01_preprocessing

December 21, 2019

1 Kaggle Titanic survival - data preprocessing

Can we predict which passengers would survive the sinking of the Titanic?

Orginal kaggle page:https://www.kaggle.com/c/titanic

Subsequent machine learning notebooks using Titanic survival also provide links to load preprocessed data directly, so this notebook is not strictly needed before using other notebooks, but processing data into a useable form is often a key stage of any machine learning project, and so all practitioners will want to get to grips with common methods.

This Nntebook introduces the following:

- Using Pandas to load and process data (though some familiarity with Pandas is assumed)
- Looking at data types
- Listing feature headings
- Showing data
- Showing a statistical summary of data
- Filling in (imputing) missing data
- Encoding non-numerical fields
- Removing unwanted columns
- Saving processed data

The data includes.

Variable	Definition
survival	Survival (0 = No, 1 = Yes)
pclass	Ticket class
sex	Sex
Age	Age in years
sibsp	# of siblings / spouses aboard the Titanic
parch	# of parents / children aboard the Titanic
ticket	Ticket number
fare	Passenger fare
cabin	Cabin number
embarked	Port of Embarkation(C=Cherbourg, Q=Queenstown, S=Southampton)

1.1 Load modules

[1]:
```
import pandas as pd
import numpy as np
```

2 Load data

Data should be in a sub folder named data.

It may be downloaded from:

https://gitlab.com/michaelallen1966/1908_coding_club_kaggle_titanic/tree/master/data

Usually the first thing we will do is split data in training and test (usually with randomisation first), and we hold back the test data until model building is complete. In the case of this kaggle data a separate test data set is supplied, so we do not need to hold back and of the data.

We will load the kaggle data and make a copy we will work on (so we can always refer back to the orginal data if we wish).

[2]:
```
download_required = True

if download_required:

    # Download processed data:
    address = 'https://raw.githubusercontent.com/MichaelAllen1966/' + \
                '1804_python_healthcare/master/titanic/data/train.csv'

    data = pd.read_csv(address)

    # Create a data subfolder if one does not already exist
    import os
    data_directory ='./data/'
    if not os.path.exists(data_directory):
        os.makedirs(data_directory)

    # Save data
    data.to_csv(data_directory+'train.csv')
```

[3]:
```
orginal_data = pd.read_csv('./data/train.csv')
data = orginal_data.copy()
```

Let’s have a look at some general information on the table.

[4]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
```

```
Data columns (total 14 columns):
Unnamed: 0      891 non-null int64
Unnamed: 0.1    891 non-null int64
PassengerId     891 non-null int64
Survived        891 non-null int64
Pclass          891 non-null int64
Name            891 non-null object
Sex             891 non-null object
Age             714 non-null float64
SibSp           891 non-null int64
Parch           891 non-null int64
Ticket          891 non-null object
Fare            891 non-null float64
Cabin           204 non-null object
Embarked        889 non-null object
dtypes: float64(2), int64(7), object(5)
memory usage: 97.6+ KB
```

At this point we can note we have 891 passengers, but that 'Age', 'Cabin' and 'Embarked' have some data missing.

Let's list the data fields:

[5]:
```
list(data)
```

[5]:
```
['Unnamed: 0',
 'Unnamed: 0.1',
 'PassengerId',
 'Survived',
 'Pclass',
 'Name',
 'Sex',
 'Age',
 'SibSp',
 'Parch',
 'Ticket',
 'Fare',
 'Cabin',
 'Embarked']
```

Let's look at the top of our data.

[6]:
```
data.head()
```

[6]:
```
   Unnamed: 0  Unnamed: 0.1  PassengerId  Survived  Pclass  \
0           0             0            1         0       3
1           1             1            2         1       1
2           2             2            3         1       3
3           3             3            4         1       1
```

```
4          4             4            5         0       3

                                              Name     Sex   Age  SibSp  \
0                            Braund, Mr. Owen Harris    male  22.0      1
1  Cumings, Mrs. John Bradley (Florence Briggs Th…  female  38.0      1
2                             Heikkinen, Miss. Laina  female  26.0      0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
4                           Allen, Mr. William Henry    male  35.0      0

   Parch            Ticket     Fare Cabin Embarked
0      0         A/5 21171   7.2500   NaN        S
1      0          PC 17599  71.2833   C85        C
2      0  STON/O2. 3101282   7.9250   NaN        S
3      0            113803  53.1000  C123        S
4      0            373450   8.0500   NaN        S
```

We can count the number of empty values. We can see that we will need to deal with 'age', 'cabin', and 'embarked'.

[7]:
```
data.isna().sum()
```

[7]:
```
Unnamed: 0       0
Unnamed: 0.1     0
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
```

2.1 Showing a summary of the data

We can use the pandas `describe()` method to show a summary of the data. Note that this only shows numercial data.

[8]:
```
data.describe()
```

[8]:
```
       Unnamed: 0  Unnamed: 0.1  PassengerId    Survived      Pclass  \
count  891.000000    891.000000   891.000000  891.000000  891.000000
mean   445.000000    445.000000   446.000000    0.383838    2.308642
```

```
std     257.353842    257.353842   257.353842    0.486592     0.836071
min       0.000000      0.000000     1.000000    0.000000     1.000000
25%     222.500000    222.500000   223.500000    0.000000     2.000000
50%     445.000000    445.000000   446.000000    0.000000     3.000000
75%     667.500000    667.500000   668.500000    1.000000     3.000000
max     890.000000    890.000000   891.000000    1.000000     3.000000

              Age       SibSp       Parch        Fare
count  714.000000  891.000000  891.000000  891.000000
mean    29.699118    0.523008    0.381594   32.204208
std     14.526497    1.102743    0.806057   49.693429
min      0.420000    0.000000    0.000000    0.000000
25%     20.125000    0.000000    0.000000    7.910400
50%     28.000000    0.000000    0.000000   14.454200
75%     38.000000    1.000000    0.000000   31.000000
max     80.000000    8.000000    6.000000  512.329200
```

Of most likely useful fields we are missing sex and whether a patiened embarker or not. So let's code those numerically.

2.2 Filling in (imputing) missing data

For numerical data we may commonly choose to impute mssing values with zero, mean or median. We will use the median for age.

We will also create a new column showing which values were imputed (this may be useful information in a machine learning model)

[9]:
```python
def impute_missing_with_median(_series):
    """
    Replace missing values in a Pandas series with median,
    Returns a comppleted series, and a series shwoing which values are imputed
    """
    # Copy the series to avoid change to the original series.
    series = _series.copy()
    median = series.median()
    missing = series.isna()
    series[missing] = median

    return series, missing
```

[10]:
```python
age, imputed = impute_missing_with_median(data['Age'])
data['Age'] = age
data['AgeImputed'] = imputed
```

We will impute missing embarked text with a 'missing' label

[11]:
```python
def impute_missing_with_missing_label(_series):
    """Replace missing values in a Pandas series with the text 'missing'"""
    # Copy the series to avoid change to the original series.
    series = _series.copy()
    missing = series.isna()
    series[missing] = 'missing'

    return series, missing
```

[12]:
```python
embarked, imputed = impute_missing_with_missing_label(data['Embarked'])
data['Embarked'] = embarked
data['EmbarkedImputed'] = imputed
```

3 Sorting out cabin data

Cabin data is messy! Some passesngers have more than one cabin (in which case we will split out the multiple cabins and just use the first one). Cabin numbers are a letter followed by a number. We will separate out the letter and the number.

[13]:
```python
# Get cabin data from dataframe
cabin = data['Cabin']

# Set up strings to add each passenger data to
CabinLetter = []
CabinLetterImputed = []
CabinNumber = []
CabinNumberImputed = []

# Convert all cabin data to string (empty cells are current stored as 'float')
cabin = cabin.astype(str)

# Iterate through rows
for index, value in cabin.items():
    # If cabin info is missing (string is 'nan' then add imputed data)
    if value == 'nan':
        CabinLetter.append('missing')
        CabinLetterImputed.append(True)
        CabinNumber.append(0)
        CabinNumberImputed.append(True)
    # Otherwise split string by spaces where there are multiple cabins
    else:
        # Split multiple cabins
        cabins = value.split(' ')
        # Take first cabin
        use_cabin = cabins[0]
```

```
        letter = use_cabin[0] # First letter
        CabinLetter.append(letter)
        CabinLetterImputed.append(False)
        if len(use_cabin) > 1:
            number = use_cabin[1:]
            CabinNumber.append(number)
            CabinNumberImputed.append(False)
        else:
            CabinNumber.append(0)
            CabinNumberImputed.append(True)

data['CabinLetter'] = CabinLetter
data['CabinLetterImputed'] = CabinLetterImputed
data['CabinNumber'] = CabinNumber
data['CabinNumberImputed'] = CabinNumberImputed

data.drop('Cabin', axis=1, inplace=True)
```

[14]:

```
data.head()
```

[14]:

```
   Unnamed: 0  Unnamed: 0.1  PassengerId  Survived  Pclass  \
0           0             0            1         0       3
1           1             1            2         1       1
2           2             2            3         1       3
3           3             3            4         1       1
4           4             4            5         0       3

                                                Name     Sex   Age  SibSp  \
0                            Braund, Mr. Owen Harris    male  22.0      1
1  Cumings, Mrs. John Bradley (Florence Briggs Th…  female  38.0      1
2                             Heikkinen, Miss. Laina  female  26.0      0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
4                           Allen, Mr. William Henry    male  35.0      0

   Parch            Ticket     Fare Embarked  AgeImputed  EmbarkedImputed  \
0      0         A/5 21171   7.2500        S       False            False
1      0          PC 17599  71.2833        C       False            False
2      0  STON/O2. 3101282   7.9250        S       False            False
3      0            113803  53.1000        S       False            False
4      0            373450   8.0500        S       False            False

  CabinLetter  CabinLetterImputed CabinNumber  CabinNumberImputed
0     missing                True           0                True
1           C               False          85               False
2     missing                True           0                True
3           C               False         123               False
```

```
4     missing                True           0                True
```

Let's check our missing numbers totals again

[15]:
```
data.isna().sum()
```

[15]:
```
Unnamed: 0              0
Unnamed: 0.1            0
PassengerId             0
Survived                0
Pclass                  0
Name                    0
Sex                     0
Age                     0
SibSp                   0
Parch                   0
Ticket                  0
Fare                    0
Embarked                0
AgeImputed              0
EmbarkedImputed         0
CabinLetter             0
CabinLetterImputed      0
CabinNumber             0
CabinNumberImputed      0
dtype: int64
```

3.1 Encoding non-numerical fields.

There are three types of non-numerical field:

- Dichotomous, which have two, and only two, possibilities (e.g. male/female, alive/dead). These may be recoded as 0 or 1.
- Categorical, which have any number of possibilties that cannot be ordered in any sensible way (e.g. colour of car'). Each possibility is coded seperately as 0/1 (e.g red = 0 or 1, green = 0 or 1, blue = 0 or 1). This is called 'one-hot encoding' as there will be one '1' (hot) in a set of columns (with all other values being zero).
- Ordinal, which have any number of possibilties but which may be ordered in a sensible way and coded by order of list. For example the zise of shirts may be xs, s, m, l and xl. These may be re-coded as size 0, 1, 2, 3, 4 (or scalled in another way if appropriate).

We'll look at sex first. Let's pull that out as a separate 'series'

[16]:
```
sex = data['Sex']
sex.head()
```

[16]:
```
0      male
1    female
2    female
3    female
4      male
Name: Sex, dtype: object
```

From looking at the data it appears passengers are either male or female, but data can contain missing values or spelling mistakes, so let's check all the values present. An easy way to do this is to use Python's `set` command which only allows one instance of each value.

[17]:
```
set(sex)
```

[17]:
```
{'female', 'male'}
```

That's good. We have just 'demale' and 'male'. Let's code a new 'male' column manually, and check the mean (the proportion of passengers who are male).

[18]:
```
male = data['Sex'] == 'male'
male.mean()
```

[18]:
```
0.6475869809203143
```

That's looks reasonable. We'll add our new column to our dataframe, and remove the old 'sex' column.

To remove a column we use the pandas `drop()` method. To show it is a column we specigy `axis=1`. To instruct removal from the data itself we use `inplace=True`. This is the equivalent of saying `data = data.drop()`.

[19]:
```
data['male'] = male
data.drop(['Sex'], axis=1, inplace=True)
```

Let's look at our table now.

[20]:
```
data.head()
```

[20]:
```
   Unnamed: 0  Unnamed: 0.1  PassengerId  Survived  Pclass  \
0           0             0            1         0       3
1           1             1            2         1       1
2           2             2            3         1       3
3           3             3            4         1       1
4           4             4            5         0       3

                                                Name   Age  SibSp  Parch  \
0                            Braund, Mr. Owen Harris  22.0      1      0
1  Cumings, Mrs. John Bradley (Florence Briggs Th…  38.0      1      0
2                             Heikkinen, Miss. Laina  26.0      0      0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  35.0      1      0
```

```
4                            Allen, Mr. William Henry  35.0      0      0

             Ticket      Fare Embarked  AgeImputed  EmbarkedImputed  \
0         A/5 21171    7.2500        S       False            False
1          PC 17599   71.2833        C       False            False
2  STON/O2. 3101282    7.9250        S       False            False
3            113803   53.1000        S       False            False
4            373450    8.0500        S       False            False

  CabinLetter  CabinLetterImputed CabinNumber  CabinNumberImputed   male
0     missing                True           0                True   True
1           C               False          85               False  False
2     missing                True           0                True  False
3           C               False         123               False  False
4     missing                True           0                True   True
```

Let's do the same with 'embarked'.

[21]:
```python
embarked = data['Embarked']
set(embarked)
```

[21]:
```
{'C', 'Q', 'S', 'missing'}
```

Ah, we have four possibilties!

We could frame this as a series of if/elif/esle statements. That is reasonable for a few possibilties, but what if have have many? We could write our own function to 'one-hot' encode this column, but pandas can already do this for us with the `get_dummies` method.

Note that we pass a couple of useful arguments: `prefix` allows us to add some text to each label, and `dummy_na=True` allows us to specifically code missing values (though we have already given them the label 'missing').

As ever, it is often useful to look at the help for these methods (`help(pd.get_dummies)`).

[22]:
```python
embarked_coded = pd.get_dummies(embarked, prefix='Embarked')
embarked_coded.head()
```

[22]:

	Embarked_C	Embarked_Q	Embarked_S	Embarked_missing
0	0	0	1	0
1	1	0	0	0
2	0	0	1	0
3	0	0	1	0
4	0	0	1	0

Nice! We'll add our new table to the data table and drop the original 'Embarked' column. Pandas `concat` method will join our dataframes.

Pandas has `concat`, `merge` and `join` methods for combining dataframes https://pandas.pydata.org/pandas-docs/stable/user_guide/merging.html

[23]:
```
data = pd.concat([data, embarked_coded], axis=1)
data.drop(['Embarked'], axis=1, inplace=True)
data.head()
```

[23]:
```
   Unnamed: 0  Unnamed: 0.1  PassengerId  Survived  Pclass  \
0           0             0            1         0       3
1           1             1            2         1       1
2           2             2            3         1       3
3           3             3            4         1       1
4           4             4            5         0       3

                                                Name   Age  SibSp  Parch  \
0                            Braund, Mr. Owen Harris  22.0      1      0
1  Cumings, Mrs. John Bradley (Florence Briggs Th…  38.0      1      0
2                             Heikkinen, Miss. Laina  26.0      0      0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  35.0      1      0
4                           Allen, Mr. William Henry  35.0      0      0

             Ticket  …  EmbarkedImputed  CabinLetter  CabinLetterImputed  \
0         A/5 21171  …            False      missing                True
1          PC 17599  …            False            C               False
2  STON/O2. 3101282  …            False      missing                True
3            113803  …            False            C               False
4            373450  …            False      missing                True

  CabinNumber  CabinNumberImputed   male  Embarked_C  Embarked_Q  Embarked_S  \
0           0                True   True           0           0           1
1          85               False  False           1           0           0
2           0                True  False           0           0           1
3         123               False  False           0           0           1
4           0                True   True           0           0           1

   Embarked_missing
0                 0
1                 0
2                 0
3                 0
4                 0

[5 rows x 22 columns]
```

[24]:
```
cabin_coded = pd.get_dummies(CabinLetter, prefix='CabinLetter')
cabin_coded.head()
```

[24]:
```
   CabinLetter_A  CabinLetter_B  CabinLetter_C  CabinLetter_D  CabinLetter_E  \
0              0              0              0              0              0
1              0              0              1              0              0
```

```
2             0             0             0             0             0
3             0             0             1             0             0
4             0             0             0             0             0

   CabinLetter_F  CabinLetter_G  CabinLetter_T  CabinLetter_missing
0              0              0              0                    1
1              0              0              0                    0
2              0              0              0                    1
3              0              0              0                    0
4              0              0              0                    1
```

Now let's add those back to the table

[25]:
```
data = pd.concat([data, cabin_coded], axis=1)
data.drop(['CabinLetter'], axis=1, inplace=True)
```

[26]:
```
data.head()
```

[26]:
```
   Unnamed: 0  Unnamed: 0.1  PassengerId  Survived  Pclass  \
0           0             0            1         0       3
1           1             1            2         1       1
2           2             2            3         1       3
3           3             3            4         1       1
4           4             4            5         0       3

                                                Name   Age  SibSp  Parch  \
0                            Braund, Mr. Owen Harris  22.0      1      0
1  Cumings, Mrs. John Bradley (Florence Briggs Th…  38.0      1      0
2                             Heikkinen, Miss. Laina  26.0      0      0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  35.0      1      0
4                           Allen, Mr. William Henry  35.0      0      0

             Ticket  …  Embarked_missing  CabinLetter_A  CabinLetter_B  \
0         A/5 21171  …                 0              0              0
1          PC 17599  …                 0              0              0
2  STON/O2. 3101282  …                 0              0              0
3            113803  …                 0              0              0
4            373450  …                 0              0              0

   CabinLetter_C CabinLetter_D  CabinLetter_E  CabinLetter_F  CabinLetter_G  \
0              0             0              0              0              0
1              1             0              0              0              0
2              0             0              0              0              0
3              1             0              0              0              0
4              0             0              0              0              0

   CabinLetter_T  CabinLetter_missing
```

```
0                 0                        1
1                 0                        0
2                 0                        1
3                 0                        0
4                 0                        1

[5 rows x 30 columns]
```

Now we will drop the Name and Ticket column (they may perhaps be useful in some way, but we'll simplify things by remiving them)

3.2 Drop columns

[27]:
```
cols_to_drop = ['Name', 'Ticket']
data.drop(cols_to_drop, axis=1, inplace=True)
data.head()
```

[27]:
```
   Unnamed: 0  Unnamed: 0.1  PassengerId  Survived  Pclass   Age  SibSp  \
0           0             0            1         0       3  22.0      1
1           1             1            2         1       1  38.0      1
2           2             2            3         1       3  26.0      0
3           3             3            4         1       1  35.0      1
4           4             4            5         0       3  35.0      0

   Parch     Fare  AgeImputed  …  Embarked_missing  CabinLetter_A  \
0      0   7.2500       False  …                 0              0
1      0  71.2833       False  …                 0              0
2      0   7.9250       False  …                 0              0
3      0  53.1000       False  …                 0              0
4      0   8.0500       False  …                 0              0

  CabinLetter_B  CabinLetter_C  CabinLetter_D  CabinLetter_E  CabinLetter_F  \
0             0              0              0              0              0
1             0              1              0              0              0
2             0              0              0              0              0
3             0              1              0              0              0
4             0              0              0              0              0

   CabinLetter_G  CabinLetter_T  CabinLetter_missing
0              0              0                    1
1              0              0                    0
2              0              0                    1
3              0              0                    0
4              0              0                    1

[5 rows x 28 columns]
```

3.3 Having a quick look at differences between survived and non-survived passengers

Phew, the data-preprocessing is done! This is often a tedious and time-consuming stage with few 'endorphin rush' rewards to be had.

Let's split our data into survied and non-survived and have a quick look to see anything obvious.

[28]:
```
mask = data['Survived'] == 1 # mask for survived passengers
survived = data[mask]

# Invert mask (for passengers who died
mask = mask == False
died = data[mask]
```

Now let's have a quick look at mean values for our two groups. We'll put them side by side in a new dataframe

[29]:
```
summary = pd.DataFrame()
summary['survived'] = survived.mean()
summary['died'] = died.mean()
summary
```

[29]:

	survived	died
Unnamed: 0	443.368421	446.016393
Unnamed: 0.1	443.368421	446.016393
PassengerId	444.368421	447.016393
Survived	1.000000	0.000000
Pclass	1.950292	2.531876
Age	28.291433	30.028233
SibSp	0.473684	0.553734
Parch	0.464912	0.329690
Fare	48.395408	22.117887
AgeImputed	0.152047	0.227687
EmbarkedImputed	0.005848	0.000000
CabinLetterImputed	0.602339	0.876138
CabinNumberImputed	0.611111	0.885246
male	0.318713	0.852459
Embarked_C	0.271930	0.136612
Embarked_Q	0.087719	0.085610
Embarked_S	0.634503	0.777778
Embarked_missing	0.005848	0.000000
CabinLetter_A	0.020468	0.014572
CabinLetter_B	0.102339	0.021858
CabinLetter_C	0.102339	0.043716
CabinLetter_D	0.073099	0.014572
CabinLetter_E	0.070175	0.014572
CabinLetter_F	0.023392	0.009107

```
CabinLetter_G         0.005848   0.003643
CabinLetter_T         0.000000   0.001821
CabinLetter_missing   0.602339   0.876138
```

3.4 Save processed data

[30]:
```
data.to_csv('./data/processed_data.csv', index=False)
```