform proper data labelling operation on dataset |
| 7 | Implement principal component analysis algorithm |
| 8 | Perform Encoding categorical features on given dataset |
| Open Ended Experiments / New Experiments | |
| 9 | Apply the appropriate Binarizationmethods on given dataset |
| 10 | Perform the Standardizationoperation on dataset |

**Assignment 1**

**Title:**Implementation of Basic Python Libraries.

**Aim**:

1. To understand and apply the Analytical concept of Python.2. To study Basic Python Libraries used for machine learning & data science.

**SOFTWARE REQUIREMENTS**:

1. Ubuntu 14.04 / 14.10

2. Python 3.9

3. Anaconda Spider/Jupiter Notebook

**THEORY:**

**Python libraries for Machine Learning**

Machine Learning, as the name suggests, is the science of programming a computer by which they are able to learn from different kinds of data. A more general definition given by Arthur Samuel is – “Machine Learning is the field of study that gives computers the ability to learn without being explicitly programmed.” They are typically used to solve various types of life problems.

In the older days, people used to perform Machine Learning tasks by manually coding all the algorithms and mathematical and statistical formula. This made the process time consuming, tedious and inefficient. But in the modern days, it is become very much easy and efficient compared to the olden days by various python libraries, frameworks, and modules. Today, Python is one of the most popular programming languages for this task and it has replaced many languages in the industry, one of the reasons is its vast collection of libraries. Python libraries that used in Machine Learning are:

* Numpy
* Pandas
* Matplotlib
* Scipy
* Scikit-learn
* Theano
* TensorFlow
* Keras
* PyTorch

**Numpy**



NumPy is a very popular python library for large multi-dimensional array and matrix processing, with the help of a large collection of high-level mathematical functions. It is very useful for fundamental scientific computations in Machine Learning. It is particularly useful for linear algebra, Fourier transform, and random number capabilities. High-end libraries like TensorFlow use NumPy internally for manipulation of Tensors.

# Python program using NumPy

# for some basic mathematical

# operations

Import numpy as np

# Creating two arrays of rank 2

x =np.array([[1, 2], [3, 4]])

y =np.array([[5, 6], [7, 8]])

# Creating two arrays of rank 1

v =np.array([9, 10])

w =np.array([11, 12])

# Inner product of vectors

print(np.dot(v, w), "\n")

# Matrix and Vector product

print(np.dot(x, v), "\n")

# Matrix and matrix product

print(np.dot(x, y))

**Output:** 

219

[29 67]

[[19 22]

[43 50]]

#### Pandas

#### 

#### Pandas is a popular Python library for data analysis. It is not directly related to Machine Learning. As we know that the dataset must be prepared before training. In this case, Pandas comes handy as it was developed specifically for data extraction and preparation. It provides high-level data structures and wide variety tools for data analysis. It provides many inbuilt methods for groping, combining and filtering data.

# Python program using Pandas for

# arranging a given set of data

# Into a table

# importing pandas as pd

Import pandas as pd

data ={"country": ["Brazil", "Russia", "India", "China", "South Africa"],

       "capital": ["Brasilia", "Moscow", "New Dehli", "Beijing", "Pretoria"],

       "area": [8.516, 17.10, 3.286, 9.597, 1.221],

       "population": [200.4, 143.5, 1252, 1357, 52.98] }

data\_table =pd.DataFrame(data)

print(data\_table)

#### Output:

#### 

#### Matplotlib

#### 

#### Matplotlib is a very popular Python library for data visualization. Like Pandas, it is not directly related to Machine Learning. It particularly comes in handy when a programmer wants to visualize the patterns in the data. It is a 2D plotting library used for creating 2D graphs and plots. A module named pyplot makes it easy for programmers for plotting as it provides features to control line styles, font properties, formatting axes, etc. It provides various kinds of graphs and plots for data visualization, viz., histogram, error charts, bar chats, etc,

#  Python program using Matplotlib

# for forming a linear plot

# importing the necessary packages and modules

Import matplotlib.pyplot as plt

Import numpy as np

# Prepare the data

x =np.linspace(0, 10, 100)

# Plot the data

plt.plot(x, x, label ='linear')

# Add a legend

plt.legend()

# Show the plot

plt.show()

#### **Output:**

#### 

#### Scikit-learn

#### Skikit-learn is one of the most popular ML libraries for classical ML algorithms. It is built on top of two basic Python libraries, viz., NumPy and SciPy. Scikit-learn supports most of the supervised and unsupervised learning algorithms. Scikit-learn can also be used for data-mining and data-analysis, which makes it a great tool who is starting out with ML.

#### Theano

#### We all know that Machine Learning is basically mathematics and statistics. Theano is a popular python library that is used to define, evaluate and optimize mathematical expressions involving multi-dimensional arrays in an efficient manner. It is achieved by optimizing the utilization of CPU and GPU. It is extensively used for unit-testing and self-verification to detect and diagnose different types of errors. Theano is a very powerful library that has been used in large-scale computationally intensive scientific projects for a long time but is simple and approachable enough to be used by individuals for their own projects.

#### TensorFlow

#### TensorFlow is a very popular open-source library for high performance numerical computation developed by the Google Brain team in Google. As the name suggests, Tensorflow is a framework that involves defining and running computations involving tensors. It can train and run deep neural networks that can be used to develop several AI applications. TensorFlow is widely used in the field of deep learning research and application.

#### Keras

#### Keras is a very popular Machine Learning library for Python. It is a high-level neural networks API capable of running on top of TensorFlow, CNTK, or Theano. It can run seamlessly on both CPU and GPU. Keras makes it really for ML beginners to build and design a Neural Network. One of the best thing about Keras is that it allows for easy and fast prototyping.

#### SciPy

#### 

SciPy is a very popular library among Machine Learning enthusiasts as it contains different modules for optimization, linear algebra, integration and statistics. There is a difference between the SciPy library and the SciPy stack. The SciPy is one of the core packages that make up the SciPy stack. SciPy is also very useful for image manipulation.

**Conclusion:**

This Practical we learned different types of Python ML Libraries.

**Assignment 2**

**Title:**Find out missing data in Dataset.

**Aim**:

1. To understand and apply the Data Pre-processing concept.2. To study detailed Data Pre-processingconcept in Python.

**SOFTWARE REQUIREMENTS**:

1. Ubuntu 16+

2. Python 3.9+

3. Anaconda Spider/Jupiter Notebook

**THEORY:**

Missing Data can occur when no information is provided for one or more items or for a whole unit. Missing Data is a very big problem in real-life scenarios. Missing Data can also refer to as NA(Not Available) values in pandas. In Data Frame sometimes many datasets simply arrive with missing data, either because it exists and was not collected or it never existed. For Example, suppose different users being surveyed may choose not to share their income, some users may choose not to share the address in this way many datasets went missing.

In Pandas missing data is represented by two value:

* None: None is a Python singleton object that is often used for missing data in Python code.
* NaN : NaN (an acronym for Not a Number), is a special floating-point value recognized by all systems that use the standard IEEE floating-point representation

Pandas treat None and NaN as essentially interchangeable for indicating missing or null values. To facilitate this convention, there are several useful functions for detecting, removing, and replacing null values in Pandas DataFrame :

* isnull()
* notnull()
* dropna()
* fillna()
* replace()
* interpolate()

Checking for missing values using isnull() and notnull()

In order to check missing values in Pandas DataFrame, we use a function isnull() and notnull(). Both function help in checking whether a value is NaN or not.These function can also be used in Pandas Series in order to find null values in a series.

Checking for missing values using isnull()

In order to check null values in Pandas DataFrame, we use isnull() function this function return dataframe of Boolean values which are True for NaN values.

# importing pandas as pd

Import pandas as pd

# importing numpy as np

Import numpy as np

# dictionary of lists

dict={'First Score':[100, 90, np.nan, 95],

        'Second Score': [30, 45, 56, np.nan],

        'Third Score':[np.nan, 40, 80, 98]}

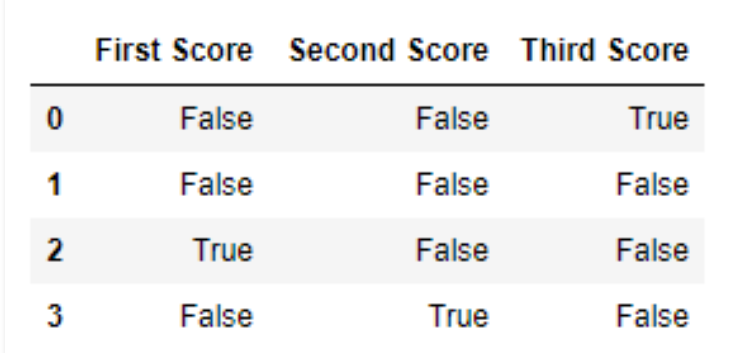
# creating a dataframe from list

df =pd.DataFrame(dict)

# using isnull() function

df.isnull()

Output



# importing pandas as pd

importpandas as pd

# importing numpy as np

importnumpy as np

# dictionary of lists

dict={'First Score':[100, 90, np.nan, 95],

        'Second Score': [30, 45, 56, np.nan],

        'Third Score':[np.nan, 40, 80, 98]}

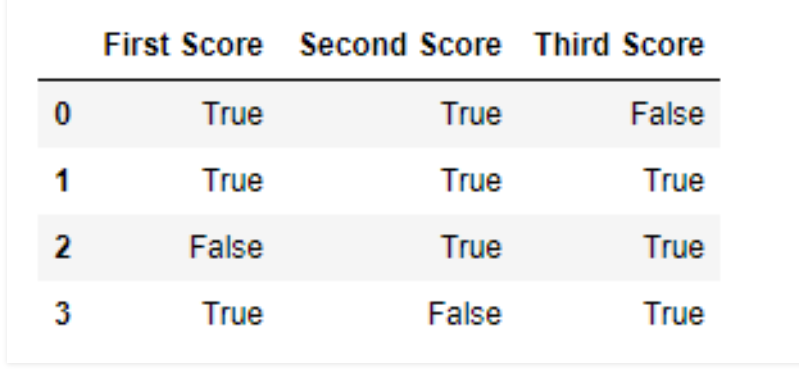
# creating a dataframe using dictionary

df =pd.DataFrame(dict)

# using notnull() function

df.notnull()

Output



**Conclusion:**

Thus we have studied different methods to replace missing data in dataset.

**Assignment 3**

**Title:**Perform the Categorization of dataset

**Theory:**

Classification is a large domain in the field of statistics and machine learning. Generally, classification can be broken down into two areas:

1. **Binary classification**, where we wish to group an outcome into one of two groups.
2. **Multi-class classification**, where we wish to group an outcome into one of multiple (more than two) groups.

In this post, the main focus will be on using a variety of classification algorithms across both of these domains, less emphasis will be placed on the theory behind them.

We can use libraries in Python such as [scikit-learn](http://scikit-learn.org/stable/) for machine learning models, and [Pandas](https://pandas.pydata.org/) to import data as data frames.

These can easily be installed and imported into Python with pip:

$ python3 -m pip install sklearn

$ python3 -m pip install pandas

importsklearn as sk

import pandas as pd

**Binary Classification**

For binary classification, we are interested in classifying data into one of two *binary* groups - these are usually represented as 0's and 1's in our data.

We will look at data regarding coronary heart disease (CHD) in South Africa. The goal is to use different variables such as *tobacco usage*, *family history*, *ldl cholesterol levels*, *alcohol usage*, *obesity* and more.

A full description of this dataset is available in the "Data" section of the [Elements of Statistical Learning](https://web.stanford.edu/~hastie/ElemStatLearn/) website.

The code below reads the data into a Pandas data frame, and then separates the data frame into a y vector of the response and an X matrix of explanatory variables:

import pandas as pd

importos

os.chdir('/Users/stevenhurwitt/Documents/Blog/Classification')

heart = pd.read\_csv('SAHeart.csv', sep=',', header=0)

heart.head()

y = heart.iloc[:,9]

X = heart.iloc[:,:9]

When running this code, just be sure to change the file system path on line 4 to suit your setup.

sbp tobacco ldl adiposity famhist typea obesity alcohol age chd

0 160 12.00 5.73 23.11 1 49 25.30 97.20 52 1

1 144 0.01 4.41 28.61 0 55 28.87 2.06 63 1

2 118 0.08 3.48 32.28 1 52 29.14 3.81 46 0

3 170 7.50 6.41 38.03 1 51 31.99 24.26 58 1

4 134 13.60 3.50 27.78 1 60 25.99 57.34 49 1

**Logistic Regression**

[Logistic Regression](https://en.wikipedia.org/wiki/Logistic_regression) is a type of [Generalized Linear Model (GLM)](https://en.wikipedia.org/wiki/Generalized_linear_model) that uses a logistic function to model a binary variable based on any kind of independent variables.

To fit a binary logistic regression with sklearn, we use the [LogisticRegression](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.htmlLogisticRegression) module with multi\_class set to "ovr" and fit X and y.

We can then use the predict method to predict probabilities of new data, as well as the score method to get the mean prediction accuracy:

importsklearn as sk

fromsklearn.linear\_model import LogisticRegression

import pandas as pd

importos

os.chdir('/Users/stevenhurwitt/Documents/Blog/Classification')

heart = pd.read\_csv('SAHeart.csv', sep=',',header=0)

heart.head()

y = heart.iloc[:,9]

X = heart.iloc[:,:9]

LR = LogisticRegression(random\_state=0, solver='lbfgs', multi\_class='ovr').fit(X, y)

LR.predict(X.iloc[460:,:])

round(LR.score(X,y), 4)

array([1, 1])

**Support Vector Machines**

[Support Vector Machines (SVMs)](https://stackabuse.com/implementing-svm-and-kernel-svm-with-pythons-scikit-learn/) are a type of classification algorithm that are more flexible - they can do linear classification, but can use other non-linear *basis functions*. The following example uses a linear classifier to fit a hyperplane that separates the data into two classes:

importsklearn as sk

fromsklearn import svm

import pandas as pd

importos

os.chdir('/Users/stevenhurwitt/Documents/Blog/Classification')

heart = pd.read\_csv('SAHeart.csv', sep=',',header=0)

y = heart.iloc[:,9]

X = heart.iloc[:,:9]

SVM = svm.LinearSVC()

SVM.fit(X, y)

SVM.predict(X.iloc[460:,:])

round(SVM.score(X,y), 4)

array([0, 1])

**Random Forests**

[Random Forests](https://stackabuse.com/random-forest-algorithm-with-python-and-scikit-learn/) are an ensemble learning method that fit multiple [Decision Trees](https://en.wikipedia.org/wiki/Decision_tree) on subsets of the data and average the results. We can again fit them using sklearn, and use them to predict outcomes, as well as get mean prediction accuracy:

importsklearn as sk

fromsklearn.ensemble import RandomForestClassifier

RF = RandomForestClassifier(n\_estimators=100, max\_depth=2, random\_state=0)

RF.fit(X, y)

RF.predict(X.iloc[460:,:])

round(RF.score(X,y), 4)

0.7338

**Neural Networks**

[Neural Networks](https://stackabuse.com/introduction-to-neural-networks-with-scikit-learn/) are a machine learning algorithm that involves fitting many *hidden layers* used to represent neurons that are connected with synaptic *activation functions*. These essentially use a very simplified model of the brain to model and predict data.

We use sklearn for consistency in this post, however libraries such as [Tensorflow](https://www.tensorflow.org/) and [Keras](https://keras.io/) are more suited to fitting and customizing neural networks, of which there are a few varieties used for different purposes:

importsklearn as sk

fromsklearn.neural\_network import MLPClassifier

NN = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden\_layer\_sizes=(5, 2), random\_state=1)

NN.fit(X, y)

NN.predict(X.iloc[460:,:])

round(NN.score(X,y), 4)

0.6537

**Multi-Class Classification**

While binary classification alone is incredibly useful, there are times when we would like to model and predict data that has more than two classes. Many of the same algorithms can be used with slight modifications.

Additionally, it is common to split data into *training* and *test* sets. This means we use a certain portion of the data to fit the model (the training set) and save the remaining portion of it to evaluate to the predictive accuracy of the fitted model (the test set).

There's no official rule to follow when deciding on a split proportion, though in most cases you'd want about 70% to be dedicated for the training set and around 30% for the test set.

To explore both multi-class classifications, as well as training/test data, we will look at another [dataset from the Elements of Statistical Learning website](https://web.stanford.edu/~hastie/ElemStatLearn/data.html). This is data used to determine which one of eleven vowel sounds were spoken:

import pandas as pd

vowel\_train = pd.read\_csv('vowel.train.csv', sep=',', header=0)

vowel\_test = pd.read\_csv('vowel.test.csv', sep=',', header=0)

vowel\_train.head()

y\_tr = vowel\_train.iloc[:,0]

X\_tr = vowel\_train.iloc[:,1:]

y\_test = vowel\_test.iloc[:,0]

X\_test = vowel\_test.iloc[:,1:]

y x.1 x.2 x.3 x.4 x.5 x.6 x.7 x.8 x.9 x.10

0 1 -3.639 0.418 -0.670 1.779 -0.168 1.627 -0.388 0.529 -0.874 -0.814

1 2 -3.327 0.496 -0.694 1.365 -0.265 1.933 -0.363 0.510 -0.621 -0.488

2 3 -2.120 0.894 -1.576 0.147 -0.707 1.559 -0.579 0.676 -0.809 -0.049

3 4 -2.287 1.809 -1.498 1.012 -1.053 1.060 -0.567 0.235 -0.091 -0.795

4 5 -2.598 1.938 -0.846 1.062 -1.633 0.764 0.394 -0.150 0.277 -0.396

We will now fit models and test them as is normally done in statistics/machine learning: by training them on the training set and evaluating them on the test set.

Additionally, since this is multi-class classification, some arguments will have to be changed within each algorithm:

import pandas as pd

importsklearn as sk

fromsklearn.linear\_model import LogisticRegression

fromsklearn import svm

fromsklearn.ensemble import RandomForestClassifier

fromsklearn.neural\_network import MLPClassifier

vowel\_train = pd.read\_csv('vowel.train.csv', sep=',',header=0)

vowel\_test = pd.read\_csv('vowel.test.csv', sep=',',header=0)

y\_tr = vowel\_train.iloc[:,0]

X\_tr = vowel\_train.iloc[:,1:]

y\_test = vowel\_test.iloc[:,0]

X\_test = vowel\_test.iloc[:,1:]

LR = LogisticRegression(random\_state=0, solver='lbfgs', multi\_class='multinomial').fit(X\_tr, y\_tr)

LR.predict(X\_test)

round(LR.score(X\_test,y\_test), 4)

SVM = svm.SVC(decision\_function\_shape="ovo").fit(X\_tr, y\_tr)

SVM.predict(X\_test)

round(SVM.score(X\_test, y\_test), 4)

RF = RandomForestClassifier(n\_estimators=1000, max\_depth=10, random\_state=0).fit(X\_tr, y\_tr)

RF.predict(X\_test)

round(RF.score(X\_test, y\_test), 4)

NN = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden\_layer\_sizes=(150, 10), random\_state=1).fit(X\_tr, y\_tr)

NN.predict(X\_test)

round(NN.score(X\_test, y\_test), 4)

Output

0.5455

Although the implementations of these models were rather naive (in practice there are a variety of parameters that can and should be varied for each model), we can still compare the predictive accuracy across the models. This will tell us which one is the most accurate for this specific training and test dataset:

|  |  |
| --- | --- |
| **Model** | **Predictive Accuracy** |
| Logistic Regression | 46.1% |
| Support Vector Machine | 64.07% |
| Random Forest | 57.58% |
| Neural Network | 54.55% |

This shows us that for the vowel data, an SVM using the default radial basis function was the most accurate.

**Conclusion**

To summarize this post, we began by exploring the simplest form of classification: binary. This helped us to model data where our response could take one of two states.

We then moved further into multi-class classification, when the response variable can take any number of states.

We also saw how to fit and evaluate models with training and test sets. Furthermore, we could explore additional ways to refine model fitting among various algorithms.

**Assignment 4**

**Title:**Execute feature scaling on given dataset.

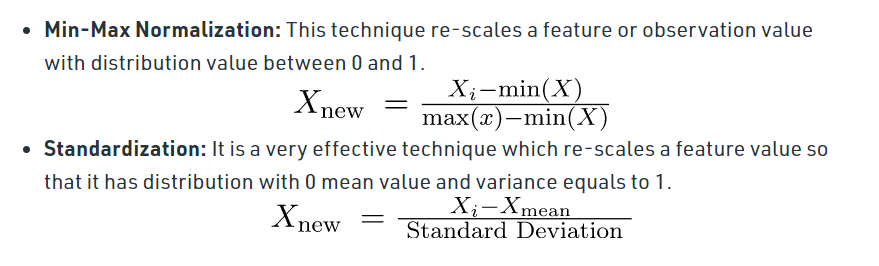
**Theory:**

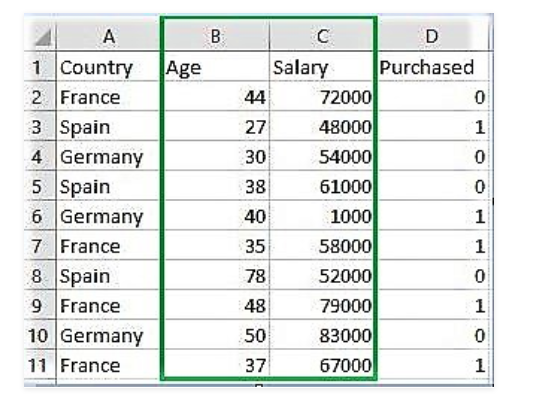
Feature Scaling is a technique to standardize the independent features present in the data in a fixed range. It is performed during the data pre-processing to handle highly varying magnitudes or values or units. If feature scaling is not done, then a machine learning algorithm tends to weigh greater values, higher and consider smaller values as the lower values, regardless of the unit of the values.

Example: If an algorithm is not using the feature scaling method then it can consider the value 3000 meters to be greater than 5 km but that’s actually not true and in this case, the algorithm will give wrong predictions. So, we use Feature Scaling to bring all values to the same magnitudes and thus, tackle this issue.

**Techniques to perform Feature Scaling**

Consider the two most important ones:

D



**Code: Python code explaining the working of Feature Scaling on the data**

# Python code explaining How to

# perform Feature Scaling

""" PART 1

    Importing Libraries """

importnumpy as np

importmatplotlib.pyplot as plt

importpandas as pd

# Sklearn library

fromsklearn importpreprocessing

""" PART 2

    Importing Data """

data\_set =pd.read\_csv('C:\\Users\\dell\\Desktop\\Data\_for\_Feature\_Scaling.csv')

data\_set.head()

# here Features - Age and Salary columns

# are taken using slicing

# to handle values with varying magnitude

x =data\_set.iloc[:, 1:3].values

print("\nOriginal data values : \n",  x)

""" PART 4

    Handling the missing values """

fromsklearn importpreprocessing

""" MIN MAX SCALER """

min\_max\_scaler =preprocessing.MinMaxScaler(feature\_range =(0, 1))

# Scaled feature

x\_after\_min\_max\_scaler =min\_max\_scaler.fit\_transform(x)

print("\nAfter min max Scaling : \n", x\_after\_min\_max\_scaler)

""" Standardisation """

Standardisation =preprocessing.StandardScaler()

# Scaled feature

x\_after\_Standardisation =Standardisation.fit\_transform(x)

print("\nAfter Standardisation : \n", x\_after\_Standardisation)

**Output :**

Country Age Salary Purchased

0 France 44 72000 0

1 Spain 27 48000 1

2 Germany 30 54000 0

3 Spain 38 61000 0

4 Germany 40 1000 1

Original data values :

[[ 44 72000]

[ 27 48000]

[ 30 54000]

[ 38 61000]

[ 40 1000]

[ 35 58000]

[ 78 52000]

[ 48 79000]

[ 50 83000]

[ 37 67000]]

After min max Scaling :

[[ 0.33333333 0.86585366]

[ 0. 0.57317073]

[ 0.05882353 0.64634146]

[ 0.21568627 0.73170732]

[ 0.25490196 0. ]

[ 0.15686275 0.69512195]

[ 1. 0.62195122]

[ 0.41176471 0.95121951]

[ 0.45098039 1. ]

[ 0.19607843 0.80487805]]

After Standardisation :

[[ 0.09536935 0.66527061]

[-1.15176827 -0.43586695]

[-0.93168516 -0.16058256]

[-0.34479687 0.16058256]

[-0.1980748 -2.59226136]

[-0.56487998 0.02294037]

[ 2.58964459 -0.25234403]

[ 0.38881349 0.98643574]

[ 0.53553557 1.16995867]

[-0.41815791 0.43586695]]

**Conclusion:**

Thus, we have studied feature scaling on given dataset.

**Assignment 5**

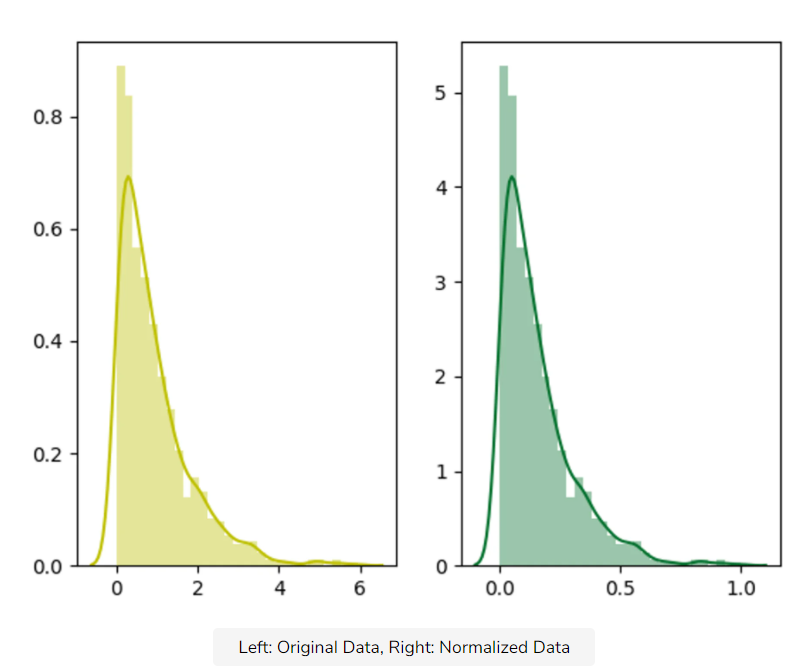
**Title:**Implement normalization on dataset.

**Theory:**

**Data normalization in Python**

Normalization refers to rescaling real-valued numeric attributes into a 00 to 11 range.

Data normalization is used in machine learning to make model training less sensitive to the scale of features. This allows our model to converge to better weights and, in turn, leads to a more accurate model.



Normalization makes the features more consistent with each other, which allows the model to predict outputs more accurately.

**Python Code**

Python provides the pre-processing library, which contains the normalize function to normalize the data. It takes an array in as an input and normalizes its values between 00 and 11. It then returns an output array with the same dimensions as the input.

fromsklearn import preprocessing

importnumpy as np

a = np.random.random((1, 4))

a = a\*20

print("Data = ", a)

# normalize the data attributes

normalized = preprocessing.normalize(a)

print("Normalized Data = ", normalized)

**Output**

Data = [[17.77383307 16.2338477 3.57341729 11.05743136]]

Normalized Data = [[0.66494409 0.60733107 0.13368657 0.41367406]]

**Conclusion:**

Thus, we have studiednormalization on dataset.

**Assignment 6**

**Title:**Perform proper data labelling operation on dataset.

**Theory:**

Data Labeling

Explore the uses and benefits of data labeling, including different approaches and best practices.

What is data labeling?

Data labeling, or data annotation, is part of the preprocessing stage when developing a [machine learning](https://www.ibm.com/cloud/learn/machine-learning#toc-what-is-ma-qhM6PX35) (ML) model. It requires the identification of raw data (i.e., images, text files, videos), and then the addition of one or more labels to that data to specify its context for the models, allowing the machine learning model to make accurate predictions.

Data labeling underpins different machine learning and deep learning use cases, including computer vision and natural language processing (NLP).

How does data labeling work?

Companies integrate software, processes and data annotators to clean, structure and label data. This training data becomes the foundation for machine learning models. These labels allow analysts to isolate variables within datasets, and this, in turn, enables the selection of optimal data predictors for ML models. The labels identify the appropriate data vectors to be pulled in for model training, where the model, then, learns to make the best predictions.

Along with machine assistance, data labeling tasks require “[human-in-the-loop (HITL)](https://www.research.ibm.com/publications/human-in-the-loop-business-modelling-for-emergent-external-factors)” participation. HITL leverages the judgment of human “data labelers” toward creating, training, fine-tuning and testing ML models. They help guide the data labeling process by feeding the models datasets that are most applicable to a given project.

Labeled data vs. unlabeled data

Computers use labeled and unlabeled data to train ML models, but [what is the difference](https://www.ibm.com/cloud/blog/supervised-vs-unsupervised-learning)?

* Labeled data is used in [supervised learning](https://www.ibm.com/cloud/learn/supervised-learning), whereas unlabeled data is used in [unsupervised learning](https://www.ibm.com/cloud/learn/unsupervised-learning) .
* Labeled data is more difficult to acquire and store (i.e. time consuming and expensive), whereas unlabeled data is easier to acquire and store.
* Labeled data can be used to determine actionable insights (e.g. forecasting tasks), whereas unlabeled data is more limited in its usefulness. Unsupervised learning methods can help discover new clusters of data, allowing for new categorizations when labeling.

Computers can also use combined data for semi-supervised learning, which reduces the need for manually labeled data while providing a large annotated dataset.

Data labeling approaches

Data labeling is a critical step in developing a high-performance ML model. Though labeling appears simple, it’s not always easy to implement. As a result, companies must consider multiple factors and methods to determine the best approach to labeling. Since each data labeling method has its pros and cons, a detailed assessment of task complexity, as well as the size, scope and duration of the project is advised.

Here are some paths to labeling your data:

* **Internal labeling** - Using in-house data science experts simplifies tracking, provides greater accuracy, and increases quality. However, this approach typically requires more time and favors large companies with extensive resources.
* **Synthetic labeling** - This approach generates new project data from pre-existing datasets, which enhances data quality and time efficiency. However, synthetic labeling requires extensive computing power, which can increase pricing.
* **Programmatic labeling** - This automated data labeling process uses scripts to reduce time consumption and the need for human annotation. However, the possibility of technical problems requires HITL to remain a part of the quality assurance (QA) process.
* **Outsourcing**- This can be an optimal choice for high-level temporary projects, but developing and managing a freelance-oriented workflow can also be time-consuming. Though freelancing platforms provide comprehensive candidate information to ease the vetting process, hiring managed data labeling teams provides pre-vetted staff and pre-built data labeling tools.
* **Crowdsourcing** - This approach is quicker and more cost-effective due to its micro-tasking capability and web-based distribution. However, worker quality, QA, and project management vary across crowdsourcing platforms. One of the most famous examples of crowdsourced data labeling is Recaptcha. This project was two-fold in that it controlled for bots while simultaneously improving data annotation of images. For example, a Recaptcha prompt would ask a user to identify all the photos containing a car to prove that they were human, and then this program could check itself based on the results of other users. The input of from these users provided a database of labels for an array of images.

Benefits and challenges of data labeling

The general tradeoff of data labeling is that while it can decrease a business’s time to scale, it tends to come at a cost. More accurate data generally improves model predictions, so despite its high cost, the value that it provides is usually well worth the investment. Since data annotation provides more context to datasets, it enhances the performance of exploratory data analysis as well as machine learning (ML) and artificial intelligence (AI) applications. For example, data labeling produces more relevant search results across search engine platforms and better product recommendations on e-commerce platforms. Let’s delve deeper into other key benefits and challenges:

*Benefits*

Data labeling provides users, teams and companies with greater context, quality and usability. More specifically, you can expect:

* **More Precise Predictions:** Accurate data labeling ensures better quality assurance within machine learning algorithms, allowing the model to train and yield the expected output. Otherwise, as the old saying goes, “garbage in, garbage out.” Properly labeled data  provide the “[ground truth](https://www.research.ibm.com/publications/designing-ground-truth-and-the-social-life-of-labels)” (i.e., how labels reflect “real world” scenarios) for testing and iterating subsequent models.
* **Better Data Usability:** Data labeling can also improve usability of data variables within a model. For example, you might reclassify a categorical variable as a binary variable to make it more consumable for a model.  Aggregating data in this way can optimize the model by reducing the number of model variables or enable the inclusion of control variables. Whether you’re using data to build computer vision models (i.e. putting bounding boxes around objects) or NLP models (i.e. classifying text for social sentiment), utilizing high-quality data is a top priority.

**Challenges**

Data labeling is not without its challenges. In particular, some of the most common challenges are:

* **Expensive and time-consuming:**While data labeling is critical for machine learning models, it can be costly from both a resource and time perspective. If a business takes a more automated approach, engineering teams will still need to set up data pipelines prior to data processing, and manual labeling will almost always be expensive and time-consuming.
* **Prone to Human-Error:** These labeling approaches are also subject to human-error (e.g. coding errors, manual entry errors), which can decrease the quality of data. This, in turn, leads to inaccurate data processing and modeling. Quality assurance checks are essential to maintaining data quality.

**Data labeling best practices**

No matter the approach, the following best practices optimize data labeling accuracy and efficiency:

* **Intuitive and streamlined task interfaces**minimize cognitive load and context switching for human labelers.
* **Consensus:**Measures the rate of agreement between multiple labelers(human or machine). A consensus score is calculated by dividing the sum of agreeing labels by the total number of labels per asset.
* **Label auditing:**Verifies the accuracy of labels and updates them as needed.
* [**Transfer learning:**](https://developer.ibm.com/technologies/artificial-intelligence/articles/transfer-learning-for-deep-learning/) Takes one or more pre-trained models from one dataset and applies them to another. This can include multi-task learning, in which multiple tasks are learned in tandem.
* **Active learning:** A category of ML algorithms and subset of semi-supervised learning that helps humans identify the most appropriate datasets. Active learning approaches include:
  + *Membership query synthesis*- Generates a synthetic instance and requests a label for it.
  + *Pool-based sampling* - Ranks all unlabeled instances according to informativeness measurement and selects the best queries to annotate.
  + *Stream-based selective sampling*- Selects unlabeled instances one by one, and labels or ignores them depending on their informativeness or uncertainty.

Data labeling use cases

Though data labeling can enhance accuracy, quality and usability in multiple contexts across industries, its more prominent use cases include:

* **Computer vision:** A field of AI that uses training data to build a computer vision model that enables image segmentation and category automation, identifies key points in an image and detects the location of objects. In fact, IBM offers a computer vision platform, [*Maximo Visual Inspection*](https://www.ibm.com/products/maximo/remote-monitoring#section-heading-4), that enables subject matter experts (SMEs) to label and train deep learning vision models that can be deployed in the cloud, edge devices, and local data centers. Computer vision is used in multiple industries - from energy and utilities to manufacturing and automotive. By 2022, this surging field is expected to reach a market value of $48.6 billion.
* **Natural language processing (NLP):** A branch of AI that combines computational linguistics with statistical, machine learning, and deep learning models to identify and tag important sections of text that generate training data for sentiment analysis, entity name recognition and optical character recognition. NLP is increasingly being used in enterprise solutions like spam detection, machine translation, [speech recognition](https://www.ibm.com/cloud/learn/speech-recognition), text summarization, virtual assistants and chatbots, and voice-operated GPS systems. This has made NLP a critical component in the evolution of mission-critical business processes.

IBM and data labeling

IBM offers more resources to help transcend data labeling challenges and maximize your overall data labeling experience.

* [IBM Cloud Annotations](https://cloud.annotations.ai/) (link resides outside IBM) - A collaborative open-source image annotation tool that uses AI models to help developers create fully labeled datasets of images, in real time, without manually drawing the labels.
* [IBM Cloud Object Storage](https://www.ibm.com/cloud/object-storage) - Encrypted at-rest and accessible from anywhere, it stores sensitive data and safeguards data integrity, availability and confidentiality via Information Dispersal Algorithm (IDA) and All-or-Nothing Transform (AONT).
* [IBM Watson](https://www.ibm.com/watson/products-services) - AI platform with NLP-driven tools and services that enable organizations to optimize employees’ time, automate complex business processes and gain critical business insights to predict future outcomes.

No matter your project size or timeline, IBM Cloud and IBM Watson can enhance your data training processes, expand your data classification efforts, and simplify complex forecasting models.

**Conclusion:**

Thus, we have studieddata labeling operation on dataset.

**Assignment 7**

**Title:**Implement principal component analysis algorithm.

**Theory:**

**Implementing PCA in Python with scikit-learn**

In this practical, we will learn about PCA (Principal Component Analysis) in Python with scikit-learn. Let’s start our learning step by step.

**WHY PCA?**

* When there are many input attributes, it is difficult to visualize the data. There is a very famous term ‘Curse of dimensionality in the machine learning domain.
* Basically, it refers to the fact that a higher number of attributes in a dataset adversely affects the accuracy and training time of the machine learning model.
* Principal Component Analysis (PCA) is a way to address this issue and is used for better data visualization and improving accuracy.

**How does PCA work?**

* PCA is an unsupervised pre-processing task that is carried out before applying any ML algorithm. PCA is based on “orthogonal linear transformation” which is a mathematical technique to project the attributes of a data set onto a new coordinate system. The attribute which describes the most variance is called the first principal component and is placed at the first coordinate.
* Similarly, the attribute which stands second in describing variance is called a second principal component and so on. In short, the complete dataset can be expressed in terms of principal components. Usually, more than 90% of the variance is explained by two/three principal components.
* Principal component analysis, or PCA, thus converts data from high dimensional space to low dimensional space by selecting the most important attributes that capture maximum information about the dataset.

**Python Implementation:**

* To implement PCA in Scikit learn, it is essential to standardize/normalize the data before applying PCA.
* PCA is imported from sklearn.decomposition. We need to select the required number of principal components.
* Usually, n\_components is chosen to be 2 for better visualization but it matters and depends on data.
* By the fit and transform method, the attributes are passed.
* The values of principal components can be checked using components\_ while the variance explained by each principal component can be calculated using explained\_variance\_ratio.

**1. Import all the libraries**

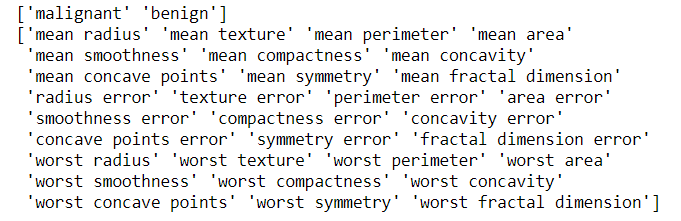
|  |
| --- |
| # import all libraries  importpandas as pd  importnumpy as np  importmatplotlib.pyplot as plt  %matplotlib inline  fromsklearn.decomposition importPCA  fromsklearn.preprocessing importStandardScaler |

**2. Loading Data**

Load the breast\_cancer dataset from sklearn.datasets. It is clear that the dataset has 569 data items with 30 input attributes. There are two output classes-benign and malignant. Due to 30 input features, it is impossible to visualize this data

|  |
| --- |
| #import the breast \_cancer dataset  fromsklearn.datasets importload\_breast\_cancer  data=load\_breast\_cancer()  data.keys()    # Check the output classes  print(data['target\_names'])    # Check the input attributes  print(data['feature\_names']) |

**Output:**



**3. Apply PCA**

* Standardizethe dataset prior toPCA.
* Import PCA from sklearn.decomposition.
* Choose the number of principal components.

Let us select it to 3. After executing this code, we get to know that the dimensions of x are (569,3) while the dimension of actual data is (569,30). Thus, it is clear that with PCA, the number of dimensions has reduced to 3 from 30. If we choose n\_components=2, the dimensions would be reduced to 2.

|  |
| --- |
| # construct a dataframe using pandas  df1=pd.DataFrame(data['data'],columns=data['feature\_names'])  # Scale data before applying PCA  scaling=StandardScaler()  # Use fit and transform method  scaling.fit(df1)  Scaled\_data=scaling.transform(df1)  # Set the n\_components=3  principal=PCA(n\_components=3)  principal.fit(Scaled\_data)  x=principal.transform(Scaled\_data)    # Check the dimensions of data after PCA  print(x.shape) |

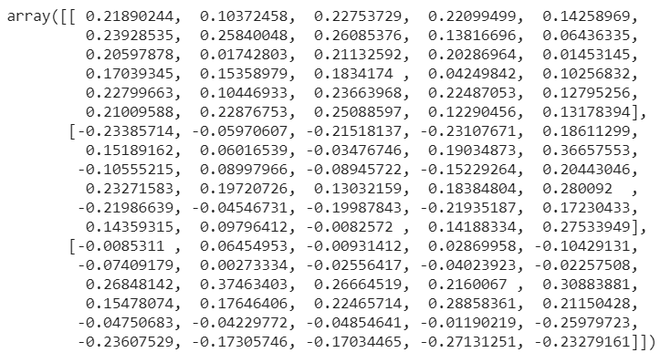
**Output:**

**(569,3)**

**4. Check Components**

The principal.components\_ provide an array in which the number of rows tells the number of principal components while the number of columns is equal to the number of features in actual data.  We can easily see that there are three rows as n\_components was chosen to be 3. However, each row has 30 columns as in actual data.

|  |
| --- |
| # Check the values of eigen vectors  # prodeced by principal components  principal.components\_ |

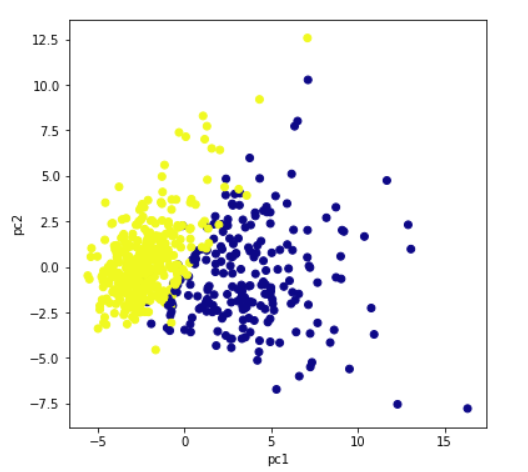


**5. Plot the components (Visualization)**

Plot the principal components for better data visualization.  Though we had taken n\_components =3, here we are plotting a 2d graph as well as 3d using first two principal components and 3 principal components respectively. For three principal components, we need to plot a 3d graph. The colors show the 2 output classes of the original dataset-benign and malignant. It is clear that principal components show clear separation between two output classes.

|  |
| --- |
| plt.figure(figsize=(10,10))  plt.scatter(x[:,0],x[:,1],c=data['target'],cmap='plasma')  plt.xlabel('pc1')  plt.ylabel('pc2') |

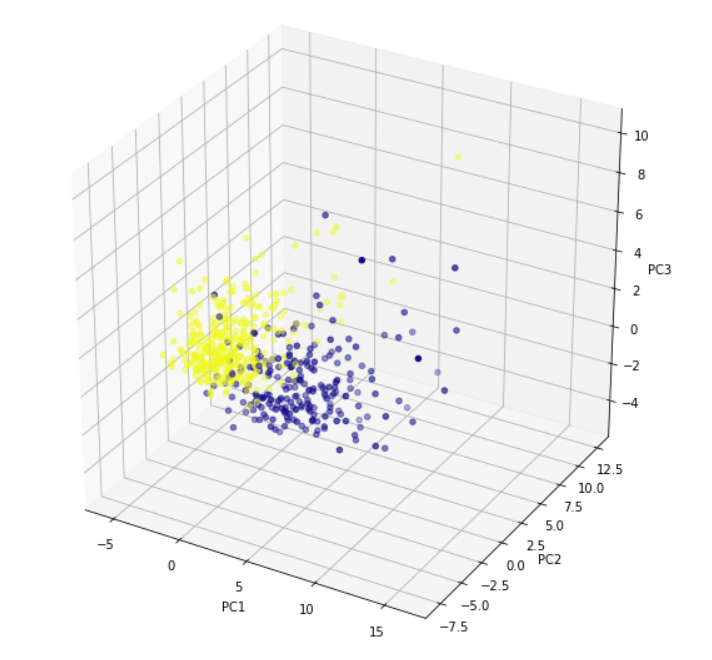
**Output:**



For three principal components, we need to plot a 3d graph. x[:,0] signifies the first principal component. Similarly, x[:,1] and x[:,2] represent the second and the third principal component.

|  |
| --- |
| # import relevant libraries for 3d graph  frommpl\_toolkits.mplot3d importAxes3D  fig =plt.figure(figsize=(10,10))  # choose projection 3d for creating a 3d graph  axis =fig.add\_subplot(111, projection='3d')  # x[:,0]is pc1,x[:,1] is pc2 while x[:,2] is pc3  axis.scatter(x[:,0],x[:,1],x[:,2], c=data['target'],cmap='plasma')  axis.set\_xlabel("PC1", fontsize=10)  axis.set\_ylabel("PC2", fontsize=10)  axis.set\_zlabel("PC3", fontsize=10) |

**Output:**



**6. Calculate variance ratio**

Explained\_variance\_ratio provides an idea of how much variation is explained by principal components.

|  |
| --- |
| # check how much variance is explained by each principal component  print(principal.explained\_variance\_ratio\_) |

**Output:**

array([0.44272026, 0.18971182, 0.09393163])

**Conclusion:**

Thus, we have implemented principal component analysis algorithm.

**Assignment 8**

**Title:**Perform Encoding categorical features on given dataset

**Theory:**

Overview

* Understand what is Categorical Data Encoding
* Learn different encoding techniques and when to use them

Introduction

The performance of a machine learning model not only depends on the model and the hyperparameters but also on how we process and feed different types of variables to the model. Since most machine learning models only accept numerical variables, preprocessing the categorical variables becomes a necessary step. We need to convert these categorical variables to numbers such that the model is able to understand and extract valuable information.

A typical data scientist spends 70 – 80% of his time cleaning and preparing the data. And converting categorical data is an unavoidable activity. It not only elevates the model quality but also helps in better feature engineering. Now the question is, how do we proceed? Which categorical data encoding method should we use?

In this practical, We will be studying various types of categorical data encoding methods with implementation in Python.

Table of content

* What is Categorical Data?
* Label Encoding or Ordinal Encoding
* One hot Encoding
* Dummy Encoding
* Effect Encoding
* Binary Encoding
* BaseN Encoding
* Hash Encoding
* Target Encoding

What is categorical data?

Since we are going to be working on categorical variables in this article, here is a quick refresher on the same with a couple of examples. Categorical variables are usually represented as ‘strings’ or ‘categories’ and are finite in number. Here are a few examples:

1. The city where a person lives: Delhi, Mumbai, Ahmedabad, Bangalore, etc.
2. The department a person works in: Finance, Human resources, IT, Production.
3. The highest degree a person has: High school, Diploma, Bachelors, Masters, PhD.
4. The grades of a student:  A+, A, B+, B, B- etc.

In the above examples, the variables only have definite possible values. Further, we can see there are two kinds of categorical data-

* **Ordinal Data:** The categories have an inherent order
* **Nominal Data:** The categories do not have an inherent order

In Ordinal data, while encoding, one should retain the information regarding the order in which the category is provided. Like in the above example the highest degree a person possesses, gives vital information about his qualification. The degree is an important feature to decide whether a person is suitable for a post or not.

While encoding Nominal data, we have to consider the presence or absence of a feature. In such a case, no notion of order is present. For example, the city a person lives in. For the data, it is important to retain where a person lives. Here, We do not have any order or sequence. It is equal if a person lives in Delhi or Bangalore.

For encoding categorical data, we have a python package category\_encoders. The following code helps you install easily.

**pip install category\_encoders**

Label Encoding or Ordinal Encoding

We use this categorical data encoding technique when the categorical feature is ordinal. In this case, retaining the order is important. Hence encoding should reflect the sequence.

In Label encoding, each label is converted into an integer value. We will create a variable that contains the categories representing the education qualification of a person.

importcategory\_encoders as ce

import pandas as pd

train\_df=pd.DataFrame({'Degree':['High school','Masters','Diploma','Bachelors','Bachelors','Masters','Phd','High school','High school']})

# create object of Ordinalencoding

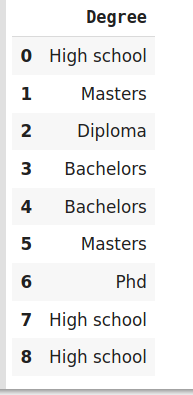
encoder= ce.OrdinalEncoder(cols=['Degree'],return\_df=True,

mapping=[{'col':'Degree',

'mapping':{'None':0,'High school':1,'Diploma':2,'Bachelors':3,'Masters':4,'phd':5}}])

#Original data

train\_df



#fit and transform train data

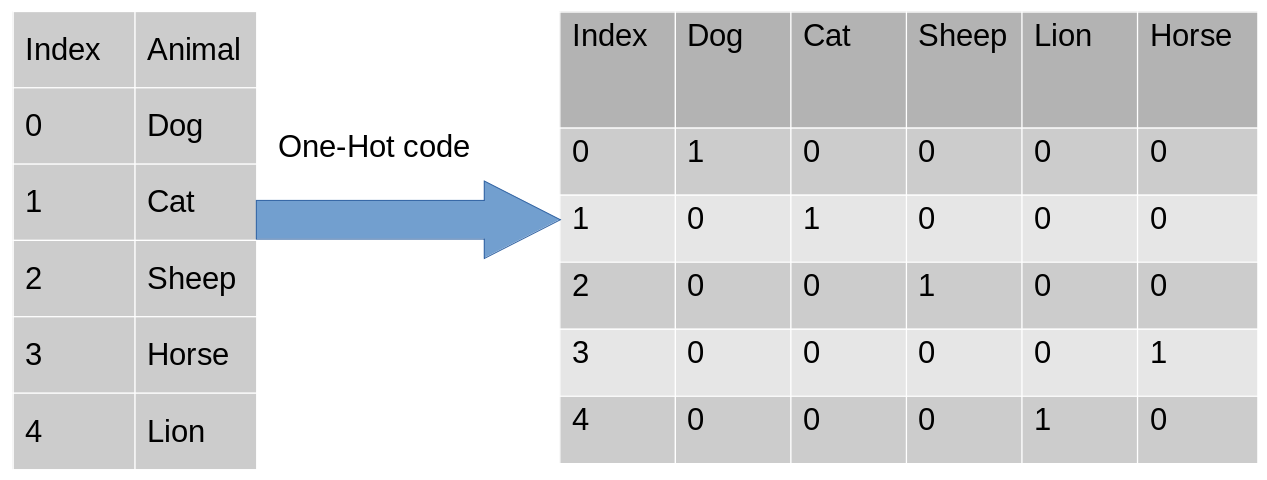
df\_train\_transformed = encoder.fit\_transform(train\_df)



One Hot Encoding

We use this categorical data encoding technique when the features are nominal(do not have any order). In one hot encoding, for each level of a categorical feature, we create a new variable. Each category is mapped with a binary variable containing either 0 or 1. Here, 0 represents the absence, and 1 represents the presence of that category.

These newly created binary features are known as**Dummy variables.** The number of dummy variables depends on the levels present in the categorical variable. This might sound complicated. Let us take an example to understand this better. Suppose we have a dataset with a category animal, having different animals like Dog, Cat, Sheep, Cow, Lion. Now we have to one-hot encode this data.



After encoding, in the second table, we have dummy variables each representing a category in the feature Animal. Now for each category that is present, we have 1 in the column of that category and 0 for the others. Let’s see how to implement a one-hot encoding in python.

importcategory\_encoders as ce

import pandas as pd

data=pd.DataFrame({'City':[

'Delhi','Mumbai','Hydrabad','Chennai','Bangalore','Delhi','Hydrabad','Bangalore','Delhi'

]})

#Create object for one-hot encoding

encoder=ce.OneHotEncoder(cols='City',handle\_unknown='return\_nan',return\_df=True,use\_cat\_names=True)

#Original Data

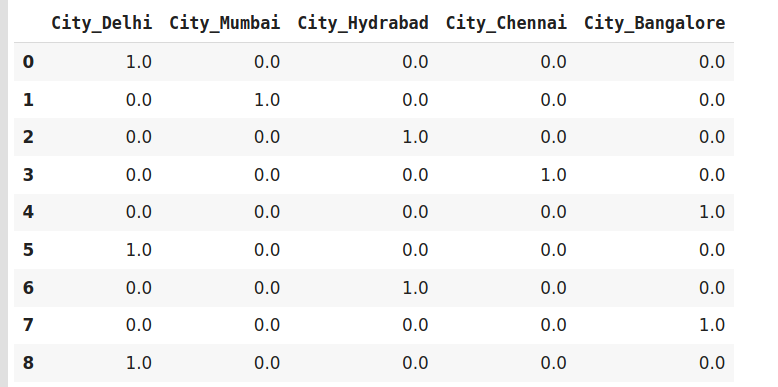
data



#Fit and transform Data

data\_encoded = encoder.fit\_transform(data)

data\_encoded

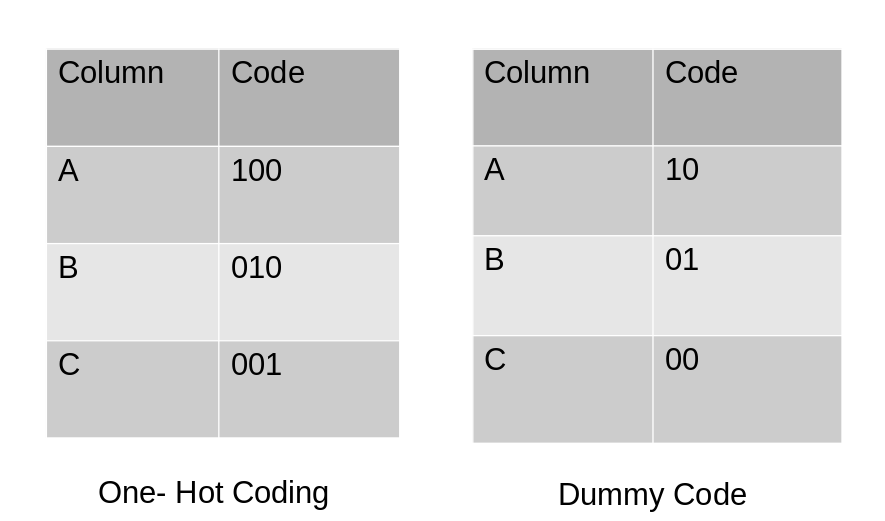


Now let’s move to another very interesting and widely used encoding technique i.eDummy encoding.

Dummy Encoding

Dummy coding scheme is similar to one-hot encoding. This categorical data encoding method transforms the categorical variable into a set of binary variables (also known as dummy variables). In the case of one-hot encoding, for N categories in a variable, it uses N binary variables. The dummy encoding is a small improvement over one-hot-encoding. Dummy encoding uses N-1 features to represent N labels/categories.

To understand this better let’s see the image below. Here we are coding the same data using both one-hot encoding and dummy encoding techniques. While one-hot uses 3 variables to represent the data whereas dummy encoding uses 2 variables to code 3 categories.



Let us implement it in python.

importcategory\_encoders as ce

import pandas as pd

data=pd.DataFrame({'City':['Delhi','Mumbai','Hyderabad','Chennai','Bangalore','Delhi,'Hyderabad']})

#Original Data

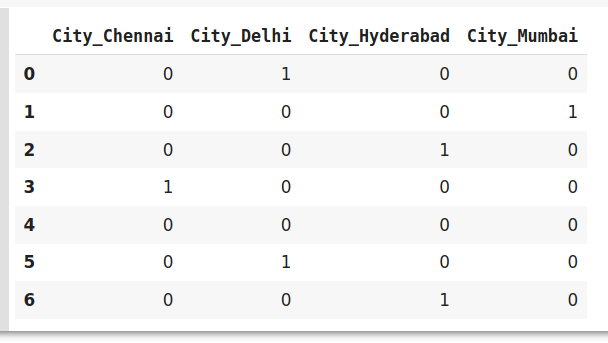
data



#encode the data

data\_encoded=pd.get\_dummies(data=data,drop\_first=True)

data\_encoded



Here using *drop\_first*  argument, we are representing the first label Bangalore using 0.

Drawbacks of  One-Hot and Dummy Encoding

One hot encoder and dummy encoder are two powerful and effective encoding schemes. They are also very popular among the data scientists, But may not be as effective when-

1. A large number of levels are present in data. If there are multiple categories in a feature variable in such a case we need a similar number of dummy variables to encode the data. For example, a column with 30 different values will require 30 new variables for coding.
2. If we have multiple categorical features in the dataset similar situation will occur and again we will end to have several binary features each representing the categorical feature and their multiple categories e.g a dataset having 10 or more categorical columns.

In both the above cases, these two encoding schemes introduce sparsity in the dataset i.e several columns having 0s and a few of them having 1s. In other words, it creates multiple dummy features in the dataset without adding much information.

Also, they might lead to a Dummy variable trap. It is a phenomenon where features are highly correlated. That means using the other variables, we can easily predict the value of a variable.

Due to the massive increase in the dataset, coding slows down the learning of the model along with deteriorating the overall performance that ultimately makes the model computationally expensive. Further, while using tree-based models these encodings are not an optimum choice.

Effect Encoding:

This encoding technique is also known as **Deviation Encoding** or **Sum Encoding.** Effect encoding is almost similar to dummy encoding, with a little difference. In dummy coding, we use 0 and 1 to represent the data but in effect encoding, we use three values i.e. 1,0, and -1.

The row containing only 0s in dummy encoding is encoded as -1 in effect encoding.  In the dummy encoding example, the city Bangalore at index 4  was encoded as 0000. Whereas in effect encoding it is represented by -1-1-1-1.

Let us see how we implement it in python-

importcategory\_encoders as ce

import pandas as pd

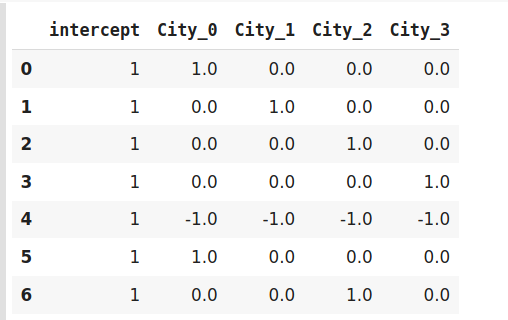
data=pd.DataFrame({'City':['Delhi','Mumbai','Hyderabad','Chennai','Bangalore','Delhi,'Hyderabad']}) encoder=ce.sum\_coding.SumEncoder(cols='City',verbose=False,)

#Original Data

data



encoder.fit\_transform(data)



Effect encoding is an advanced technique. In case you are interested to know more about effect encoding, refer to [this](https://www.researchgate.net/publication/256349393_Categorical_Variables_in_Regression_Analysis_A_Comparison_of_Dummy_and_Effect_Coding) interesting paper.

Hash Encoder

To understand Hash encoding it is necessary to know about hashing. Hashing is the transformation of arbitrary size input in the form of a fixed-size value. We use hashing algorithms to perform hashing operations i.e to generate the hash value of an input. Further, hashing is a one-way process, in other words, one can not generate original input from the hash representation.

Hashing has several applications like data retrieval, checking data corruption, and in data encryption also. We have multiple hash functions available for example Message Digest (MD, MD2, MD5), Secure Hash Function (SHA0, SHA1, SHA2), and many more.

Just like one-hot encoding, the Hash encoder represents categorical features using the new dimensions. Here, the user can fix the number of dimensions after transformation using***n\_component*** argument. Here is what I mean – A feature with 5 categories can be represented using N new features similarly, a feature with 100 categories can also be transformed using N new features. Doesn’t this sound amazing?

By default, the Hashing encoder uses **the md5**hashing algorithm but a user can pass any algorithm of his choice. If you want to explore the md5 algorithm, I suggest [this](https://ieeexplore.ieee.org/document/5474379) paper.

importcategory\_encoders as ce

import pandas as pd

#Create the dataframe

data=pd.DataFrame({'Month':['January','April','March','April','Februay','June','July','June','September']})

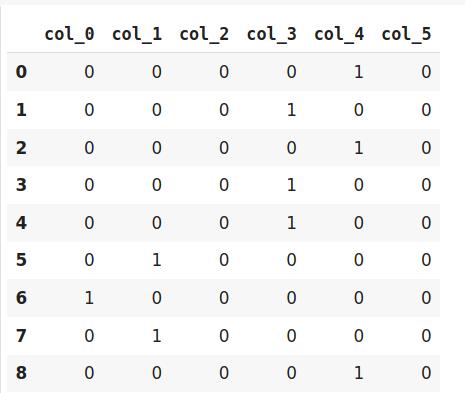
#Create object for hash encoder

encoder=ce.HashingEncoder(cols='Month',n\_components=6)



#Fit and Transform Data

encoder.fit\_transform(data)



Since Hashing transforms the data in lesser dimensions, it may lead to loss of information. Another issue faced by hashing encoder is the **collision.** Since here, a large number of features are depicted into lesser dimensions, hence multiple values can be represented by the same hash value, this is known as a collision.

Moreover, hashing encoders have been very successful in some Kaggle competitions. It is great to try if the dataset has high cardinality features.

Binary Encoding

Binary encoding is a combination of Hash encoding and one-hot encoding. In this encoding scheme, the categorical feature is first converted into numerical using an ordinal encoder. Then the numbers are transformed in the binary number. After that binary value is split into different columns.

Binary encoding works really well when there are a high number of categories. For example the cities in a country where a company supplies its products.

#Import the libraries

importcategory\_encoders as ce

import pandas as pd

#Create the Dataframe

data=pd.DataFrame({'City':['Delhi','Mumbai','Hyderabad','Chennai','Bangalore','Delhi','Hyderabad','Mumbai','Agra']})

#Create object for binary encoding

encoder= ce.BinaryEncoder(cols=['city'],return\_df=True)

#Original Data

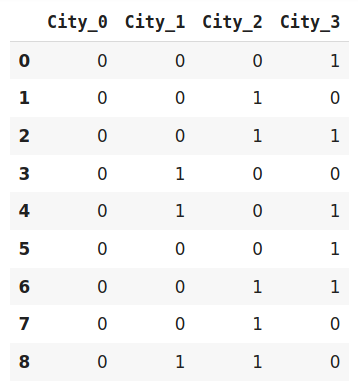
data



#Fit and Transform Data

data\_encoded=encoder.fit\_transform(data)

data\_encoded



Binary encoding is a memory-efficient encoding scheme as it uses fewer features than one-hot encoding. Further, It reduces the curse of dimensionality for data with high cardinality.

Base N Encoding

Before diving into BaseN encoding let’s first try to understand what is Base here?

In the numeral system, the Base or the radix is the number of digits or a combination of digits and letters used to represent the numbers. The most common base we use in our life is 10  or decimal system as here we use 10 unique digits i.e 0 to 9 to represent all the numbers. Another widely used system is binary i.e. the base is 2. It uses 0 and 1 i.e 2 digits to express all the numbers.

For Binary encoding, the Base is 2 which means it converts the numerical values of a category into its respective Binary form. If you want to change the Base of encoding scheme you may use Base N encoder. In the case when categories are more and binary encoding is not able to handle the dimensionality then we can use a larger base such as 4 or 8.

#Import the libraries

importcategory\_encoders as ce

import pandas as pd

#Create the dataframe

data=pd.DataFrame({'City':['Delhi','Mumbai','Hyderabad','Chennai','Bangalore','Delhi','Hyderabad','Mumbai','Agra']})

#Create an object for Base N Encoding

encoder= ce.BaseNEncoder(cols=['city'],return\_df=True,base=5)

#Original Data

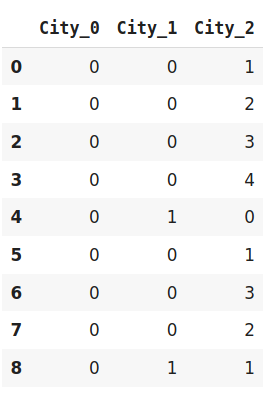
data



#Fit and Transform Data

data\_encoded=encoder.fit\_transform(data)

data\_encoded



In the above example, I have used base 5 also known as the Quinary system. It is similar to the example of Binary encoding. While Binary encoding represents the same data by 4 new features the BaseN encoding uses only 3 new variables.

Hence BaseN encoding technique further reduces the number of features required to efficiently represent the data and improving memory usage. The default Base for Base N is 2 which is equivalent to Binary Encoding.

Target Encoding

Target encoding is a Baysian encoding technique.

Bayesian encoders use information from dependent/target variables to encode the categorical data.

In target encoding, we calculate the mean of the target variable for each category and replace the category variable with the mean value. In the case of the categorical target variables, the posterior probability of the target replaces each category..

#import the libraries

import pandas as pd

importcategory\_encoders as ce

#Create the Dataframe

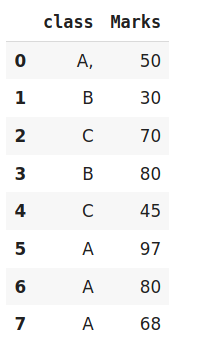
data=pd.DataFrame({'class':['A,','B','C','B','C','A','A','A'],'Marks':[50,30,70,80,45,97,80,68]})

#Create target encoding object

encoder=ce.TargetEncoder(cols='class')

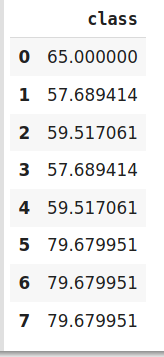
#Original Data

Data



#Fit and Transform Train Data

encoder.fit\_transform(data['class'],data['Marks'])



We perform Target encoding for train data only and code the test data using results obtained from the training dataset. Although, a very efficient coding system, it has the following **issues** responsible for deteriorating the model performance-

1. It can lead to target leakage or overfitting. To address overfitting we can use different techniques.
   1. In the leave one out encoding, the current target value is reduced from the overall mean of the target to avoid leakage.
   2. In another method, we may introduce some Gaussian noise in the target statistics. The value of this noise is hyperparameter to the model.
2. The second issue, we may face is the improper distribution of categories in train and test data. In such a case, the categories may assume extreme values. Therefore the target means for the category are mixed with the marginal mean of the target.

**Conclusion**

To summarize, encoding categorical data is an unavoidable part of the feature engineering. It is more important to know what coding scheme we should use. Having into consideration the dataset we are working with and the model we are going to use. In this practical, we have seen various encoding techniques along with their issues and suitable use cases.