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

[ ]:
```
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).

[ ]:
```
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')
```

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

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

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

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:

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

Let's look at the top of our data.

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

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

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

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.

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

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)

[ ]:
```
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
```

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

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

[ ]:
```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
```

[ ]:
```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.

[ ]:
```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)
```

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

Let's check our missing numbers totals again

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

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'

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

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.

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

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).

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

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()`.

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

Let's look at our table now.

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

Let's do the same with 'embarked'.

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

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)`).

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

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

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

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

Now let's add those back to the table

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

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

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

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

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.

[ ]:
```
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

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

3.4 Save processed data

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