Lab 4 (Part I): Predicting median housing price of census blocks

In this lab, we will work with census data to develop machine learning models that can predict the median housing prices of different census blocks in California.

1. Download data: download "housing.csv" from UBLearns. Save it into a folder called "data". In this dataset, each row contains the information about a census block in California. There are a number of attributes (columns): longitude and latitude (which represent the center coordinates of these census blocks), housing_median_age, total_rooms, total_bedrooms, population, households, median_income, median_house_value (note: this is the value we are trying to predict), ocean_proximity (categorical data, i.e., the relation between this block and the ocean). Please open this csv file and explore it yourself to get a feeling about it. Note the median income attribute has been scaled and capped at 15 (actually 15.0001) for higher median incomes, and at 0.5 (actually 0.4999) for lower median incomes. The numbers represent roughly tens of thousands of dollars (e.g., 3 actually means about $30,000).

2. Formalize the problem: the goal of this lab is to predict the median housing price of each census block based on the *features* we have in the data (e.g., median income and population). As you probably have noticed, the median housing price to be predicted is a continuous numeric value. Thus, we can formalize it into a regression problem. This is also a supervised learning problem, because we do have the labels (median housing price) for the data records.

3. Select a performance measure: how are we going to measure the performance of a developed machine learning model? In this lab, we will use Root Mean Square Error (RMSE), which quantifies the differences between the predicted and the true values.

$$\mathrm{RMSE}(\mathbf{X}, h) = \sqrt{\frac{1}{m}\sum_{i=1}^{m}\left(h\left(\mathbf{x}^{(i)}\right) - y^{(i)}\right)^2}$$

4. Load and explore data using Python library *pandas*: Pandas is an open source data analysis library for Python programming language. Here, we will use Pandas to load and explore housing.csv data. Create a Jupyter notebook named "Lab4_FirstName_LastName". In a new cell, type in the following command:

   import pandas as pd

   housing_data = pd.read_csv("data/housing.csv")

   housing_data.head(10)

We can get more information about this data by the following code:

housing_data.info()

The info() method is useful to get a quick description of the data, in particular the total number of rows, and each attribute's type and number of non-null values. Note there are some missing values for the *total_bedrooms* attribute. All attributes are numerical, except the *ocean_proximity* field. Its type is "object", so it could hold any kind of Python object, but since we have explored this data from a CSV file, you know that it contains categorical data represented as texts. To see how many different categories are in this attribute, we can use the following code:

housing_data["ocean_proximity"].value_counts()

As we can see, there are 5 different categories for the *ocean_proximity* field. We can further explore this dataset via:

housing_data.describe()

which shows a summary of the numeric attributes, such as count, mean, min, and max. Another quick way to get a feel of the type of data you are dealing with is to plot a histogram for each numerical attribute.

%matplotlib inline # for displaying figures inside a jupyter notebook

import matplotlib.pyplot as plt

housing_data.hist(bins=50, figsize=(20,15))

From the histograms, we may notice that two attributes, housing median age and the median house value, are capped. Housing median age will not exceed 52 years while median house value will not exceed $500,001.

We can remove these data limitations to reduce their impact on the trained model

housing_data = housing_data[housing_data["housing_median_age"] != 52]

housing_data = housing_data[housing_data["median_house_value"] != 500001]

We can plot the histogram again:

housing_data.hist(bins=50, figsize=(20,15))

5. Divide data into training and testing: after getting some sense about the data, we need to divide the data into training and testing datasets. Open a new cell, and type in the following command:

import numpy as np

def split_train_test(data, test_ratio): # define a new function for splitting the data

```
    shuffled_indices = np.random.permutation(len(data))
    test_set_size = int(len(data) * test_ratio)
    test_indices = shuffled_indices[:test_set_size]
    train_indices = shuffled_indices[test_set_size:]
    return data.iloc[train_indices], data.iloc[test_indices]
```

We can then type the following code for generating training and testing data:

```
np.random.seed(42)
train_set, test_set = split_train_test(housing_data, 0.2)
print(len(train_set),"training records and", len(test_set), "testing records")
```

So far we have considered purely random sampling methods. This is generally fine if your dataset is large enough (especially relative to the number of attributes). However, we might run the risk of introducing a significant sampling bias. **Stratified sampling**: the population is divided into homogeneous subgroups called strata, and the right number of instances is sampled from each stratum to guarantee that the test set is representative of the overall population. Suppose we are informed that median income is an important attribute, and we want to ensure that our training and testing data keep the ratio of median income in the population. Thus, we first look into the histogram of median income:

```
housing_data["median_income"].hist()
```

As can be seen, most median income values are clustered around 2 to 5 (i.e., $20,000–$50,000), but some median incomes go far beyond 6 (i.e., $60,000). To do stratified sampling, we need to first divide the data into several groups based on the median income. As an example, we divide the median income by an interval of 1.5 (to limit the number of income categories), and round up using ceil (to have discrete categories), and then keeping only the categories lower than 5 and merging the other categories into category 5:

```
housing_data["income_cat"] = np.ceil(housing_data["median_income"]/1.5)
housing_data["income_cat"].where(housing_data["income_cat"]<5,5.0, inplace=True)
```

We can then look at the values in the new attribute “income_cat”

```
housing_data["income_cat"].value_counts()
housing_data["income_cat"].hist()
```

Now let’s perform stratified sampling based on the median income groups

```
from sklearn.model_selection import StratifiedShuffleSplit
```

split = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=42)

for train_index, test_index in split.split(housing_data, housing_data["income_cat"]):

strat_train_set = housing_data.iloc[train_index]

strat_test_set = housing_data.iloc[test_index]

We can check the percentages of different income categories in the obtained training and testing data:

strat_train_set["income_cat"].value_counts()/len(strat_train_set)

strat_test_set["income_cat"].value_counts()/ len(strat_test_set)

We can compare these percentages with those in the original data:

housing_data["income_cat"].value_counts()/len(housing_data)

If we do random sampling, the obtained training and testing data are not as similar to the whole data as the stratified result:

train_set, test_set = split_train_test(housing_data, 0.2)

test_set["income_cat"].value_counts()/len(test_set)

Now, we have obtained training and testing data via stratified sampling. We need to remove the attribute of *income_cat*.

for _set in (strat_train_set, strat_test_set):

_set.drop("income_cat",axis=1, inplace=True)

There will be a warning but we can ignore it.

6. Explore and visualize data. Let's do some exploratory analysis with the training data. Note, we are going to look at only the training data for now. The testing data will be put aside and will be used only in the evaluation stage.

Before we do any data exploration on the training data, we will make a copy of it (in case we changes the data during our exploration process):

Let's make a copy and rename the training and testing data:

explore_data = strat_train_set.copy()

explore_data.plot(kind="scatter", x="longitude", y="latitude")

Setting the alpha parameter to 0.1 makes it easier to visualize the places where there is a high density of data points

explore_data.plot(kind="scatter", x="longitude", y="latitude", alpha=0.1)

Now that's much better: you can clearly see the high-density areas, namely the Bay Area and around Los Angeles and San Diego, plus a long line of fairly high density in the Central Valley, in particular around Sacramento and Fresno.

We will further show populations and housing prices in this figure. We will use circles in different sizes to represent the district's population (option s), and different colors to represent housing prices (option c). We will use a predefined color map (option cmap) called 'jet', which ranges from blue (low values) to red (high prices)

explore_data.plot(kind="scatter", x="longitude", y="latitude", alpha= 0.4,
s=explore_data["population"]/100, label="population",
c=explore_data["median_house_value"], cmap=plt.get_cmap("jet"), colorbar = True,
figsize=(10,7), sharex=False)

Now, you have completed the Part I of Lab 4. Please save your Jupyter Notebook, and you will continue working on it in our next lab.