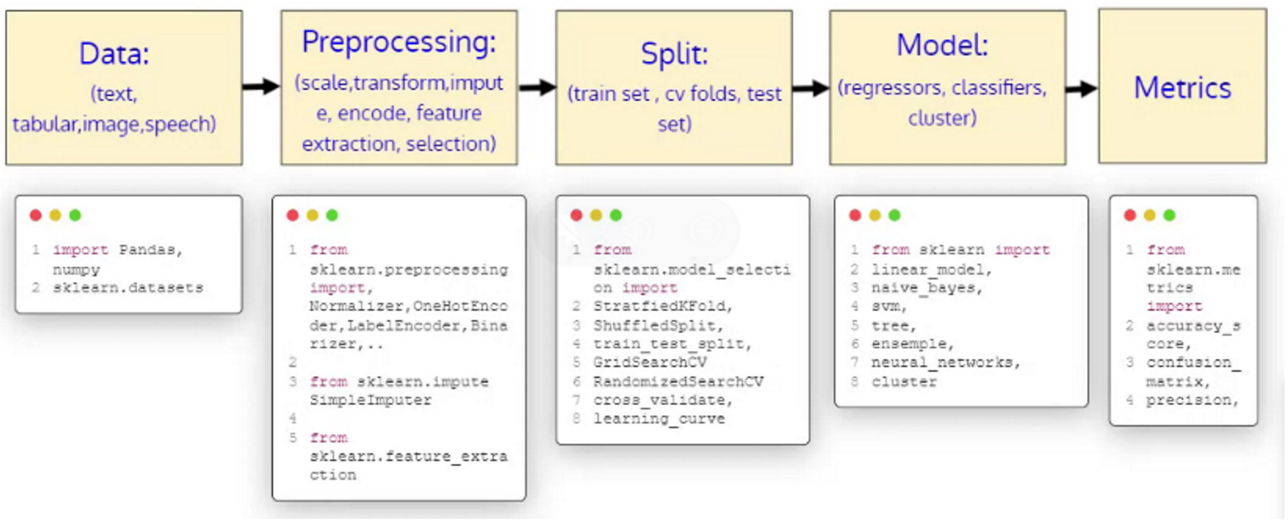
Sklearn API

28 January 2023 04:09 PM

1. ML Pipeline

   a.



Data:
(text, tabular,image,speech)
Preprocessing:
(scale,transform,impute, encode, feature extraction, selection)
Split:
(train set , cv folds, test set)
Model:
(regressors, classifiers, cluster)
Metrics

```
import Pandas, numpy
sklearn.datasets
```

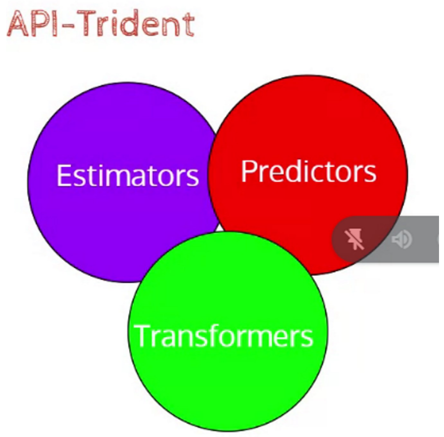
```
from sklearn.preprocessing import, Normalizer,OneHotEncoder,LabelEncoder,Binarizer,..

from sklearn.impute SimpleImputer

from sklearn.feature_extraction
```

```
from sklearn.model_selection import
StratfiedKFold,
ShuffledSplit,
train_test_split,
GridSearchCV
RandomizedSearchCV
cross_validate,
learning_curve
```

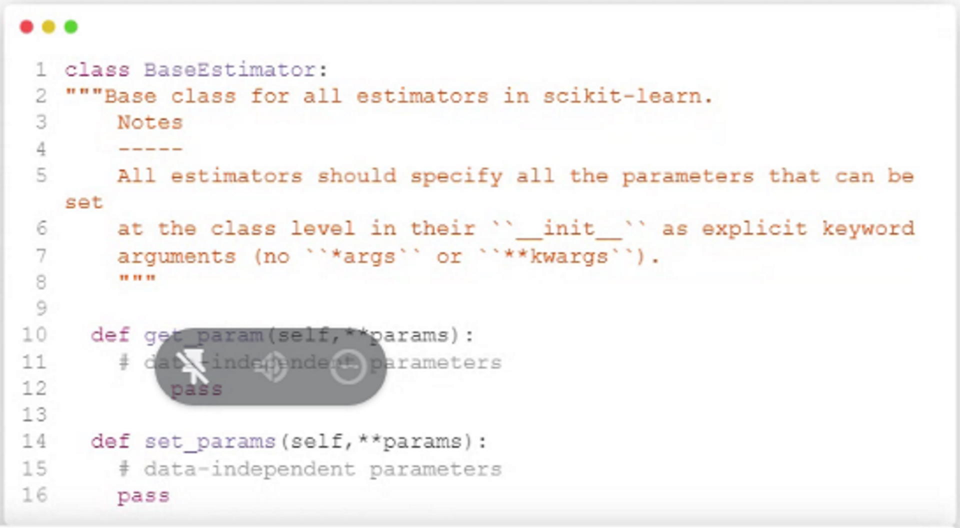
```
from sklearn import
linear_model,
naive_bayes,
svm,
tree,
ensemple,
neural_networks,
cluster
```

```
from sklearn.metrics import
accuracy_score,
confusion_matrix,
precision,
```

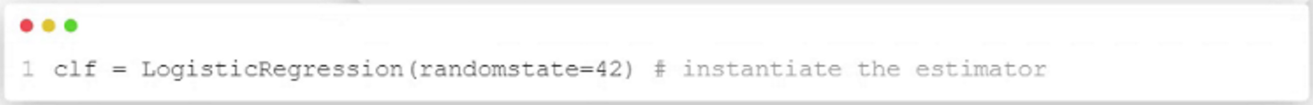
1. API Trident:
   a. Estimators- fit
   b. Predictors- predict or score
   c. Transformers -pipeline, imputer scaler etc



Trident

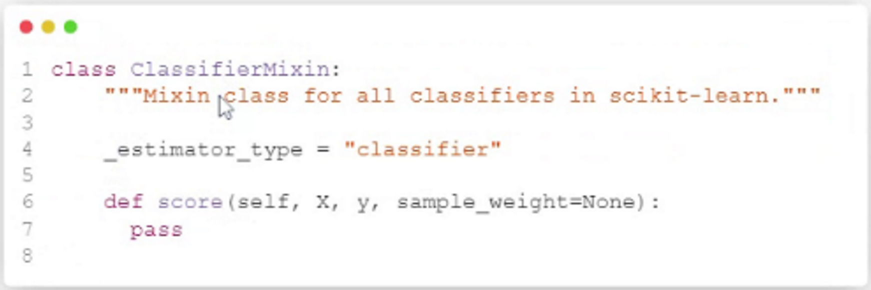


```
class BaseEstimator:
"""Base class for all estimators in scikit-learn.
    Notes
    -----
    All estimators should specify all the parameters that can be set
    at the class level in their ``__init__`` as explicit keyword
    arguments (no ``*args`` or ``**kwargs``).
    """

  def get_param(self,**params):
    # data-independent parameters
      pass

  def set_params(self,**params):
    # data-independent parameters
    pass
```

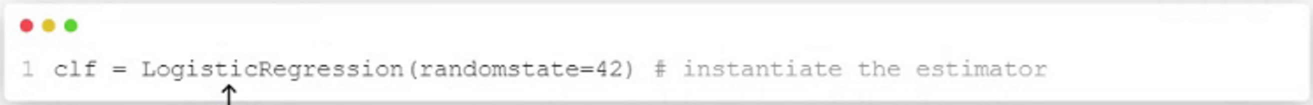


```
clf = LogisticRegression(randomstate=42) # instantiate the estimator
```

Trident



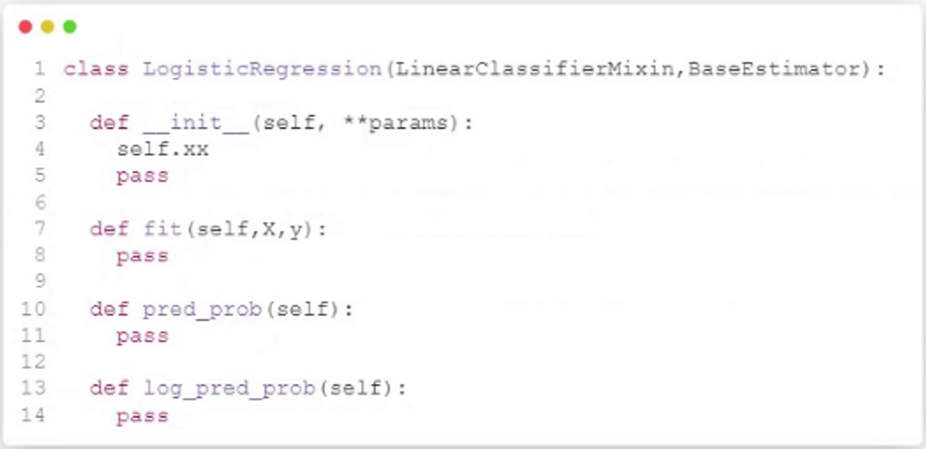
```
class ClassifierMixin:
    """Mixin class for all classifiers in scikit-learn."""

    _estimator_type = "classifier"

    def score(self, X, y, sample_weight=None):
      pass

```



```
clf = LogisticRegression(randomstate=42) # instantiate the estimator
```

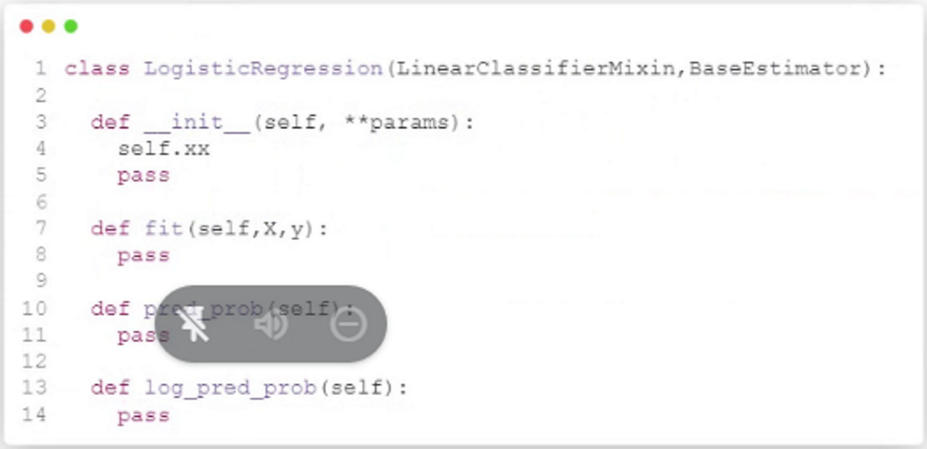
$\uparrow$

This class inherits BaseEstimator and ClassifierMixin



```python
class LogisticRegression(LinearClassifierMixin,BaseEstimator):

  def __init__(self, **params):
    self.xx
    pass

  def fit(self,X,y):
    pass

  def pred_prob(self):
    pass

  def log_pred_prob(self):
    pass
```

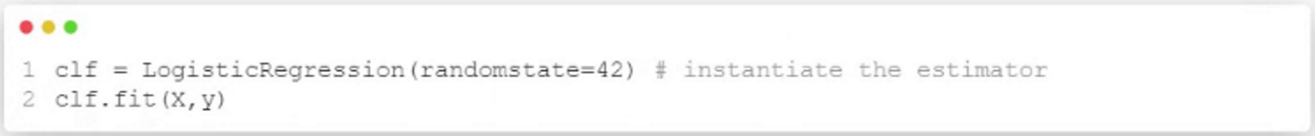
Estimators
Classifier



```python
class LogisticRegression(LinearClassifierMixin,BaseEstimator):

  def __init__(self, **params):
    self.xx
    pass

  def fit(self,X,y):
    pass

  def pred_prob(self):
    pass

  def log_pred_prob(self):
    pass
```

The classifier implementation **must** implement `__init__` and `fit` methods

The data X and (optionally) label Y must be passed to `fit` method and `__init__` always take model speicifc arguments like hyper-parameters



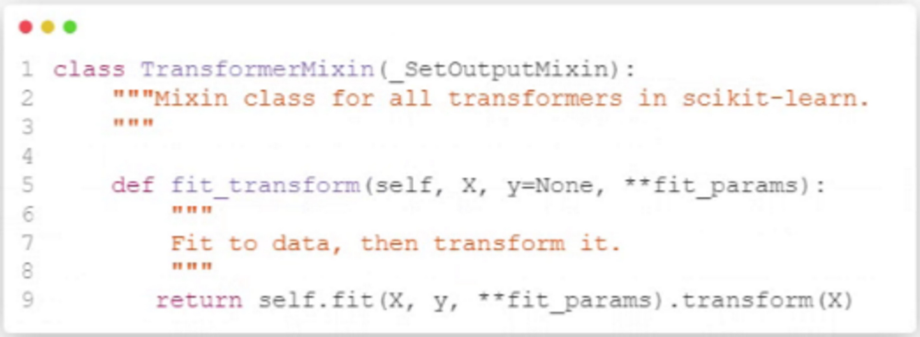
```python
clf = LogisticRegression(randomstate=42) # instantiate the estimator
clf.fit(X,y)
```

Upon executing the `fit` method, some parameters estimated **using the data** are added to the instance(`clf` in this case) attributes.

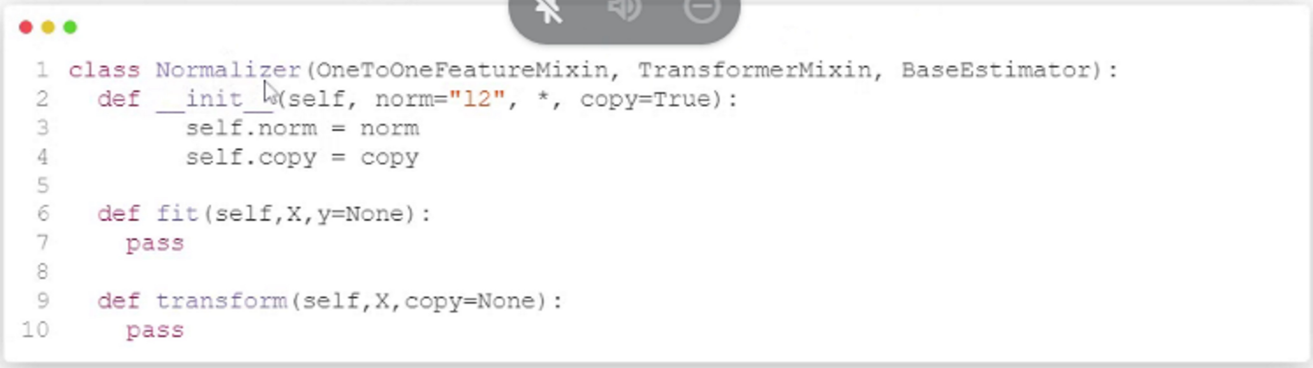
For ex: `coef_` , `intercept_`

Any attributes with a trailing underscore denotes the paramters estimated using data

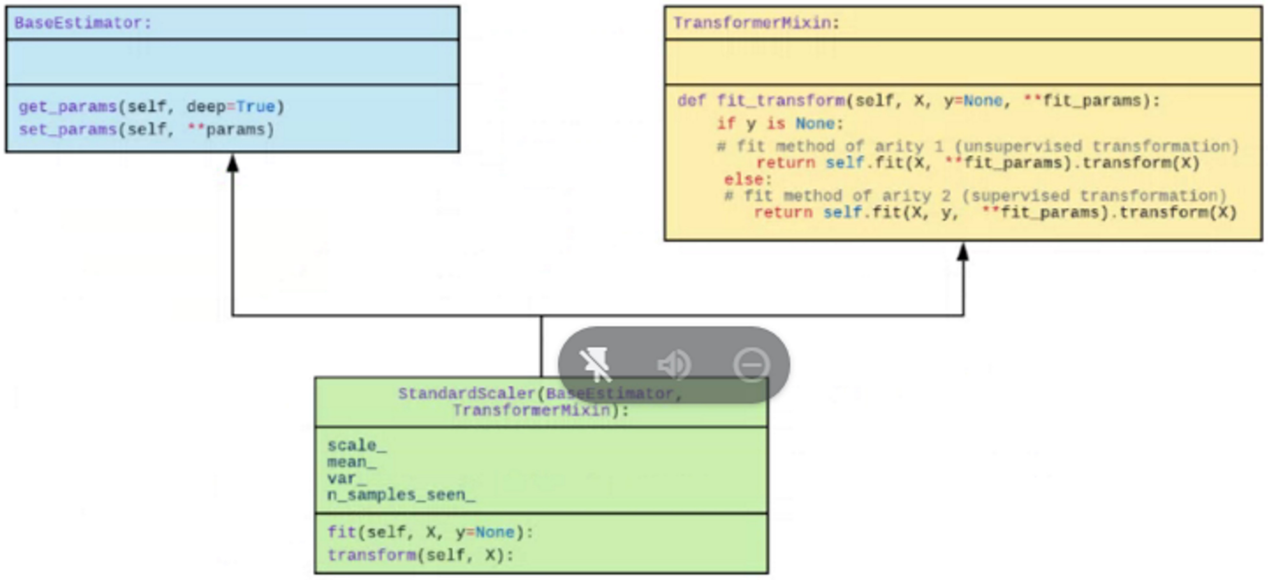
Transformers



```python
class TransformerMixin(_SetOutputMixin):
    """Mixin class for all transformers in scikit-learn.
    """

    def fit_transform(self, X, y=None, **fit_params):
        """
        Fit to data, then transform it.
        """
       return self.fit(X, y, **fit_params).transform(X)
```



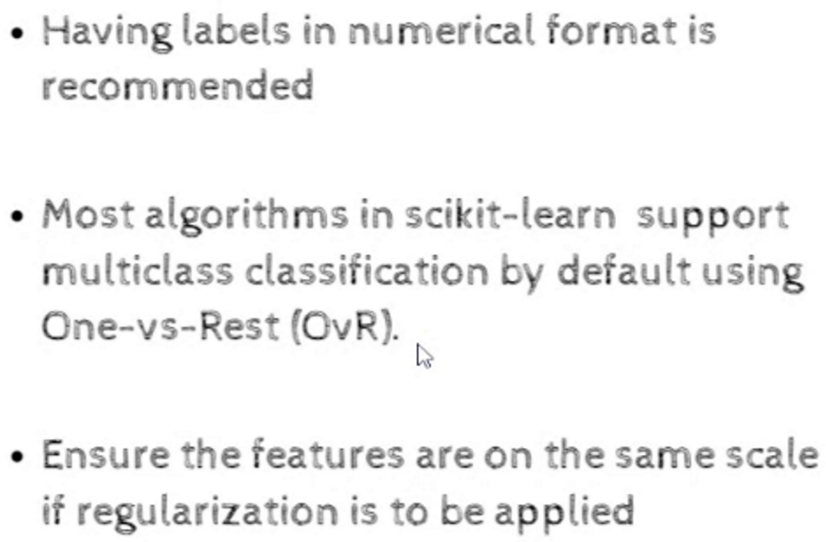
```python
class Normalizer(OneToOneFeatureMixin, TransformerMixin, BaseEstimator):
  def __init__(self, norm="l2", *, copy=True):
        self.norm = norm
        self.copy = copy

  def fit(self,X,y=None):
    pass

  def transform(self,X,copy=None):
    pass
```



```
BaseEstimator:

get_params(self, deep=True)
set_params(self, **params)
```

```
TransformerMixin:

def fit_transform(self, X, y=None, **fit_params):
    if y is None:
    # fit method of arity 1 (unsupervised transformation)
        return self.fit(X, **fit_params).transform(X)
     else:
     # fit method of arity 2 (supervised transformation)
        return self.fit(X, y,  **fit_params).transform(X)
```

```
StandardScaler(BaseEstimator,
TransformerMixin):

scale_
mean_
var_
n_samples_seen_

fit(self, X, y=None):
transform(self, X):
```



- Having labels in numerical format is recommended
- Most algorithms in scikit-learn support multiclass classification by default using One-vs-Rest (OvR).
- Ensure the features are on the same scale if regularization is to be applied

arunprakash@study.iitm.ac.in
https://iitm-pod.slides.com/arunprakash_ai/sklearn-introduction/fullscreen