Fundamentals of Data Mining
Data Science
Faculty of Computing
Sri Lanka Institute of Information Technology

Practical 09 - Data Preprocessing

Why Data Pre – Processing is important?

Data preprocessing is a data mining technique that involves transforming raw data into an understandable format. Real-world data is often incomplete, inconsistent, and or lacking in certain behaviors or trends, and is likely to contain many errors. Data preprocessing is a proven method of resolving such issues. Data preprocessing prepares raw data for further processing.

- **Incomplete**: lacking attribute values, lacking certain attributes of interest, or containing only aggregate data
- **Noisy**: containing errors or outliers
- **Inconsistent**: containing discrepancies in codes or names

Data Preprocessing Tasks

Data Cleaning	**-** Fill in missing values, smooth noisy data, identify or remove outliers and noisy data, and resolve inconsistencies
Data integration	- Integrate data with multiple databases or files.
Data transformation -	Data normalization and aggregation
Data reduction	**-** Data reduction in volume but produces the same or similar analytical results
Data discretization	- (for numerical data)

First Step – Import the dataset and libraries

Open the Jupyter Notebook and create a new Notebook. Name the Notebook as data_preprocessing, You need to import following libraries for this activity.

```
import pandas as pd
import numpy as np
```

pandas library provides a really fast and efficient way to manage and explore data and provides data analysis tools for the Python programming. Now lets import the dataset.
Download the “Titanic.csv” dataset from the courseweb,

```
Df_dataset = pd.read_csv(“titanic.csv”)
```

Df_dataset is a DataFrame where it is a two-dimensional data structure.

Second Step – Explore the dataset

We will have a look at on the dataset, what is the content of the file, what are the issues that we need to address before analyzing of data. Dataset contains following columns.

- Survived - Survived (1) or died (0)
- Pclass - Passenger's class (1 = 1st, 2 = 2nd, 3 = 3rd)
- Name - Passenger's name
- Sex - Passenger's sex
- Age - Passenger's age
- SibSp - Number of siblings/spouses aboard
- Parch - Number of parents/children aboard (Some children travelled only with a nanny, therefore parch=0 for them.)
- Ticket - Ticket number
- Ticket Booking Period
- Fare - Fare
- Cabin - Cabin
- Embarked - Port of embarkation (C = Cherbourg, Q = Queenstown, S = Southampton)

To view the first few rows,

```
Df_dataset.head()
```

To view the last few rows,

```
Df_dataset.tail()
```

Explore the data types of the columns using,

```
Df_dataset.dtypes

Df_dataset.describe()
```

If you want to explore particular column, you can use the following command.

```
Df_dataset['Embarked'].describe()
```

Third Step – Removing irrelevant columns

When we analyze the dataset, its clear that some columns are not relevant to our analysis, therefore we can remove them form our dataset. For that we can use, Df_dataset.drop() command. The list of unwanted columns can be input as parameters along with the "inplace = True" command, which means, we are nothing returned after the drop method. If "inplace=False"; a copy of the object with the operation performed is returned. The default for inplace is False.
Another parameter in drop method is "axis",

```
Df_dataset.drop(['Name','SibSp','Ticket'], axis=1, inplace=True)
```

Fourth Step – Handling missing data

Having large number of missing values is a huge effect for the analysis of data, for that we will remove the columns which have more than 30% of missing values. We are going to remove the empty rows also.

To identify the missing values we are using “isnull()” method in pandas.

```
col_num=0
TotalObjects =Df_dataset.shape[0]
print ("Column\t\t\t\t\t Null Values%")
for x in Df_dataset:
   nullCount =Df_dataset[x].isnull().sum();
   nullPercent = nullCount*100 / (TotalObjects)
   if nullCount > 0 and nullPercent > 30 :
      col_num=col_num+1
      Df_dataset.drop(x, axis=1,inplace=True)
      print(str(x)+"\t\t\t\t\t "+str(nullPercent))
print ("A total of "+str(col_num)+" deleted !")
```

Also you use the “info()” function to have an idea about null columns.

```
Df_dataset.info()
```

Another method to deal with missing values is, “fillna()”. This can be used to fill the NaN values by indicating the missing values in the column.

```
Df_dataset [‘Embarked'].fillna('Unknown',inplace = True)
Df_dataset[‘Age’].fillna(0, inplace=True)
```

There is another way to fill up missing values, where we can replace the mean value of the relevant column

```
Df_dataset[‘Age’].fillna(Df_dataset[‘Age’].mean(), inplace=True)
```

Finally, if you check the null values count in the Df_dataset, now you can see that, now all the columns are filled.

```
Df_dataset.isnull().sum()
```

Fifth Step – Formatting data

"Ticket Booking Period" column of the data frame contains the information about the ticket reservation time period, observe the content of the column.

Unique function in numpy would be helpful here. Try the code segment below:

pd.unique(Df_dataset['Ticket Booking Period'].values)

It returns the unique values in the column as an array.

Note that the content of the column includes text as well. For calculations we need the "Ticket Booking Period", in numerical format rather than in a textual format. Therefore, let's format the data in the column in a way that the "Ticket Booking Period" contains only numeric values.

Start by replacing the NaN values in the column with 0s. Then, use the following function in python to replace years with numeric values.
The rstrip() method returns a copy of the string with trailing characters removed (based on the string argument passed)

```
def CalculateTicketBookingPeriod(week):
    if week == '< 1 week':
        return 0.5
    elif week == '4+ weeks':
        return 4
    else:
        wk=str(week)
        return wk.rstrip('weeks')
```

Now we can apply the function to the "Ticket Booking Period" column to convert the weeks to numeric format.

```
Df_dataset['Ticket Booking Period']=Df_dataset['Ticket Booking Period'].apply(CalculateTicketBookingPeriod)
```

Sixth Step – Dealing with categorical Data

When dealing with large and real-world datasets, categorical data is almost inevitable. Categorical variables represent types of data which may be divided into groups. Examples of categorical variables are race, sex, age group, educational level etc. These variables often has letters or words as its values. Categorical variables need to be coded in to numbers. Having coded the categorical variable into numbers may just not be enough.

```
def getNumber(str):
    if str=="male":
        return 1
```

```
    else:
        return 2
Df_dataset["Gender"]= Df_dataset["Gender"].apply(getNumber)
```

We will do the same process for the “Embarked” column.

```
def getEmb(str):
    if str=="S":
        return 1
    elif str=='Q':
        return 2
    else:
        return 3
Df_dataset["Embarked"]=Df_dataset["Embarked"].apply(getEmb)
Df_dataset.head()
```

Save to a csv,

```
Df_dataset . to_csv(“cleaned_dataset.csv”,index=False)
```

Seventh Step – Visualization

Data visualization is the representation of data or information in a graph, chart, or other visual format. Looking at a summarized view of data in forms of plots and charts would be helpful to get insights of data before analysis

```
import matplotlib.pyplot as plt
from matplotlib import style

males = (Df_dataset['Gender'] == 1).sum()
#Summing up all the values of column gender with a
#condition for male and similary for females
females = (Df_dataset['Gender'] == 2).sum()
print(males)
print(females)
p = [males, females]
plt.pie(p,    #giving array
        labels = ['Male', 'Female'], #Correspndingly giving labels
        colors = ['green', 'yellow'],   # Corresponding colors
        explode = (0.15, 0),    #How much the gap should me there between the pies
        startangle = 0)  #what start angle should be given
plt.axis('equal')
plt.show()
```

For a more detailed visualization,

```
MaleS=Df_dataset[Df_dataset.Gender==1][Df_dataset.Survived==1].shape[0]
print(MaleS)
MaleN=Df_dataset[Df_dataset.Gender==1][Df_dataset.Survived==0].shape[0]
print(MaleN)
FemaleS=Df_dataset[Df_dataset.Gender==2][Df_dataset.Survived==1].shape[0]
print(FemaleS)
FemaleN=Df_dataset[Df_dataset.Gender==2][Df_dataset.Survived==0].shape[0]
print(FemaleN)



chart=[MaleS,MaleN,FemaleS,FemaleN]
colors=['lightskyblue','yellowgreen','Yellow','Orange']
labels=["Survived Male","Not Survived Male","Survived Female","Not Survived Female"]
explode=[0,0.05,0,0.1]
plt.pie(chart,labels=labels,colors=colors,explode=explode,startangle=100,counterclock=False,autopct="%.2
f%%")
plt.axis("equal")
plt.show()
```