Data Preprocessing in python

By Ashik-E-Rabbani

ID: 161-15-7093

In this worksheet I am using Titanic dataset of Kaggle for this experiment. Using this dataset, we need to predict whom were alive.

Step 1: Importing Packages

First step is usually setting up the environment through importing the libraries that will be needed in the execution of program. A library is essentially a collection of modules that can be called and used. A lot of the things in the programming world do not need to be written explicitly every time they are required.

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

Step 2: Import preprocessing

The sklearn.preprocessing package provides several common utility functions and transformer classes to change raw feature vectors into a representation that is more suitable for the downstream estimators.

```
from sklearn import preprocessing
import matplotlib.pyplot as plt
```

Step 3: Import the Dataset

Our titanic dataset comes in CSV format. We will need to locate the directory of the CSV file at first (it's more efficient to keep the dataset in the same directory as our

program) and read it using a method called read_csv which can be found in the library called pandas.

```
# get titanic & test csv files as a DataFrame

#developmental data (train)
titanic_df = pd.read_csv("train.csv")

#cross validation data (hold-out testing)
test_df    = pd.read_csv("test.csv")

# preview developmental data
titanic_df.head()
```

Checking first elements of the DataFrame with .head() method

Output:

Out[3]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

Checking last elements of the DataFrame with .tail() method

Output:

In [4]:
```
df.tail()
```

Out[4]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.00	NaN	S
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.00	B42	S
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.45	NaN	S
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.00	C148	C
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.75	NaN	Q

To sum up, these methods return the top and bottom of the **dataframe**. The default number of rows is set to 5. But, we can change it by writing number of rows that you want to see inside the parentheses.

Step 4: Checking for Missing Data in Dataset

Real-world data often has missing values. Data can have missing values for a number of reasons such as observations that were not recorded and data corruption. Here we can see that we have many categorical features and some numeric ones too. Before turning this dataset into vectors of numbers that our classification algorithms can use, we should deal with missing values. Let's check how many missing values has our dataset per feature.

```
In [5]: df.isnull().sum()
Out[5]: PassengerId      0
        Survived         0
        Pclass           0
        Name             0
        Sex              0
        Age            177
        SibSp            0
        Parch            0
        Ticket           0
        Fare             0
        Cabin          687
        Embarked         2
        dtype: int64
```

train dataset

```
In [7]: df2.isnull().sum()
Out[7]: PassengerId      0
        Pclass           0
        Name             0
        Sex              0
        Age             86
        SibSp            0
        Parch            0
        Ticket           0
        Fare             1
        Cabin          327
        Embarked         0
        dtype: int64
```

test dataset

We could delete the training samples that have NULL[NaN] values but in this case we don't have a huge dataset. First we are going to transform the Cabin feature into a Deck feature, each cabin starts with a letter that denotes the deck and we don't really need more information than that.

```
# make a list of all the posible Decks, the last element is used when no cabin co
de is present
cabin_list = ['A', 'B', 'C', 'D', 'E', 'F', 'T', 'G', 'Unknown']
#define a function that replaces the cabin code with the deck character
def search_substring(big_string, substring_list):
    for substring in substring_list:
        if substring in big_string:
            return substring
    return substring_list[-1]
```

We have a similar problem with the Name feature, we have too much information that is hard to encode and not useful. So we can take only the title of the name for each person, lets define a function for that.

```
# replace passenger's name with his/her title (Mr, Mrs, Miss, Master)
def get_title(string):
    import re
    regex = re.compile(r'Mr|Don|Major|Capt|Jonkheer|Rev|Col|Dr|Mrs|Countess|Dona|
Mme|Ms|Miss|Mlle|Master', re.IGNORECASE)
    results = regex.search(string)
    if results != None:
        return(results.group().lower())
    else:
        return(str(np.nan))
```

Step 5: Encoding categorical data

Sometimes our data is in qualitative form, that is we have texts as our data. We can find categories in text form.

```
# dictionary to map to generate the new feature vector
title_dictionary = {
    "capt":"Officer",
    "col":"Officer",
    "major":"Officer",
    "dr":"Officer",
    "jonkheer":"Royalty",
    "rev":"Officer",
    "countess":"Royalty",
    "dona":"Royalty",
    "lady":"Royalty",
    "don":"Royalty",
```

```
    "mr":"Mr",
    "mme":"Mrs",
    "ms":"Mrs",
    "mrs":"Mrs",
    "miss":"Miss",
    "mlle":"Miss",
    "master":"Master",
    "nan":"Mr"
}
```

Now that we have the functions we need, lets apply them and create the features Title and Deck.

```
df['Deck'] = df['Cabin'].map(lambda x: search_substring(str(x), cabin_list))
df2['Deck'] = df2['Cabin'].map(lambda x: search_substring(str(x), cabin_list))
# delete the Cabin feature
df.drop('Cabin', 1, inplace=True)
df2.drop('Cabin', 1, inplace=True)

df['Title'] = df['Name'].apply(get_title)
df2['Title'] = df2['Name'].apply(get_title)
df['Title'] = df['Title'].map(title_dictionary)
df2['Title'] = df2['Title'].map(title_dictionary)
# delete the Name feature
df.drop('Name', 1, inplace=True)
df2.drop('Name', 1, inplace=True)
```

Let's take a look at the results we got

```
In [16]: df.tail()
```

Out[16]:

	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Ticket	Fare	Embarked	Deck	Title
886	887	0	2	male	27.0	0	0	211536	13.00	S	Unknown	Officer
887	888	1	1	female	19.0	0	0	112053	30.00	S	B	Miss
888	889	0	3	female	NaN	1	2	W./C. 6607	23.45	S	Unknown	Miss
889	890	1	1	male	26.0	0	0	111369	30.00	C	C	Mr
890	891	0	3	male	32.0	0	0	370376	7.75	Q	Unknown	Mr

Now we will drop the Ticket feature that does not really give much insight.

```
      #dropping ticket column
Df.drop('Ticket', 1, inplace=True)
Df2.drop('Ticket', 1, inplace=True)
```

We can also filter the data more by the prediction percentage whether it is above acceptance or not.

Step 6: The Final – Feature scaling

After finishing the step 4 and 5 our dataset is clean and ready to use. To identify the best feature of a dataset we need to find the prediction output first, suppose in this dataset using death people name we can identify the alive people prediction. Also the people who Left from Cherbourg they didn't die, similarly the peoples from Queenstown and Southampton were more alive than others. So our feature can be this places. If the inputted people is from those places they have most of the chance to survive.

```
#test_set.fillna(0, inplace=True)
y = df['Survived'].values
X = df.drop(['Survived','PassengerId'], axis=1).values
X_test = df2.drop('PassengerId', axis=1).values
```

Out[25]:

	PassengerId	Survived	Pclass	Sex	Age
0	1	0	3	1	-0.587428
1	2	1	1	0	0.617618
2	3	1	3	0	-0.286167
3	4	1	1	0	0.391672
4	5	0	3	1	0.391672