Machine Learning

|Creating Test set

Prof. Dr. Mostafa Elhosseini

https://youtube.com/drmelhosseini

Agenda

- Test set
-

Create a Test Set

Set aside part of the data early on:

- Before delving deep into the data analysis or choosing algorithms, it's crucial to reserve a portion of the data as a test set. This step should be done early in the process to **prevent any unintentional bias**.

Data Snooping

- Data snooping bias occurs when a **statistical model is influenced by examining the test set** and allowing this insight to inform the model selection, parameter tuning, or feature selection.
- This bias can result in an **overly optimistic estimation** of the model's performance because the model has been (consciously or unconsciously) tuned to perform well on the specific test dataset, rather than on new data.

Data Snooping

- **Repeated Use of Test Data**: If the test set is used repeatedly during the model building process, for example, to choose between different models or to finetune hyperparameters, there is a risk that the model will become **inadvertently optimized for the test set**.
- **Inadvertent Leak of Information**: Even if the test set is not directly used in the model training process, it can influence the choices made by the data scientist. For instance, if a **particular feature** in the test set is **noticed to be a strong predictor**, a data scientist might inadvertently give this feature more weight in the model.

Data Snooping

- **Multiple Comparisons**: Continuously evaluating a variety of models or hypotheses on the same test set increases the likelihood that you will find a model that performs exceptionally well on the test set due to chance rather than because it has learned generalizable patterns.

Data Snooping

To prevent data snooping bias, it's important to:

- **Preserve the Test Set**: Keep the test set isolated and use it only for the final evaluation of the model.
- **Use a Validation Set**: Employ a separate validation set (or cross-validation techniques) during the model selection and tuning phase.
- **Prevent Data Leakage**: Ensure that no information from the test set leaks into the training process.

Create a Test Set

- Creating a test set is theoretically simple; **pick some instances randomly**, typically 20% of the dataset (or less if your dataset is very large), and set them aside

```
import numpy as np

def shuffle_and_split_data(data, test_ratio):
    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]
```

```
train_set, test_set = shuffle_and_split_data(housing, 0.2)
len(train_set)
```

```
16512
```

```
len(test_set)
```

```
4128
```

Create a Test Set

- If the test set generation process is not controlled, there's a risk that over multiple runs, **all data points might end up being used as part of the test set at some point**. This exposure defeats the purpose of having a separate test set for unbiased evaluation.
- Running the program multiple times results in **different test sets** each time. This inconsistency can lead to **challenges in validating and reproducing results**, which is a cornerstone of scientific studies and machine learning projects.

Create a Test Set

- **Potential for data leakage**: When the model indirectly gets to "see" all the data over time, it can lead to data leakage, where information from the test set influences the training process, thus invalidating the performance metrics.
- **Avoiding overfitting**: Ensuring that the test set remains unseen and consistent helps in preventing overfitting, where the model is tuned too closely to the specific characteristics of the test data, harming its ability to generalize to new data.

Create a Test Set

- One solution is to **save the test set** on the first run and then load it in subsequent runs.
- Another option is to set the random number generator's seed (e.g., with np.random.**seed**(42)) before calling np.random.permutation() so that it **always generates the same shuffled indices**.

```
np.random.seed(42)
```

Create a Test Set

- However, both these solutions will break the next time you fetch an **updated dataset**.
- **When the dataset is updated** (e.g., new instances are added), the fixed seed will no longer generate a test set that includes a proportional representation of both old and new data.
- New instances might be excluded entirely or included disproportionately, leading to a test set that no longer accurately reflects the updated dataset's characteristics.

Create a Test Set

- To maintain a consistent division between training and testing data, use **unique identifiers** for each instance.
- Compute a **hash** for each instance's identifier and include the instance in the test set if the hash is within the top 20% of the hash value range.
- A hash function is an **algorithm** that takes an input (or 'message') and returns a fixed-size string of bytes. The output, known as the **hash value**, **digest**, or **hash code**, typically has a fixed length and is designed to **uniquely represent the input data**.

Create a Test Set

- This method ensures the test set remains unchanged over multiple executions, even when the dataset is updated.
- When the dataset is refreshed, the new test set will **include 20% of the newcomers while keeping prior training instances intact**.

Create a Test Set – Colab session

- This method ensures the test set remains unchanged over multiple executions, even when the dataset is updated.
- When the dataset is refreshed, the new test set will **include 20% of the newcomers while keeping prior training instances intact**.

	ID	Feature1	Feature2
0	1	0.374540	0.020584
1	2	0.950714	0.969910
2	3	0.731994	0.832443
3	4	0.598658	0.212339
4	5	0.156019	0.181825
5	6	0.155995	0.183405
6	7	0.058084	0.304242
7	8	0.866176	0.524756
8	9	0.601115	0.431945
9	10	0.708073	0.291229

Initial Dataset:

	ID	Feature1	Feature2	is_test
0	1	0.374540	0.020584	False
1	2	0.950714	0.969910	False
2	3	0.731994	0.832443	False
3	4	0.598658	0.212339	True
4	5	0.156019	0.181825	True
5	6	0.155995	0.183405	False
6	7	0.058084	0.304242	False
7	8	0.866176	0.524756	False
8	9	0.601115	0.431945	False
9	10	0.708073	0.291229	False

	ID	Feature1	Feature2	is_test
0	1	0.374540	0.020584	False
1	2	0.950714	0.969910	False
2	3	0.731994	0.832443	False
3	4	0.598658	0.212339	True
4	5	0.156019	0.181825	True
5	6	0.155995	0.183405	False
6	7	0.058084	0.304242	False
7	8	0.866176	0.524756	False
8	9	0.601115	0.431945	False
9	10	0.708073	0.291229	False
10	11	0.611853	0.366362	NaN
11	12	0.139494	0.456070	NaN
12	13	0.292145	0.785176	NaN

Updated Dataset:

	ID	Feature1	Feature2	is_test
0	1	0.374540	0.020584	False
1	2	0.950714	0.969910	False
2	3	0.731994	0.832443	False
3	4	0.598658	0.212339	True
4	5	0.156019	0.181825	True
5	6	0.155995	0.183405	False
6	7	0.058084	0.304242	False
7	8	0.866176	0.524756	False
8	9	0.601115	0.431945	False
9	10	0.708073	0.291229	False
10	11	0.611853	0.366362	True
11	12	0.139494	0.456070	False
12	13	0.292145	0.785176	False

Create a Test Set

- Unfortunately, the housing dataset does not have an identifier column. The simplest solution is to use the row index as the ID:

```
housing_with_id = housing.reset_index()  # adds an `index` column
train_set, test_set = split_data_with_id_hash(housing_with_id, 0.2, "index")
```

- If you use the row index as a unique identifier, you need to make sure that new data gets appended to the end of the dataset and that no row ever gets deleted. If this is not possible, then you can try to use the **most stable features to build a unique identifier**

Create a Test Set

- District's latitude and longitude are guaranteed to be stable for a few million years

```
housing_with_id["id"] = housing["longitude"] * 1000 + housing["latitude"]
train_set, test_set = split_data_with_id_hash(housing_with_id, 0.2, "id")
```