PIPELINES IN SCIKIT-LEARN

https://scikit-learn.org/stable/modules/compose.html#pipeline

- Pipelines in sklearn are sequences of estimators
  - An estimator in sklearn is either a transformer (pre-processing method) or a classifier/regressor (model training method)
  - Transformers: feature selection, scaler, imputer, category encoder, principal component analysis, …
    - They have two methods: .fit (train) and .transform (test)
  - Classifier / regressors: decision trees, knn, …
    - They have two methods: .fit (train) and .predict (test)

- A sequence of transformers IS a transformer
- A sequence of several transformers plus a classifier/regressor IS a classifier/regressor
- All estimators in a pipeline, except the last one, must be transformers

Doing the preprocessing correctly with scikit-learn PIPELINES

- *E.g. clf* is a pipeline: it is a training method with two steps: attribute selection and model training.
- Notice that which attributes should be selected is decided only during training (fit). For prediction, only the attributes selected during training are used.

```
from sklearn.pipeline import Pipeline

clf = Pipeline([
  ('feature_selection', SelectKBest(f_regression)),
  ('regression', tree.DecisionTreeRegressor())
])

clf.fit(X_train,y_train)
y_test_pred = clf.predict(X_test)
```

Doing the preprocessing correctly with and without PIPELINES

```
from sklearn.neighbors import KNeighborsRegressor
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

iris = load_iris()
X = iris.data
y = iris.target

X_train, X_test, y_train, y_test = train_test_split(X,y,
                                                    test_size=1/3,
                                                    random_state=42)
```

WITH PIPELINES

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler

# Define steps in the pipeline
knn = KNeighborsRegressor()
scaler = StandardScaler()
classif = Pipeline([
    ('standarization', scaler),
    ('knn', knn)
])

classif.fit(X_train,y_train)
y_hat = classif.predict(X_test)
```

WITHOUT PIPELINES (if outer is train/test)

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler

# Scale train and test
# Also: X_train = scaler.fit_transform(X_train)
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

classif = KNeighborsRegressor()

classif.fit(X_train,y_train)
y_hat = classif.predict(X_test)
```

Doing the preprocessing correctly with and without PIPELINES

```
from sklearn.neighbors import KNeighborsRegressor
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

iris = load_iris()
X = iris.data
y = iris.target

X_train, X_test, y_train, y_test = train_test_split(X,y,
                                                    test_size=1/3,
                                                    random_state=42)
```

WITHOUT PIPELINES

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler

# Scale train and test
# Also: X_train = scaler.fit_transform(X_train)
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

classif = KNeighborsRegressor()

classif.fit(X_train,y_train)
y_hat = classif.predict(X_test)
```

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler

# Scale train and test
# Also: X_train = scaler.fit_transform(X_train)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

classif = KNeighborsRegressor()

classif.fit(X_train,y_train)
y_hat = classif.predict(X_test)
```

PIPELINE FOR IMPUTATION AND FEATURE SCALING WITH KNN

```
from sklearn.datasets import load_boston
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.neighbors import KNeighborsRegressor
from sklearn.pipeline import Pipeline

m_boston = load_boston()
X = m_boston.data
y = m_boston.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)

scaler = StandardScaler()
imp = SimpleImputer(missing_values=np.nan, strategy='mean')
knn = KNeighborsRegressor()

regr = Pipeline([
    ('imp-mean', imp),
    ('scaling', scaler),
    ('knn', knn)
])

regr.fit(X_train, y_train)
prediction = regr.predict(X_test)
```

SECOND CORRECT WAY OF DOING IMPUTATION AND FEATURE SCALING WITH KNN. THIS TIME, WITHOUT PIPELINES

```
# Load Boston dataset
m_boston = load_boston()
X = m_boston.data
y = m_boston.target

# Split data into training and testing sets. This is the most important part:
# train/test split must be done before preprocessing.
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,
random_state=42)

# Imputation
imp = SimpleImputer(missing_values=np.nan, strategy='mean')
X_train_imp = imp.fit_transform(X_train)
X_test_imp = imp.transform(X_test)

# Scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train_imp)
X_test_scaled = scaler.transform(X_test_imp)

# Fit k-NN regressor on scaled, imputed training data
knn = KNeighborsRegressor()
knn.fit(X_train_scaled, y_train)

# Make predictions on scaled, imputed testing data
prediction = knn.predict(X_test_scaled)
```

PIPELINES IN SCIKIT-LEARN: hyper-parameter tuning

- Given that a pipeline, although it is a sequence of estimators, is itself an estimator, its hyper-parameters can be tuned by grid-search/random-search/bayesian optimization
- The hyper-parameters of a pipeline is the union of the hyper-parameters of the individual estimators (steps in the pipeline)

PIPELINES IN SCIKIT-LEARN: hyper-parameter tuning

- The names of the hyper parameters are:
  - stepname__hyperparameter
  - Feature selection + tree has two hyper-parameters:
    - **select__**k (how many of the top features to select)
    - **tree__**max_depth (max. depth of the tree)

```
from sklearn.pipeline import Pipeline
from sklearn.feature_selection import SelectKBest, f_regression
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import GridSearchCV
```

```
estimators = [('select', SelectKBest(f_regression)),
              ('tree', DecisionTreeRegressor())]
pipeline = Pipeline(estimators)
```

```
pipeline
```

```
# Names of hyper-parameters are
param_grid = {"select__k": [2, 5, 10],
              "tree__max_depth": [0.1, 10, 100]
```

```
pipeline_model = GridSearchCV(pipeline, param_grid=param_grid, cv=3)
pipeline_model.fit(X=X_train, y=y_train)
```

Transforming individual features

- Up to now, all pre-processing steps process all attributes in the dataset
- But in some cases, different attributes/features need to follow different pre-processing steps.
- For instance, categorical attributes should undergo some pre-processing and numerical attributes some other pre-processing.
- **ColumnTransformer** can be used for that

Transforming individual features

- Let's suppose that we start with the titanic dataset which is a **Pandas dataframe**

	Survived	Pclass	Sex	Age	Fare	Embarked
	0	3	male	22.0	7.2500	S
	1	1	female	38.0	71.2833	C
	1	3	female	26.0	7.9250	S
	1	1	female	35.0	53.1000	S
	0	3	male	35.0	8.0500	S

Transforming individual features

- Each attribute or each type of attribute (numeric, categorical, ...) can be transformed in a different way
  - https://scikit-learn.org/stable/modules/compose.html#pipeline

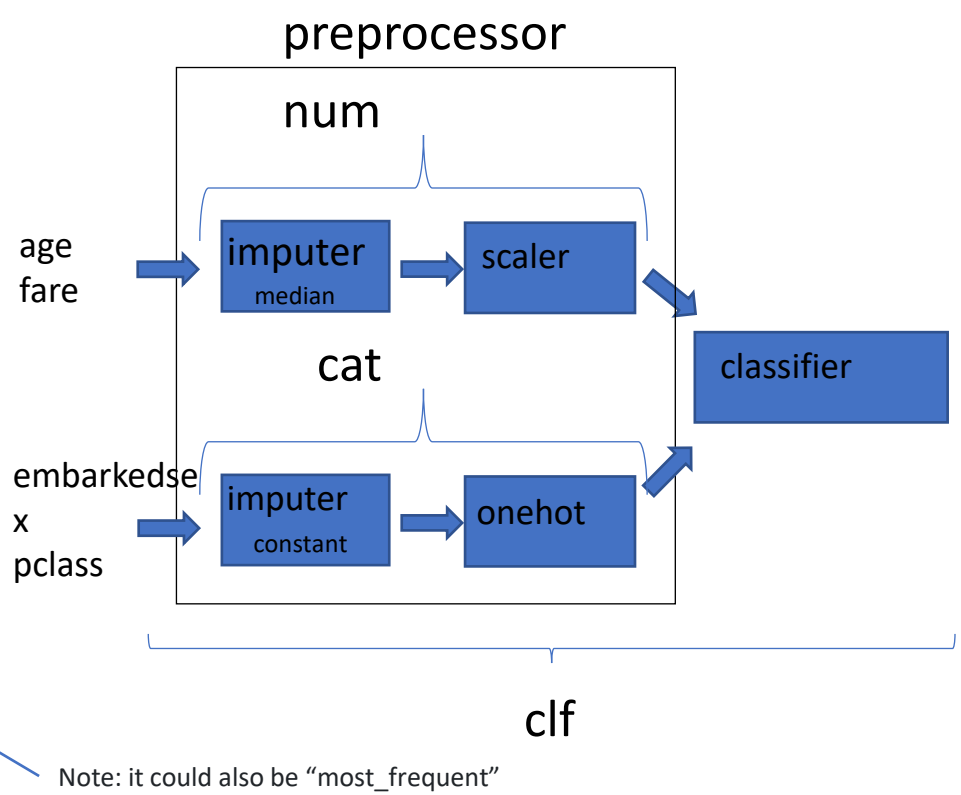
```
# Numeric Features:
# - age: float.
# - fare: float.
# Categorical Features:
# - embarked: categories encoded as strings {'C', 'S', 'Q'}.
# - sex: categories encoded as strings {'female', 'male'}.
# - pclass: ordinal integers {1, 2, 3}.

# We create the preprocessing pipelines for both numeric and categorical data.
numeric_features = ['age', 'fare']
numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler())])

categorical_features = ['embarked', 'sex', 'pclass']
categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='constant', fill_value='missing')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))])

preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('cat', categorical_transformer, categorical_features)])

# Append classifier to preprocessing pipeline.
# Now we have a full prediction pipeline.
clf = Pipeline(steps=[('preprocessor', preprocessor),
                      ('classifier', LogisticRegression(solver='lbfgs'))])
```



PIPELINE PERSISTENCE

- Trained / fit pipelines can be saved into a file in pickle format, to be used later
- Caution! if the version of sklearn changes, or a different architecture is used (e.g. saving in Windows10 and loading in Linux), this may lead to unexpected results

```
from joblib import dump, load
dump(pca_sel_knn, 'pca_sel_knn.joblib')
pca_sel_knn = load('pca_sel_knn.joblib')
```