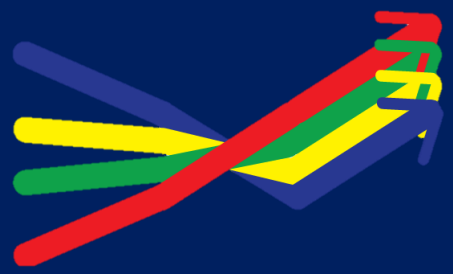
Object Oriented Programming for Machine Learning



Learn and transform, learn and predict

The different steps of the Machine Learning pipeline:

- Learn parameters from data
- Use those parameters to make transformations or predictions

Procedural Programming in ML

Code:

- Learn the parameters
- Make the transformations
- Make the predictions

Data:

- Store the parameters
- Mean values, regression coefficients, etc.

Procedural Programming: Hard-coded values

```
# encoding parameters
FREQUENT_LABELS = {
    'MSZoning': ['FV', 'RH', 'RL', 'RM'],
    'Neighborhood': ['Blmngtn', 'BrDale', 'BrkSide', 'ClearCr', 'CollgCr',
                     'Crawfor', 'Edwards', 'Gilbert', 'IDOTRR', 'MeadowV',
                     'Mitchel', 'NAmes', 'NWAmes', 'NoRidge', 'NridgHt',
                     'OldTown', 'SWISU', 'Sawyer', 'SawyerW', 'Somerst',
                     'StoneBr', 'Timber'],
    'RoofStyle': ['Gable', 'Hip'],
    'MasVnrType': ['BrkFace', 'None', 'Stone'],
    'BsmtQual': ['Ex', 'Fa', 'Gd', 'Missing', 'TA'],
    'BsmtExposure': ['Av', 'Gd', 'Missing', 'Mn', 'No'],
    'HeatingQC': ['Ex', 'Fa', 'Gd', 'TA'],
    'CentralAir': ['N', 'Y'],
    'KitchenQual': ['Ex', 'Fa', 'Gd', 'TA'],
    'FireplaceQu': ['Ex', 'Fa', 'Gd', 'Missing', 'Po', 'TA'],
    'GarageType': ['Attchd', 'Basment', 'BuiltIn', 'Detchd', 'Missing'],
    'GarageFinish': ['Fin', 'Missing', 'RFn', 'Unf'],
    'PavedDrive': ['N', 'P', 'Y']}


ENCODING_MAPPINGS = {'MSZoning': {'Rare': 0, 'RM': 1, 'RH': 2, 'RL': 3, 'FV': 4},
                     'Neighborhood': {'IDOTRR': 0, 'MeadowV': 1, 'BrDale': 2, 'Edwards': 3,
                                      'BrkSide': 4, 'OldTown': 5, 'Sawyer': 6, 'SWISU': 7, 'NAmes':
                                      'Mitchel': 9, 'SawyerW': 10, 'Rare': 11, 'NWAmes': 12, 'Gilber
                                      'Blmngtn': 14, 'CollgCr': 15, 'Crawfor': 16, 'ClearCr': 17, 'S
                                      'Timber': 19, 'StoneBr': 20, 'NridgHt': 21, 'NoRidge': 22},
                     'RoofStyle': {'Gable': 0, 'Rare': 1, 'Hip': 2},
                     'MasVnrType': {'None': 0, 'Rare': 1, 'BrkFace': 2, 'Stone': 3},
                     'BsmtQual': {'Missing': 0, 'Fa': 1, 'TA': 2, 'Gd': 3, 'Ex': 4},
                     'BsmtExposure': {'Missing': 0, 'No': 1, 'Mn': 2, 'Av': 3, 'Gd': 4},
                     'HeatingQC': {'Rare': 0, 'Fa': 1, 'TA': 2, 'Gd': 3, 'Ex': 4},
                     'CentralAir': {'N': 0, 'Y': 1},
                     'KitchenQual': {'Fa': 0, 'TA': 1, 'Gd': 2, 'Ex': 3},
                     'FireplaceQu': {'Po': 0, 'Missing': 1, 'Fa': 2, 'TA': 3, 'Gd': 4, 'Ex': 5},
                     'GarageType': {'Missing': 0, 'Rare': 1, 'Detchd': 2, 'Basment': 3, 'Attchd': 4,
                     'GarageFinish': {'Missing': 0, 'Unf': 1, 'RFn': 2, 'Fin': 3},
                     'PavedDrive': {'N': 0, 'P': 1, 'Y': 2}}


# ======= FEATURE GROUPS =============
```

Line 1, Column 1 master 9 Tab Size: 4

- Straightforward
- Hard-code parameters
- Save multiple objects or data structures

```
OUTPUT_SCALER_PATH = 'scaler.pkl'
OUTPUT_MODEL_PATH = 'lasso_regression.pkl'
```

Object Oriented Programming - OOP

In **Object-oriented programming** (**OOP**) we write code in the form of “objects”.

This “objects” can store **data** and can also store instructions or procedures (**code**) to modify that data, or do something else, like obtaining predictions.

- Data ⇒ attributes, properties
- Code or Instructions ⇒ methods (procedures)

OOP for Machine Learning

In **Object-oriented programming** (**OOP**) the “objects” can learn and store parameters

- Parameters get automatically refreshed every time model is re-trained
- No need of manual hard-coding

    - Methods:
        - Fit: learns parameters
        - Transform: transforms data with the learned parameters

    - Attributes: store the learn parameters

A Class

Creating a Class in Python

```
class MeanImputer:

    pass
```

A Class - __init__()

The properties or parameters that the class takes whenever it is initialized, are indicated in the **__init__()** method.

The first parameters will always be a variable called ***self***.

We can give any number of parameters to __init__()

```
class MeanImputer:


    def __init__(self, variables):

        self.variables = variables
```

A Class - __init__()

The properties or parameters that the class takes whenever it is initialized, are indicated in the **__init__()** method.

The first parameters will always be a variable called self.

We can give any number of parameters to __init__()

```
class MeanImputer:


    def __init__(self, variables):
        self.variables = variables
```

```
>> my_imputer = MeanImputer(
>>        variables = [‘age’, ‘fare’]
>> )


>> my_imputer.variables
[‘age’, ‘fare’]
```

A Class - methods

Methods are functions defined inside a class and can only be called from an instance of that class.

The first parameters will always be a variable called self.

Our fit() methods learns parameters

```
class MeanImputer:
    def __init__(self, variables):
        self.variables = variables

    def fit(self, X, y=None):
        self.imputer_dict_ =
        X[self.variables].mean().to_dict()

        return self
```

A Class - methods

Methods are functions defined inside a class and can only be called from an instance of that class.

The first parameters will always be a variable called self.

Our fit() methods learns parameters

```
>> my_imputer = MeanImputer(
>>        variables = [‘age’, ‘fare’]
>> )


>> my_imputer.fit(my_data)
>> my_imputer.imputer_dict_


{‘age’: 39, ‘fare’: 100}
```

A Class - methods

Methods are functions defined inside a class and can only be called from an instance of that class.

The first parameters will always be a variable called self.

Our fit() methods learns parameters

Our transform() method transforms data

```
class MeanImputer:
    def __init__(self, variables):
        self.variables = variables

    def fit(self, X, y=None):
        self.imputer_dict_ =
        X[self.variables].mean().to_dict()
        return self

    def transform(self, X):
        for x in self.variables:
            X[x] = X[x].fillna(
                self.imputer_dict_[x])
        return X
```