Data Ingestion

Thursday, January 5, 2017 3:40 PM

Data Ingestion

Data

Index	Country	Age	Salary	Purcha
0	France	44	7.2e+04	No
1	Spain	27	4.8e+04	Yes
2	Germany	30	5.4e+04	No
3	Spain	38	6.1e+04	No
4	Germany	40	nan	Yes
5	France	35	5.8e+04	Yes
6	Spain	nan	5.2e+04	No
7	France	48	7.9e+04	Yes
8	Germany	50	8.3e+04	No
9	France	37	6.7e+04	Yes

Python

Use pandas

```
Import pandas as pd
Dataset = pd.read_csv('Data.csv')
X = dataset.iloc[:,:-1].values
Y = dataset.iloc[:,3].values
```

Iloc -> Integer location, used for addressing data frame

R

Default ingestion

For setting working directory, use

Setwd()

For ingesting data, use

```
Data = read.csv("Data.csv")
```

Data[1:5, 2:3] -> Indexing locations, splicing

Handling Missing Data

Friday, January 6, 2017 9:13 AM

Python

Imputer

Definition : Imputer(self, missing_values="NaN", strategy="mean", axis=0, verbose=0, copy=True)

The imputation strategy.

- If “mean”, then replace missing values using the mean along the axis.
- If “median”, then replace missing values using the median along the axis.
- If “most_frequent”, then replace missing using the most frequent value along the axis.

Axis = 0 -> Column, Axis = 1 -> Rows

```
From sklearn.preprocessing import Imputer
Imputer = Imputer(missing_values='NaN',strategy='mean',axis=0)
Imputer = imputer.fit(X[:,1:3])
X[:,1:3] = imputer.transform(X[:,1:3])
```

What does it do ?

Select columns which contain NaN's and fit the imputer object to it. Now transform to get rid of NaN values by replacing it with the mean value of the column.

R

Index columns using

Data$Age

```
dataset$Age = ifelse(is.na(dataset$Age), mean(dataset$Age, na.rm = T), dataset
$Age)
dataset$Salary = ifelse(is.na(dataset$Salary), mean(dataset$Salary, na.rm=T),
dataset$Salary)
```

Data after imputing

	Country	Age	Salary	Purchased
1	France	44.00000	72000.00	No
2	Spain	27.00000	48000.00	Yes
3	Germany	30.00000	54000.00	No
4	Spain	38.00000	61000.00	No
5	Germany	40.00000	63777.78	Yes
6	France	35.00000	58000.00	Yes
7	Spain	38.77778	52000.00	No
8	France	48.00000	79000.00	Yes
9	Germany	50.00000	83000.00	No
10	France	37.00000	67000.00	Yes

Encoding Categorical Data

Friday, January 6, 2017 9:14 AM

Python

LabelEncoder does this job of encoding columns having categorical data into integer representations

```
From sklearn.preprocessing import LabelEncoder
labelencoder_X = LabelEncoder()
X[:,0] = labelencoder_X.fit_transform(X[:,0])

labelencoder_y=LabelEncoder()
y=labelencoder_y.fit_transform(y)
```

['France' 'Spain' 'Germany' 'Spain' 'Germany' 'France' 'Spain' 'France' 'Germany' 'France'] -> becomes -> [0 2 1 2 1 0 2 0 1 0]

Problem with this is Machine learning models are mathematical and therefore the magnitude of numerical values matter. If this is the case, then the ML model will think that Spain is greater than France which is not the case.

For preventing this, we use what is called as dummy encoding

Country
France
Spain
Germany
Spain
Germany
France
Spain
France
Germany
France

France	Germany	Spain
1	0	0
0	0	1
0	1	0
0	0	1
0	1	0
1	0	0
0	0	1
1	0	0
0	1	0
1	0	0

To solve this issue, we use the OneHotEncoder class.

Definition : `OneHotEncoder(n_values="auto", categorical_features="all", dtype=np.float, sparse=True,handle_unknown='error')`

categorical_features: "all" or array of indices or mask

Specify what features are treated as categorical.

- 'all' (default): All features are treated as categorical.
- array of indices: Array of categorical feature indices.
- mask: Array of length n_features and with dtype=bool.

```
From sklearn.preprocessing import OneHotEncoder
Onehotencoder = OneHotEncoder(categorical_features=[0])
X = onehotencoder.fit_transform(X).toarray()
```

	0	1	2	3	4
0	1.000	0.000	0.000	44.000	72000.000
1	0.000	0.000	1.000	27.000	48000.000
2	0.000	1.000	0.000	30.000	54000.000
3	0.000	0.000	1.000	38.000	61000.000
4	0.000	1.000	0.000	40.000	63777.778
5	1.000	0.000	0.000	35.000	58000.000
6	0.000	0.000	1.000	38.778	52000.000
7	1.000	0.000	0.000	48.000	79000.000
8	0.000	1.000	0.000	50.000	83000.000
9	1.000	0.000	0.000	37.000	67000.000

NOTE :- For OneHotEncoder, we need to send the data which has already been processed by the LabelEncoder. We can't provide categorical values to this directly.

R

In R it is very simple, categorical variables can be represented as Factors.

factor(x = character(), levels, labels = levels, exclude = NA, ordered = is.ordered(x), nmax = NA)

x	a vector of data, usually taking a small number of distinct values.
levels	an optional vector of the values (as character strings) thatx might have taken. The default is the unique set of values taken by as.character(x), sorted into increasing order*of x*. Note that this set can be specified as smaller thansort(unique(x)).
labels	*either* an optional character vector of labels for the levels (in the same order as levels after removing those inexclude), *or* a character string of length 1.

```
dataset$Country = factor(dataset$Country, levels = c('France','Spain','Germany'), labels = c(1, 2, 3))
dataset$Purchased = factor(dataset$Purchased, levels = c('No','Yes'), labels = c(0, 1))
```

Country	Age	Salary	Purchased
1	44.00000	72000.00	0
2	27.00000	48000.00	1
3	30.00000	54000.00	0
2	38.00000	61000.00	0
3	40.00000	63777.78	1
1	35.00000	58000.00	1
2	38.77778	52000.00	0
1	48.00000	79000.00	1
3	50.00000	83000.00	0
1	37.00000	67000.00	1

Splitting the Dataset

Friday, January 6, 2017 9:15 AM

We need to split the data into 2 different sets. The training set on which the machine learning models learns and the test set used for testing the performance of what the Machine Learning model has learnt

Python

Definition : train_test_split(*arrays, **options)

*arrays : sequence of indexables with same length / shape[0]

allowed inputs are lists, numpy arrays, scipy-sparse matrices or pandas dataframes.

New in version 0.16: preserves input type instead of always casting to numpy array.

test_size: float, int, or None (default is None)

If float, should be between 0.0 and 1.0 and represent the proportion of the dataset to include in the test split. If int, represents the absolute number of test samples. If None, the value is automatically set to the complement of the train size. If train size is also None, test size is set to 0.25.

```
From sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2,random_state=0)
```

R

For R we need a package called caTools.

To Install this package, use

```
install.packages('caTools')
```

To use this library/package, we need to use

```
library(caTools)
set.seed(123)
split = sample.split(dataset$Purchased, SplitRatio = 0.8)
training_set = subset(dataset, split == TRUE)
test_set = subset(dataset, split == FALSE)
```

Feature Scaling

Saturday, January 14, 2017 8:18 AM

We need to scale all the features roughly to the same scale in order for the machine learning models to work well and converge faster. This is a very important step in preprocessing.

	Country	Age	Salary	Purchased
1	France	44.00000	72000.00	No
2	Spain	27.00000	48000.00	Yes
3	Germany	30.00000	54000.00	No
4	Spain	38.00000	61000.00	No
5	Germany	40.00000	63777.78	Yes
6	France	35.00000	58000.00	Yes
7	Spain	38.77778	52000.00	No
8	France	48.00000	79000.00	Yes
9	Germany	50.00000	83000.00	No
10	France	37.00000	67000.00	Yes

Here, the range of salary and age are completely different. This leads to an inherent bias for the model.

Standardisation	Normalisation
$x_{\text{stand}} = \frac{x - \text{mean}(x)}{\text{standard deviation } (x)}$	$x_{\text{norm}} = \frac{x - \min(x)}{\max(x) - \min(x)}$

Python

```
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.fit_transform(X_test)
```

We don't need to apply feature scaling on y. Because for this instance, y is a categorical variable. If it is a regression problem, then we need to apply feature scaling.

R

Very simple. Identify numeric columns that needs to be scaled. Factors cannot be scaled.
Once the columns are selected, use scale function to scale the columns.

```
training_set[,2:3] = scale(training_set[,2:3])
Test_set[,2:3] = scale(test_set[,2:3])
```

Country	Age	Salary	Purchased
1	0.90101716	0.9392746	0
2	-1.58847494	-1.3371160	1
3	-1.14915281	-0.7680183	0
2	0.02237289	-0.1040711	0
3	0.31525431	0.1594000	1
2	0.13627122	-0.9577176	0
1	1.48678000	1.6032218	1
1	-0.12406783	0.4650265	1