Chapter 1 ML - Data Processing

March 30, 2020

1 Chapter 1 Data Preprocessing

- Data set has to be preprocessed before putting into ML algorithms
- dataset has attributes set, comprises of dependent (D) and independent variables (I)
- there exists a map (f) such that $f : I \to D$
- ML algorithms find f

1.1 Import Libraries

- Numpy : Numeric , math computation
- matplotlib : plotting
- pandas : data import and management

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

1.2 Importing the dataset

for the test purpose I'm using a open source dataset from https://www.superdatascience.com/pages/machine-learning stored as `/ds/Data.csv`

[2]:
```
dataset=pd.read_csv('ds/Data.csv')
```

[3]:
```
dataset.head(5)
```

[3]:

	Country	Age	Salary	Purchased
0	France	44.0	72000.0	No
1	Spain	27.0	48000.0	Yes
2	Germany	30.0	54000.0	No
3	Spain	38.0	61000.0	No
4	Germany	40.0	NaN	Yes

1.2.1 Seperate Dependent and Independent variables

[4]:
```
# Independent Variables
X=dataset.iloc[:,:-1].values   #[all rows, all col but last one]

# Dependent Variables
Y=dataset.iloc[:,-1].values    #[all roes, last col only]
```

[5]:
```
pd.DataFrame(X)
```

[5]:

	0	1	2
0	France	44	72000
1	Spain	27	48000
2	Germany	30	54000
3	Spain	38	61000
4	Germany	40	NaN
5	France	35	58000
6	Spain	NaN	52000
7	France	48	79000
8	Germany	50	83000
9	France	37	67000

[6]:
```
pd.DataFrame(Y)
```

[6]:

	0
0	No
1	Yes
2	No
3	No
4	Yes
5	Yes
6	No
7	Yes
8	No
9	Yes

1.3 Dealing with missing data

some times the data set may contains missing data. there are two strategies 1. Delete the rows with missing data (Dangerous) 2. fill with mean value of the given attributes (Preferred)

[7]:
```
# import the class
from sklearn.preprocessing import Imputer

# create an object
imputer = Imputer(missing_values='NaN', strategy='mean', axis=0)
```

```
# apply impurter
imputer = imputer.fit(X[:,1:3]) #specify the target attribute with missing data␣
 ↪(0 index)
X[:,1:3] = imputer.transform(X[:,1:3])

pd.DataFrame(X)
```

[7]:

	0	1	2
0	France	44	72000
1	Spain	27	48000
2	Germany	30	54000
3	Spain	38	61000
4	Germany	40	63777.8
5	France	35	58000
6	Spain	38.7778	52000
7	France	48	79000
8	Germany	50	83000
9	France	37	67000

the impurter object takes * `missing_value` argument name. this is the name it look for replacing. * `strategy` is by default `mean`, however other startegies are (Median, Most Frequent) * `Axis` 0 = mean along the colums (Veritcal) , 1 = mean along rows (Horizontal)

1.4 Encode Catagorical Data

since ML models are based on numeric computation. Thus, it is nessesary to encode any string value into numbers.

[8]:
```
from sklearn.preprocessing import LabelEncoder

#create an object
le_X = LabelEncoder()
X[:,0] = le_X.fit_transform(X[:,0])   # transform 0th col of X and replace␣
 ↪original
pd.DataFrame(X)
```

[8]:

	0	1	2
0	0	44	72000
1	2	27	48000
2	1	30	54000
3	2	38	61000
4	1	40	63777.8
5	0	35	58000
6	2	38.7778	52000
7	0	48	79000
8	1	50	83000

```
9  0       37    67000
```

[9]:
```
Y=le_X.fit_transform(Y[:])
pd.DataFrame(Y)
```

[9]:
```
   0
0  0
1  1
2  0
3  0
4  1
5  1
6  0
7  1
8  0
9  1
```

- Now this may lead to another problem, since the transformation will create a ordered number list for each identical item. the model may try to find corelation between them which is absolutely makes no sense (Since if the categories are not always ordinal)
- in such a case we use **Dummy Encoding** where each type is treated as a seperate column and encoded accordingly

1.4.1 One Hot Encoding

one hot encoding is used to perform "Dummy Encoding". the object has follwing attributes * categorical_features : Specify which column you want to encode

[10]:
```
from sklearn.preprocessing import OneHotEncoder
OHE_X=OneHotEncoder(categorical_features=[0]) # specify target column []
X= OHE_X.fit_transform(X).toarray()

pd.DataFrame(X)
```

[10]:

	0	1	2	3	4
0	1.0	0.0	0.0	44.000000	72000.000000
1	0.0	0.0	1.0	27.000000	48000.000000
2	0.0	1.0	0.0	30.000000	54000.000000
3	0.0	0.0	1.0	38.000000	61000.000000
4	0.0	1.0	0.0	40.000000	63777.777778
5	1.0	0.0	0.0	35.000000	58000.000000
6	0.0	0.0	1.0	38.777778	52000.000000
7	1.0	0.0	0.0	48.000000	79000.000000
8	0.0	1.0	0.0	50.000000	83000.000000
9	1.0	0.0	0.0	37.000000	67000.000000

- Use label Encoder if variable is (yes/no) or ordinal categorigal

- Use OHE if variable has no corelation and categorigal

1.5 Train Test Split

- ML algorithms learns model from Data sets
- it's not a good practice for ML to perform good on Dataset but not on difference data
- This occurs if the model didn't learn the concept but memorised it
- use train test split

Train_Test_Split option 1. Test_size : fraction of test set (typically .25 - .3) 2. Train_seze : train + test = 1 3. Random_State : random sampling

[11]:
```
# import library
from sklearn.cross_validation import train_test_split

# perform splitting
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2,␣
 ↪random_state = 5)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cross_validation.py:41:
DeprecationWarning: This module was deprecated in version 0.18 in favor of the
model_selection module into which all the refactored classes and functions are
moved. Also note that the interface of the new CV iterators are different from
that of this module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

1.5.1 Splitted Datasets

[12]:
```
pd.DataFrame(X_train)
```

[12]:

	0	1	2	3	4
0	0.0	1.0	0.0	30.000000	54000.000000
1	0.0	1.0	0.0	40.000000	63777.777778
2	1.0	0.0	0.0	48.000000	79000.000000
3	0.0	0.0	1.0	27.000000	48000.000000
4	1.0	0.0	0.0	44.000000	72000.000000
5	0.0	1.0	0.0	50.000000	83000.000000
6	0.0	0.0	1.0	38.777778	52000.000000
7	0.0	0.0	1.0	38.000000	61000.000000

[13]:
```
pd.DataFrame(X_test)
```

[13]:

	0	1	2	3	4
0	1.0	0.0	0.0	37.0	67000.0
1	1.0	0.0	0.0	35.0	58000.0

[14]:
```
pd.DataFrame(Y_train)
```

[14]:

	0
0	0
1	1
2	1
3	1
4	0
5	0
6	0
7	0

[15]:
```
pd.DataFrame(Y_test)
```

[15]:

	0
0	1
1	1

1.6 Feature Scaling

- attributes containing numerical data, it mey happen that two numeric attributes aren't not in same scale (eg. Age, Salary)
- ML Models are performs badly if scalling missmatch happens, as many of them uses Eucledian distance to minimise error.
- large scale variable may dominate the smaller scales, thus introducing bias
- there are two mechanism to scalling > 1. Standardisation $X_{stand} = \frac{x - mean(X)}{SD(X)} > 2$. Normalization $X_{norm} = \frac{x - min(X)}{max(x) - min(x)}$

[16]:
```
from sklearn.preprocessing import StandardScaler

sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
```

1.6.1 Facts

1. Feature scalling for Dummy variables : depends on scenario, for non-ordinal data Not needed
2. Feature scalling for Dependent Variables : Non needed for Classification but for Regression

[17]:
```
pd.DataFrame(X_train)
```

[17]:

	0	1	2	3	4
0	-0.577350	1.290994	-0.774597	-1.259796	-0.838900
1	-0.577350	1.290994	-0.774597	0.070194	-0.026540
2	1.732051	-0.774597	-0.774597	1.134186	1.238156

```
3 -0.577350 -0.774597  1.290994 -1.658793 -1.337393
4  1.732051 -0.774597 -0.774597  0.602190  0.656580
5 -0.577350  1.290994 -0.774597  1.400184  1.570485
6 -0.577350 -0.774597  1.290994 -0.092360 -1.005064
7 -0.577350 -0.774597  1.290994 -0.195804 -0.257324
```

[18]:
```
pd.DataFrame(X_test)
```

[18]:
```
          0         1         2         3         4
0  1.732051 -0.774597 -0.774597 -0.328803  0.241169
1  1.732051 -0.774597 -0.774597 -0.594801 -0.506571
```

[ ]:
```

```