Machine Intelligence

End-to-End Machine Learning Project Part 3

Prepare the Data for Machine Learning Algorithms

Get latest Python script from Blackboard

- The latest version of ‘chapter 2 load data v4.py’ is in the Content section of Brightspace under today’s lecture slides.

- Be sure to go ahead and get that one ready to go in PyCharm/VSCode

A note about calling main()

Some of you are doing this to call main

```
if __name__ == '__main__':
    main()
```

This is not necessary. Just call main() directly with no indentation

What we’ve covered so far

Part 1: How to get a quick view of your dataset using:

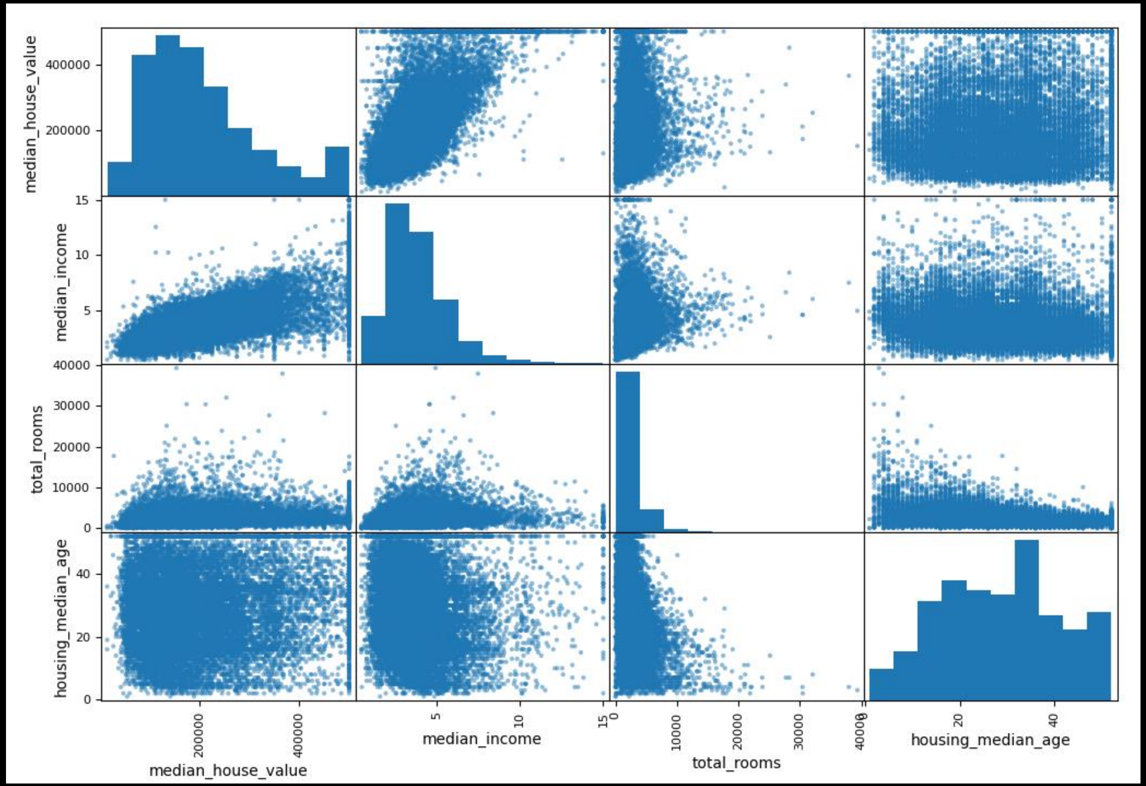
head() – first 5 rows of data
info() – quick description of the data
describe() – summary of the numerical attributes
hist() – histograms of the numerical attributes

Part 2: How to create a reliable training set and test set from the data using StratifiedShuffleSplit class

Part 2b: How to create new attributes that seem more useful (like rooms per household, etc., instead of total rooms in a district)

Summary from last time

- We learned about ***linear correlations*** between pairs of data
- We used Panda's ***scatter_matrix function*** to produce plots of correlations and got the results below for some of our housing data



Experimenting with Attribute Combinations

- ***Total number of rooms in a district*** is not very useful if you don't know how many households there are
  - Better to have the number of rooms per household
- Similarly, the ***total number of bedrooms*** by itself is not very useful – comparing it to the ***total number of rooms*** might be better
- Also, ***population per household*** seems like an interesting attribute combination to look at
- Let's create them – run chapter 2 load data v4
  - Look at the correlation matrix for our attributes

Experimenting with Attribute Combinations

- What did we learn?
  - ***bedrooms_per_room*** is much more correlated with median house value than the total number of rooms or bedrooms
  - Apparently, houses with a lower bedroom/room ratio tend to be more expensive – they have more rooms that are not bedrooms
  - The number of ***rooms per household*** is a little more informative than the ***total_rooms*** in a district
    - The larger the house the more expensive it is
- This round of exploration does not have to be thorough
- The point is to start off on the right foot and gain some quick insights to help get a good first prototype – but ***this is an iterative process***

Next step: Preparing Data for ML Algorithms

- Why is data preparation so important for ML algorithms?

- https://machinelearningmastery.com/data-preparation-is-important/

Next step: Preparing Data for ML Algorithms

- We will ***write functions to do this***! Why?
- This will ***allow us to reproduce these transformations easily on any dataset*** (e.g., the next time you get a fresh dataset)
- You will ***gradually build a library of transformation functions*** that you can reuse in future projects
- You ***can use these functions in your live system*** to transform the next data before feeding it to your algorithms
- This will ***make it possible for you to easily try various transformations*** and see which combination of transformations work best

Next step: Preparing Data for ML Algorithms

First, let's ***revert to a clean training set*** (by copying strat_train_set once again)

Then, let's ***separate the predictors and the labels*** since we don't necessarily want to apply the same transformations to the predictors and the target values

- Note that ***drop() creates a copy of the data*** and does not affect strat_train_set
- Look at the code on lines 102 to 106. Remove the comments on 101 and 107.
- Go ahead and run the program
- You can see that housing doesn't have median house value any more
- We created a variable ***housing_labels*** that has median house value

Cleaning the data

- Most ML algorithms cannot work with missing features – let's see if we have any missing values
  - Line 105 prints out the number of values for each attribute. ***total_bedrooms*** has some missing values

- Let's create a few functions to take care of any we might have – like with total_bedrooms

- We have 3 options
  - ***Option 1***: Get rid of the corresponding districts
  - ***Option 2***: Get rid of the whole attribute
  - ***Option 3***: Set the missing values to some value (zero, the mean, the median, etc.)

- We can easily do this with ***DataFrame's dropna(), drop(), and fillna() methods***
  - We will try the 3 options one at a time,   but comment out lines 102-106

Cleaning the data

- The code we will play with next is on lines 108 to 118. Comments are there to indicate which of the 3 options is used. We will uncomment and comment each of the options one at a time
  - Option 1: lines 110 and 111 – Notice all our data has 16,354 rows instead of 16,512

  https://www.w3schools.com/python/pandas/ref_df_dropna.asp
  - Option 2: lines 113 and 114 – total_bedrooms no longer exists

  https://www.w3schools.com/python/pandas/ref_df_drop.asp#:~:text=The%20drop()%20method%20removes,method%20removes%20the%20specified%20row.
  - Option 3: lines 116 – 118 – total_bedrooms now has 16,512 rows – we had Python compute the median of all the values and fill that in for the missing values (Notice bedrooms_per_room still has missing values)

  https://www.w3schools.com/python/pandas/ref_df_fillna.asp

Cleaning the data with SimpleImputer

- Scikit-Learn provides a handy class to take care of missing values: ***SimpleImputer*** ***(https://scikit-learn.org/stable/modules/generated/sklearn.impute.SimpleImputer.html)***
  - We just ***create a SimpleImputer object***
  - A ***median value can only be computed on numerical attributes***, we need to create of copy of our data without the text attribute ***ocean_proximity***
  - Now we ***fit the imputer instance to the training data*** using the ***fit()*** method
  - The ***imputer computes the median of every attribute*** and stores the result in its ***statistics_ instance*** variable
  - ***Only total_bedrooms and bedrooms_per_room are missing values, but that may not be true in the future with other datasets***
  - Then we ***use the 'trained' imputer to transform the training set*** by replacing missing values by the learned medians
  - The ***result is a NumPy array***. We need to convert it to a DataFrame object

Cleaning the data with SimpleImputer

- Make sure the code for the 3 options from before is all commented out – lines 110 - 118
- Remove the triple quotes from line 120 and line 131 and run the program
- The output from lines 126 and 126 should be identical as they are two different ways to compute the median of each attribute
- The output from line 130 shows all attributes having 16,512 rows

Scikit-Learn's main design principles: Consistency

- All objects share a consistent and simple interface
  - ***Estimators***: Any object that can estimate some parameters based on a dataset is called an estimator (imputer is an estimator). The estimation is performed by the ***fit() method***
  - ***Transformers***: Some estimators (such as imputer) can also transform a dataset; these are called transformers. The transformation is performed by the ***transform() method***. All transformers also have a convenience method called ***fit_transform()*** which calls fit() and then transform()
  - ***Predictors***: Some estimators are capable of making predictions given a dataset. These have a ***predict()*** method. They also have a ***score()*** method that measures the quality of the predictions given a test set

Scikit-Learn’s main design principles

- ***Inspection***: All the estimator’s hyperparameters (numbers and strings) and learned parameters are public (https://en.wikipedia.org/wiki/Hyperparameter_(machine_learning))
- ***Nonproliferation of classes***: Datasets are represented as NumPy arrays or SciPy sparse matrices, instead of new classes
- ***Composition***: Existing building blocks are reused as much as possible
- ***Sensible defaults***: Scikit-Learn provides reasonable default values for most parameters, making it easy to create a baseline working system quickly

Handling Text and Categorical Attributes

- Earlier we ***left out the categorical attribute ocean_proximity*** because it is a text attribute so we cannot compute its median
- Most ML algorithms prefer to work with numbers. Let's convert these categories from text to number
  - We can use Scikit-Learn's ***OrdinalEncoder class***
  - Comment out lines 121 – 130 by putting the 3 single quote marks back on lines 120 and 131
  - Remove the triple quote marks on lines 132 and 139
- One issue with this method is that ML algorithms will assume that two nearby values are more similar than two distant values.
  - What is ***one-hot encoding?*** (https://machinelearningmastery.com/why-one-hot-encode-data-in-machine-learning/)

Custom Transformers

- Although Scikit-Learn provides many useful transformers, ***you will need to write your own for tasks such as custom cleanup operations or combining specific attributes***
  - You will want your transformer to work seamlessly with Scikit-Learn functionalities (such as pipelines – discussed in a bit)

- All you need is to ***create a class and implement 3 methods***:
  - ***fit()***: (returning self)
  - ***transform()***
  - ***fit_transform()***: which calls fit() then transform()
  - You get ***fit_transform() for free is you inherit from TransformerMixin*** as a base class
  - If you add ***BaseEstimator as an inherited class, you also get get_params() and set_params***. These are useful for automatic hyperparameter tuning

Feature Scaling

- A very important transformation to apply to data

- ML algorithms don't perform well when the input numerical attributes have very different scales
  - This is ***the case for the housing data***: the total number of rooms ranges from about 6 to 39,320, while the median income only ranges from 0 to 15

- There are two common ways to get all attributes to have the same scale: min-max scaling and standardization
  - Min-max scaling (or normalization) is simple: values are shifted and rescaled, so they range from 0 to 1. The ***MinMaxScaler transformer*** does this for us
  - ***Standardization subtracts the mean value*** (so the mean is always 0), etc. This does not guarantee values from 0 to 1. This algorithm is much less affected by outliers. We will use the ***transform StandardScaler.***

Transformation Pipelines

- As you can see, there can be many data transformation steps that need to be executed in the proper order
- Scikit-Learn provides the ***Pipeline*** class to help with such sequences of transformations
- The ***Pipeline constructor*** takes a list of name/estimator pairs defining a sequence of steps. ***All but the last estimator must be transformers*** (have a fit_transform() method)
- The ***output of each fit_transform() call is passed as input to the next call***. The last pair only calls the fit() method
  - Put triple quote marks on lines 132 and 139
  - Uncomment line 121 only. We still want to remove ocean_proximity.
  - Also, remove quote marks from lines 141 and 146

Column Transformer

- So far, we’ve handled the categorical columns and the numerical columns separately
- It would be nice to have a single transformer able to handle all columns, applying the appropriate transformation to each columns
- We have the ***ColumnTransformer*** class for this
  - Remove the triple quote marks from lines 148 and 156
  - Leave lines 142 to 143 uncommented. We need lines 142-143 for this part. Comment out lines 144 and 145.

Summary

- Data preparation is a key starting point before we can run ML algorithms

- We need to do things like take care of any missing values, convert categorical data to numeric, scale all our numbers to be in a similar range

- We do this with Sci-kit Learn classes like SimpleImputer, OrdinalEncoder or OneHotEncoder, StandardScaler, Pipeline and ColumnTransformer

What’s next?

- In our next class (next Wednesday – no class on Monday) we will actually start doing machine learning!

Lab 5 and assignment 4 are now posted