Machine Learning

|Train/Test Split

Prof. Dr. Mostafa Elhosseini

https://youtube.com/drmelhosseini

Agenda

- Train Test Split
-

Split datasets into multiple subsets

- **train_test_split()** is a convenient function provided by Scikit-Learn to divide datasets into multiple subsets for training and testing.
- Performs similar operations as custom-defined functions like **shuffle_and_split_data()**, but with added features for enhanced functionality and ease of use.
- Allows the setting of a random generator seed via the **random_state** parameter to ensure reproducibility of the train/test split.
- Capable of **splitting multiple datasets** with the same number of rows on identical indices, ensuring synchronized division of data and its corresponding labels or related datasets.

Split datasets into multiple subsets

- Particularly beneficial when working with **separate DataFrames** for features and labels, ensuring that corresponding entries are correctly matched in the train and test sets.

```
from sklearn.model_selection import train_test_split

train_set, test_set = train_test_split(housing, test_size=0.2, random_state=42)
```

Random Sample

- Each subject in the population has the **same chance** of being included in that sample — a slip of paper, placed in a hat, and then draws names — Simple Random Sample.
- Measurements may vary from person to person, and just as people vary, so do samples vary.
- Predictions will, therefore, be more accurate for larger samples.

Random Sample

- with random sampling, the amount of variability from sample to sample is actually quite predictable
- The margin of error is a measure of the expected variability from one random sample to the next random sample
- Approximate margin of error $= \frac{1}{\sqrt{n}} \times 100\ \%$

Random Sample

- The **margin of error** is a measure of the expected variability from one random sample to the next random sample
  - Example: A margin of error of plus or minus 3 percentage points means it is very likely that the population percentage is no more than 3% lower or 3% higher than the reported sample percentage.

Simple Random Sample

- Good way to sample
- Every member and set of members have an equal chance of being included in the sample — Technology, random number generators
- Example—A teachers puts students' names in a hat and chooses without looking to get a sample of students.
- Why it's good: Random samples are usually fairly representative since they don't favor certain members
- A simple random sample is often just called a random sample. The “simple” adjective distinguishes this type of sampling from more complex random sampling designs

Simple Random Sample

- Good way to sample
- Every member and set of members have an equal chance of being included in the sample — Technology, random number generators
- Example—A teachers puts students' names in a hat and chooses without looking to get a sample of students.
- Why it's good: Random samples are usually fairly representative since they don't favor certain members
- A simple random sample is often just called a random sample. The “simple” adjective distinguishes this type of sampling from more complex random sampling designs

Simple Random Sample

- To select a simple random sample:
  - Number the subjects in the sampling frame.
  - Generate a set of those numbers randomly.
  - Sample the subjects whose numbers were generated.
- In practice, technology is used to generate the sample. This could be an app or a statistical program.

Stratified Random Sampling

- A statistical sampling technique used to ensure representation of identifiable subgroups or **strata** within a population in the sample.
  - Divide the entire population into distinct subgroups (strata) based on specific characteristics.
  - From each stratum, select members randomly.
  - Ensure the number of sampled units from each stratum is proportional to the stratum's size in the entire population.
  - Combine the samples from all strata to form a complete sample

Stratified Random Sampling

- Guarantees that all key subgroups are represented in the sample, preventing over- or under-representation.
- By ensuring each stratum is adequately represented, it reduces sampling bias, leading to more accurate results.

Example

- When conducting surveys, companies aim to select participants who **reflect the broader population** to ensure representativeness.
- Instead of random selection from a directory, surveyors **use criteria relevant** to their research questions to form their sample.
- For example, in the U.S., where the gender distribution is approximately **51.1% females and 48.9% males**, a survey aiming for representativeness would mirror this ratio, aiming for a sample of **511 females and 489 males**.
- This approach is especially important when the survey outcomes could be influenced by demographic factors like gender.

Example

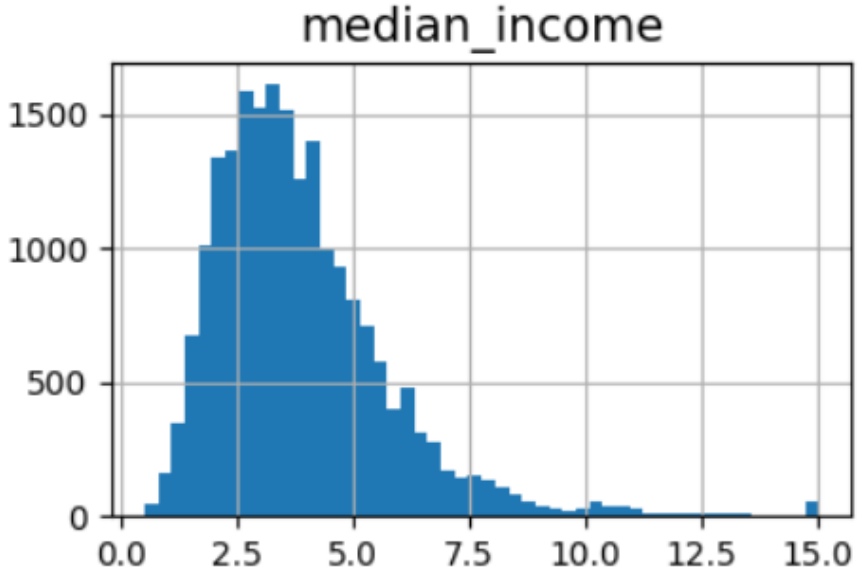
- If the people running the survey used **purely random sampling**, there would be about a **10.7% chance of sampling a skewed test** set with less than 48.5% female or more than 53.5% female participants. Either way, the survey results would likely be quite biased.
- Suppose we have a population with a known proportion of a certain attribute (in this case, the attribute is being female, with a population proportion of 51.1%).
- We want to find out the likelihood that a random sample of a specific size (1,000 individuals) deviates significantly from this known proportion, either by having too few (less than 48.5%) or too many (more than 53.5%) individuals with this attribute.

Example

- To solve this problem, we can employ the **binomial distribution**, which is suitable for scenarios where we have a fixed number of independent trials (in this case, selecting 1,000 individuals), each with two possible outcomes (female or not female), and a constant probability of success (being female) in each trial (51.1%).
- The **cumulative distribution function** (cdf) of the binomial distribution can be used to calculate the probability that the number of successes (females, in this context) in our sample is less than or equal to a certain value

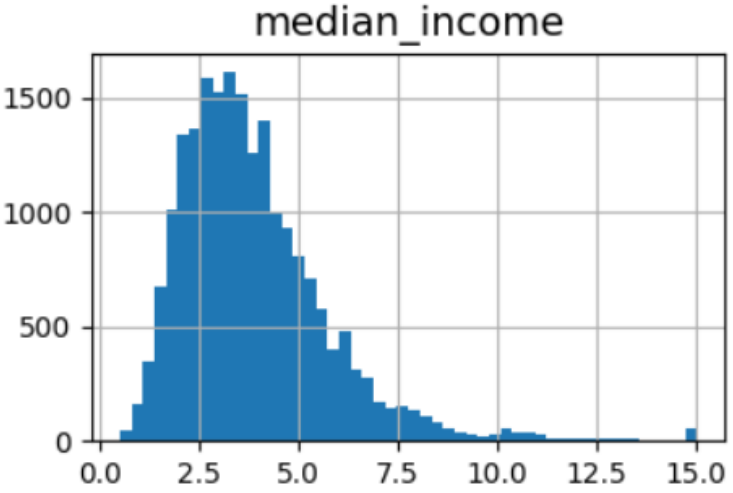
Continuous to Categorical

- Median income is a very important attribute to predict median housing prices
- You may want to ensure that the **test set** is representative of the various **categories of incomes** in the whole dataset.
- Since the median income is a continuous numerical attribute, you first need to create an income category attribute.



Continuous to Categorical

- Most median income values are clustered around 1.5 to 6 (i.e., $15,000–$60,000), but some median incomes go far beyond 6.
- It is important to have a sufficient number of instances in your dataset for each stratum, or else the estimate of a stratum's importance may be biased.
- This means that you should not have too many strata, and each stratum should be large enough.



Continuous to Categorical

- Uses the **pd.cut()** function to create an income category attribute with five categories (labeled from 1 to 5); category 1 ranges from 0 to 1.5 (i.e., less than $15,000), category 2 from 1.5 to 3, and so on.

```
housing["income_cat"] = pd.cut(housing["median_income"],
                               bins=[0., 1.5, 3.0, 4.5, 6., np.inf],
                               labels=[1, 2, 3, 4, 5])
```

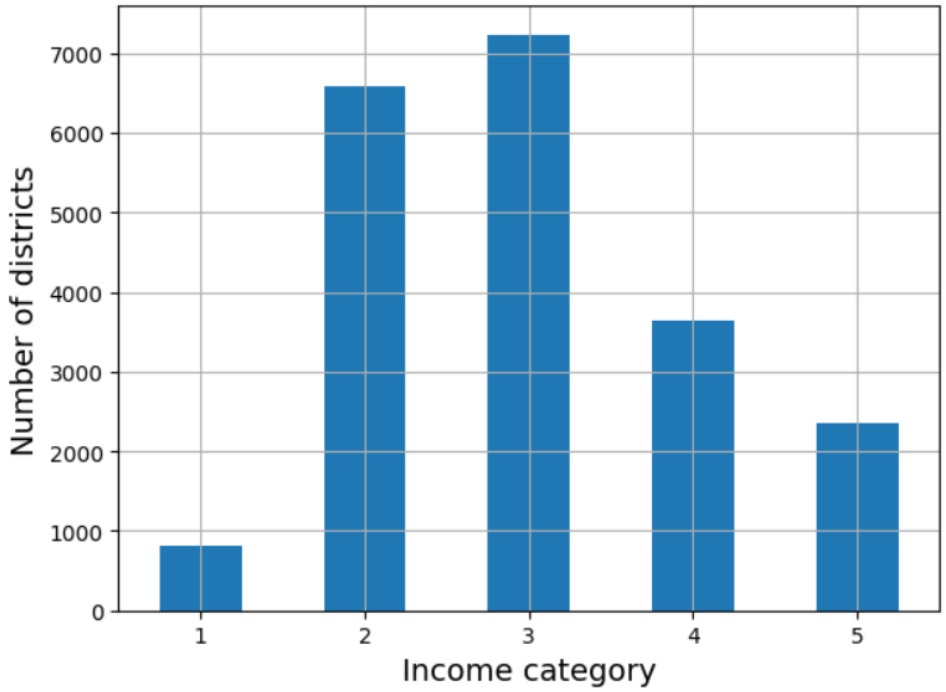
```
housing.head()
```

median_income	median_house_value	ocean_proximity	income_cat
8.3252	452600.0	NEAR BAY	5
8.3014	358500.0	NEAR BAY	5
7.2574	352100.0	NEAR BAY	5
5.6431	341300.0	NEAR BAY	4
3.8462	342200.0	NEAR BAY	3

Continuous to Categorical

- Uses the **pd.cut()** function to create an income category attribute with five categories (labeled from 1 to 5); category 1 ranges from 0 to 1.5 (i.e., less than $15,000), category 2 from 1.5 to 3, and so on.

```
housing["income_cat"].value_counts().sort_index().plot.bar(rot=0, grid=True)
plt.xlabel("Income category")
plt.ylabel("Number of districts")
save_fig("housing_income_cat_bar_plot")  # extra code
plt.show()
```



Continuous to Categorical

- **Do stratified sampling** based on the income category.
- the **split()** method yields the training and test indices, not the data itself
- Having **multiple splits** can be useful if you want to better estimate the performance of your model, as you will see when we discuss **crossvalidation.**

Continuous to Categorical

```
from sklearn.model_selection import StratifiedShuffleSplit

splitter = StratifiedShuffleSplit(n_splits=10, test_size=0.2, random_state=42)
strat_splits = []
for train_index, test_index in splitter.split(housing, housing["income_cat"]):
    strat_train_set_n = housing.iloc[train_index]
    strat_test_set_n = housing.iloc[test_index]
    strat_splits.append([strat_train_set_n, strat_test_set_n])
```

- The sum of the lengths of the training set and the testing set **in any single split** equals the total length of the main dataset.

- The above code generates 10 different stratified splits of the same dataset
- For now, you can just use the first split

```
strat_train_set, strat_test_set = strat_splits[0]
```

Continuous to Categorical

- There's a shorter way to get a **single split** using the train_test_split() function with the **stratify argument**

```
strat_train_set, strat_test_set = train_test_split(
    housing, test_size=0.2, stratify=housing["income_cat"], random_state=42)
```

- The income category proportions in the test set

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

```
3    0.350533
2    0.318798
4    0.176357
5    0.114341
1    0.039971
Name: income_cat, dtype: float64
```

Continuous to Categorical

- There's a shorter way to get a **single split** using the train_test_split() function with the **stratify argument**

```
strat_train_set, strat_test_set = train_test_split(
    housing, test_size=0.2, stratify=housing["income_cat"], random_state=42)
```

```
3    0.350533
2    0.318798
4    0.176357
5    0.114341
1    0.039971
Name: income_cat, dtype: float64
```

Income Category	Overall %	Stratified %	Random %	Strat. Error %	Rand. Error %
1	3.98	4.00	4.24	0.36	6.45
2	31.88	31.88	30.74	-0.02	-3.59
3	35.06	35.05	34.52	-0.01	-1.53
4	17.63	17.64	18.41	0.03	4.42
5	11.44	11.43	12.09	-0.08	5.63

Note

```
strat_train_set, strat_test_set = train_test_split(
    housing, test_size=0.2, stratify=housing["income_cat"], random_state=42)
```

```
for set_ in (strat_train_set, strat_test_set):
    set_.drop("income_cat", axis=1, inplace=True)
```

```
housing = strat_train_set.copy()
```

- Ready for the Next Lecture: Discover and Visualize the Data to Gain Insights