Object-Oriented Programming

Applied Machine Learning in Engineering - Exercise 02

TU Berlin, Summer Term 2025

Prof. Dr.-Ing. Merten Stender – merten.stender@tu-berlin.de

This exercise introduces the concept of **one-hot encoding** for categorical data and teaches how to implement it using both **functional programming** and **object-oriented programming** (OOP) in Python.

Learning Objectives

- Understand one-hot encoding for nominal (categorical) data.
- Implement encoding and decoding using functional programming.
- Implement the same functionality using object-oriented programming.
- Validate your implementation and compare it with `scikit-learn`'s standard library.

The file `bearing_faults.csv`, available in ISIS, contains a list of recorded bearing failures[1]. It has one column (identifying the bearing fault, e.g., `inner_race_fault`, `outer_race_fault`, etc.) and many rows (each row corresponding to one bearing). To read the file using Numpy:
`data = np.genfromtxt('bearing_faults.csv', dtype='str', delimiter=",")`.

Task 1: One-Hot Encoding using Functional Programming

Implement one-hot encoding using a functional programming style. You may optionally use test-driven development. **Hint**: `np.argmax` returns the index of the maximum value in an array.

(a) Implement the following functions:

- `fit(data)` — Finds the unique labels in the dataset.
- `encode(data)` — Maps categorical values to one-hot arrays.
- `decode(one_hots)` — Maps one-hot arrays back to original labels.

(b) Use a dictionary to map class labels to indices and vice versa:

`class_map=dict()`

`class_map['class 0']=0`.

You can get the value `'class 0'` by calling `class_map[0]` (i.e. the key of the dictionary).

(c) Validate your implementation using the provided dataset.

[1] If you want to know more about bearing faults, search the web for *Bearing Failure: Causes and Cures* by Schaeffler.

Expected Output Format

For example, encoding the list `['A', 'B', 'A']` should yield:

```
[['1', '0'], ['0', '1'], ['1', '0']]
```

Task 2: One-Hot Encoding using Object-Oriented Programming

Now implement one-hot encoding using object-oriented programming. You may optionally use test-driven development.

(a) Implement a class `OneHotEncoder()` with the following methods:

- `fit(data)` — Learns category mappings.
- `encode(data)` — Encodes using the learned mappings.
- `decode(one_hots)` — Decodes one-hot vectors.

(b) Re-use the implementations from the first task.

(c) Introduce functional class attributes and choose whether these should be accessible by the user or hidden (using an underscore `self._my_hidden_attribute`).

(d) Validate your implementation using the same dataset.

(e) Validate your implementation against the scikit-learn implementation of the `sklearn.preprocessing.OneHotEncoder` (link):

```
from sklearn.preprocessing import OneHotEncoder

enc = OneHotEncoder(sparse_output=False)
encoded = enc.fit_transform(data.reshape(-1, 1))
```

(f) Reflect on the following:

- Are your encoded results identical to scikit-learn's?
- What differences do you notice in API behavior?
- How does scikit-learn handle unknown categories?