Pipelines

What is a pipeline-

1. Almost always, we need to tie together many different processes that we use to prepare data for machine learning based model

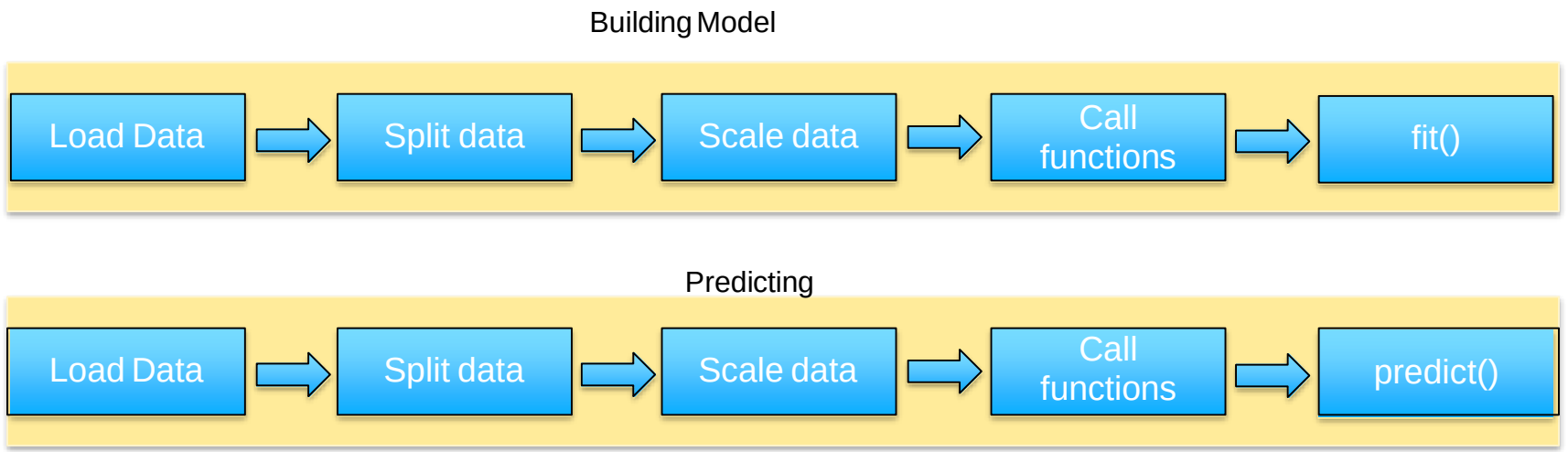
2. It is paramount that the stage of transformation of data represented by these processes are standardized

3. Pipeline class of sklearn helps simplify the chaining of the transformation steps and the model

4. Pipeline, along with the GridsearchCV helps search over the hyperparameter space applicable at each stage

Pipelines

1. Sequentially apply a list of transforms and a final estimator.
2. Intermediate steps of the pipeline must be 'transforms', that is, they must implement fit and transform methods.
3. The final estimator only needs to implement fit
4. Helps standardize the model project by enforcing consistency in building testing and production.



Build a pipeline

1. Import the pipeline class
    a. from sklearn.pipeline import Pipeline

2. Instantiate the class into an object by listing out the transformation steps. In the following example, a scaling function is followed by the logistic algorithm
    a. pipe = Pipeline([(" scaler", MinMaxScaler()), (" lr", logisticregression())])

3. Call the fit() function on the pipeline object
    a. pipe.fit(X_train, y_train)

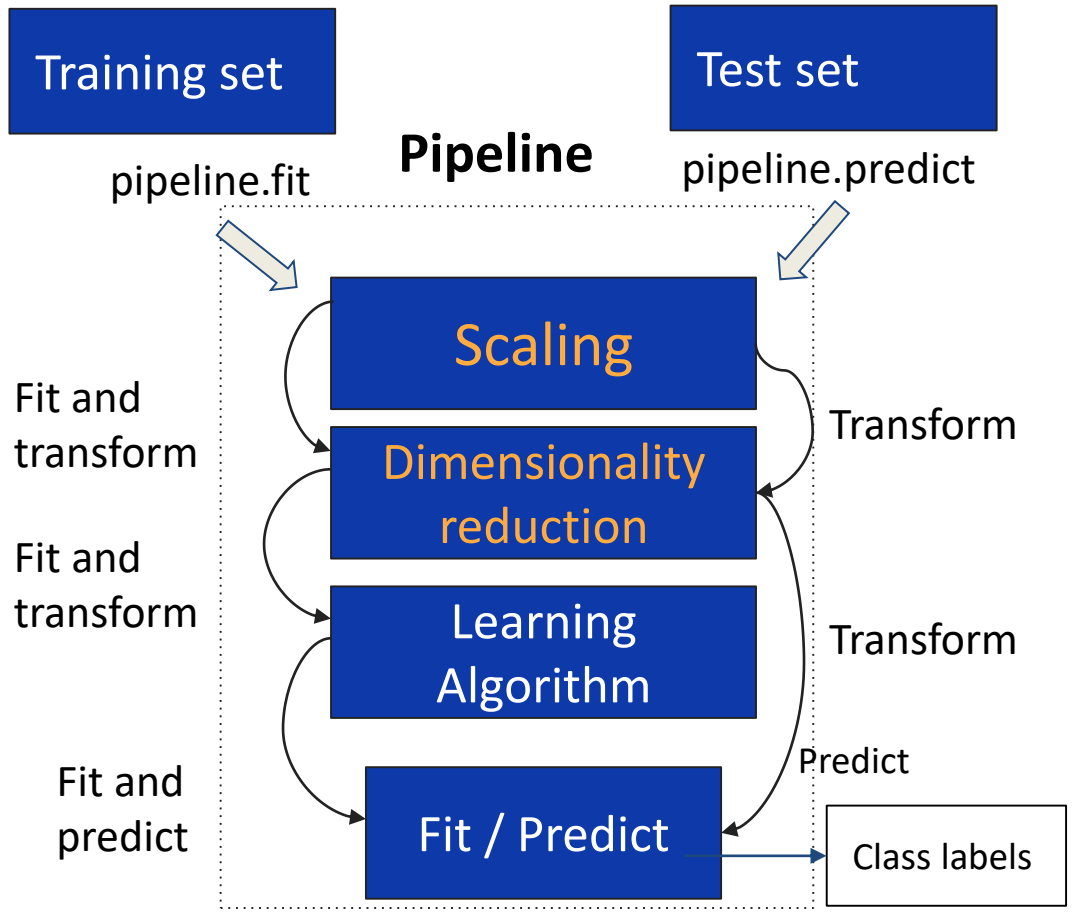
4. Call the score() function on the pipeline object or predict() function
    a. pipe.score(X_test, y_test)

In the step 2b, the pipeline object is created using a dictionary of key:value pairs. The key is specified in strings for e.g. “scaler” followed by the function to be called.

The key is the name given to a step.

Build a pipeline

- The pipeline object requires all the stages to have both ‘fit()’ and “transform()” function except for the last stage when it is an estimator
- The estimator does not have a “transform()” function because it builds the model using the data from previous step. It does not transform the data
- The transform function transforms the input data and emits transformed data as output which becomes the input to the next stage
- pipeline.fit() calls the fit and transform functions on each stage in sequence. In the last stage, if it is an estimator, only the fit function is called to create the model.
- The model become a part of the pipeline automatically

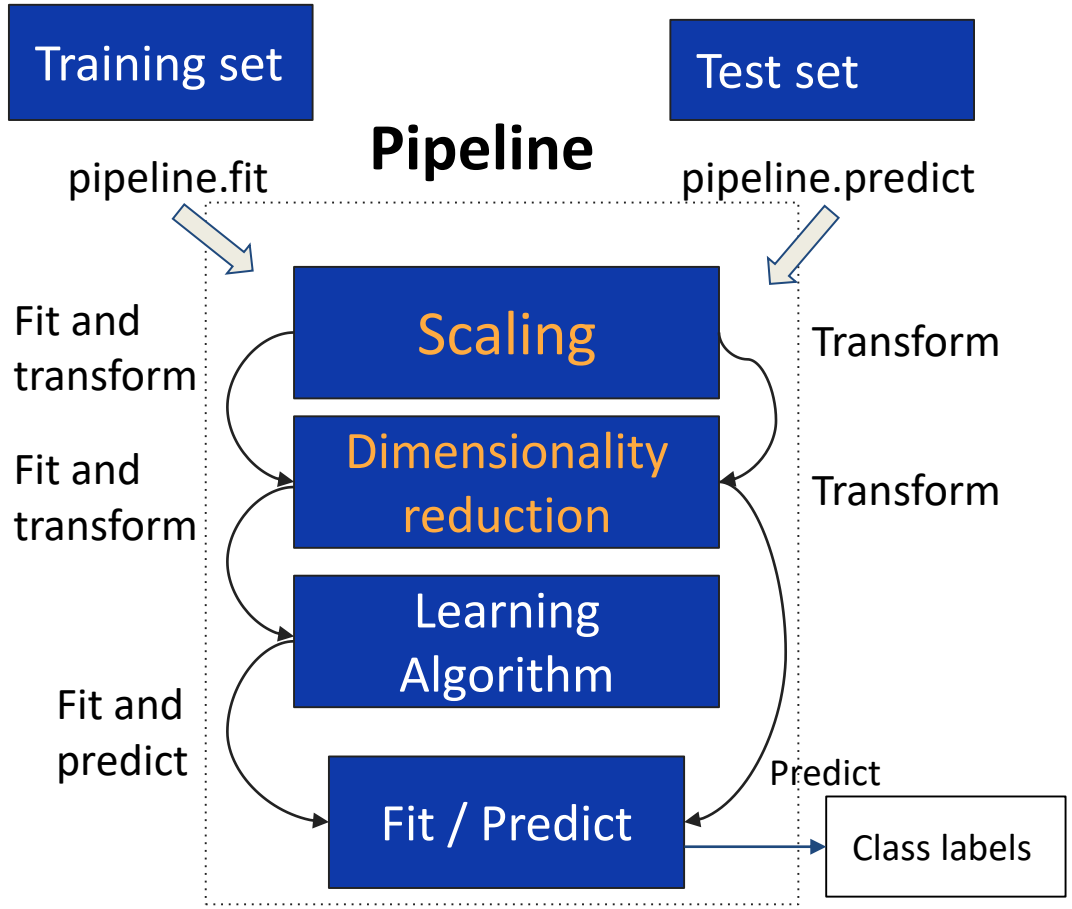


Build a pipeline

- pipeline.predict() calls the transform function at all the stages on the given data
- In the last stage, it jumps the estimator step because the model is already built
- It executes the predict() function of the model

Note:

- A pipeline can be constructed purely for data transformation alone. Which means there it is not mandatory to have an estimator



make_pipeline

- Specifying names for the different stages of a pipeline can lead to ambiguities. When there are multiple stages, each stage has to be uniquely named and we have to make sure there is consistency in the naming process such as usage of lower case letters only, each name should be unique, name should reflect the purpose of the stage etc. Manual naming is prone to ambiguity
- Alternatively we can use "make_pipeline()" function that will create the pipeline and automatically name each step as the lowercase of the name of the function called
  - from sklearn.pipeline import make_pipeline
  - pipe = make_pipeline(MinMaxScaler(), (LogisticRegression()))
  - print(" Pipeline steps:\ n{}". format(pipe.steps))
- The advantage of "make_pipeline" is the consistency in the naming of each stage, we can have multiple stages in a pipeline performing the same transformations. Each stage is guaranteed to have a unique meaningful name

HyperParameter Tuning

Hyper Parameters & Tuning

1. Hyper parameters are like handles available to the modeler to control the behavior of the algorithm used for modeling

2. Hyper parameters are supplied as arguments to the model algorithms while initializing them. For e.g. setting the criterion for decision tree building
   "dt_model = DecisionTreeClassifier(criterion = 'entropy')"

3. To get a list of hyper parameters for a given algorithm, call the function get_params()…for e.g. to get support vector classifier hyper parameters
   1. from sklearn.svm import SVC
   2. svc= SVC()
   3. svc.get_params()

4. Hyper parameters are not learnt from the data as other model parameters are. For e.g. attribute coefficients in a linear model are learnt from data while cost of error is input as hyper parameter.

Hyper Parameters & Tuning

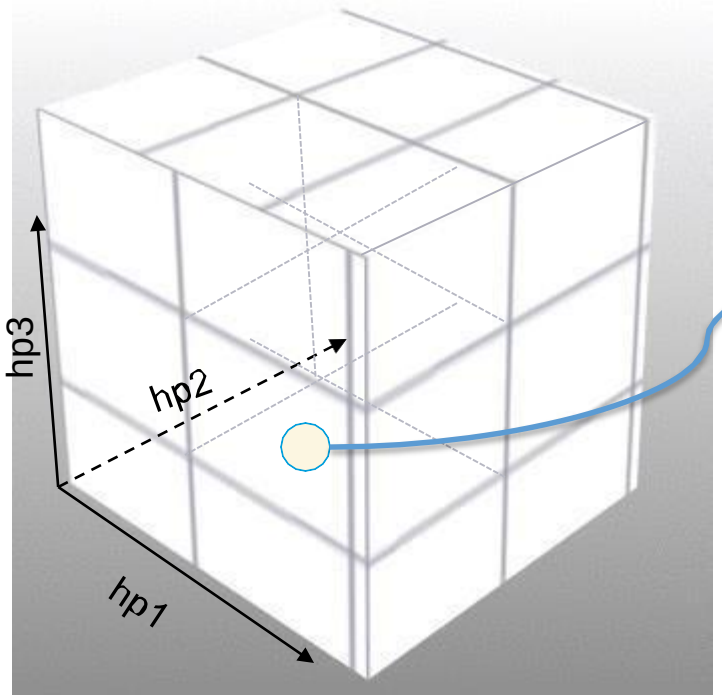
5. Fine tuning the hyper parameters is done in a sequence of steps
    1. Selecting the appropriate model type (regressor or classifier such as sklearn.svm.SVC())
    2. Identify the corresponding parameter space
    3. Decide the method for searching or sampling parameterspace;
    4. Decide the cross-validation scheme to ensure model will generalize
    5. Decide a score function to use to evaluate the model

6. Two generic approaches to searching hyper parameter space include
    1. GridSearchCV which exhaustively considers all parameter combinations
    2. RandomizedSearchCV can sample a given number of candidates from a parameter space with a specified distribution.
7. While tuning hyper parameters, the data should have been split into three parts – Training, validation and testing to **prevent data leak**

8. The testing data should be separately transformed * using the same functions that were used to transform the rest of the data for model building and hyper parameter tuning

* Any transformation where rows influence each other. For e.g. using zscore. OneHotCode transformation does not come into this category. It can be done before splitting the data

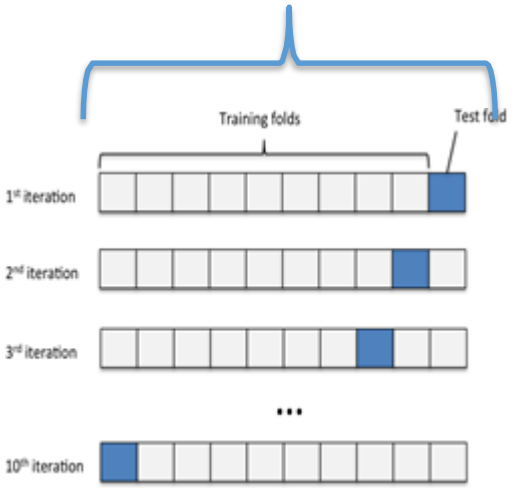
GridSearchCV



Hyper parameter space

One combination of hyper parameters used K times to train and test. The avg score of the K times is the score associated with this combination

This will repeat for all possible combinations i.e. all the cells in the space.



Hyper Parameters & Tuning (GridsearchCV/ RandomizedSearchCv)

RandomizedSearchCV –

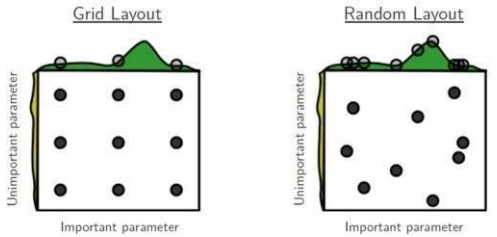
1. Random search differs from grid search. Instead of providing a discrete set of values to explore on each hyperparameter (parameter grid), we provide a statistical distribution.

2. Values for the different hyper parameters are picked up at random from this combine distribution

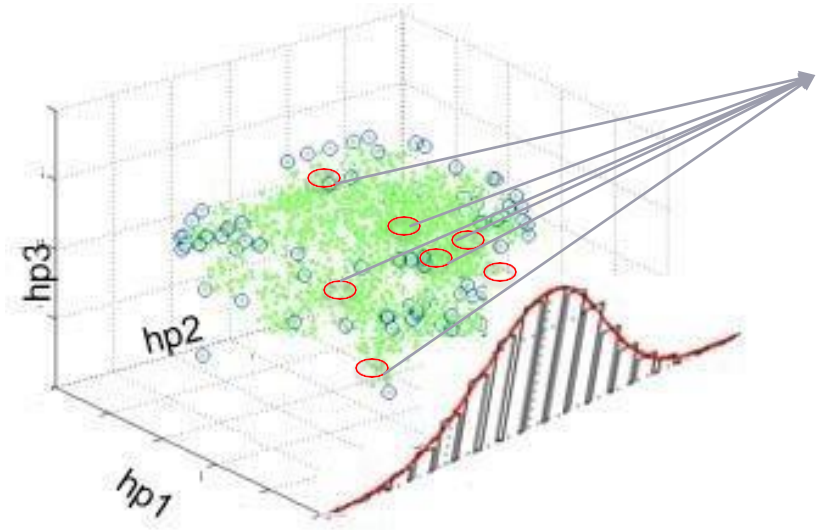
3. The motivation to use random search in place of grid search is that for many cases, hyperparameters are not *equally* important.

> *A Gaussian process analysis of the function from hyper-parameters to validation set performance reveals that* ***for most data sets only a few of the hyper-parameters really matter, but that different hyper-parameters are important on different data sets****. This phenomenon makes grid search a poor choice for configuring algorithms for new data sets. - Bergstra, 2012*

Picture by Bergstra, 2012

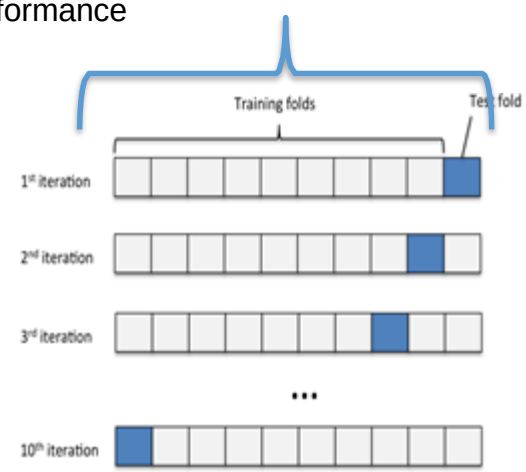


RandomizedSearchCV



Hyper parameter space

Randomly pick up n-iter samples from the hyper parameter distribution as sample, Use it K times and find avg performance



4. In contrast to GridSearchCV, not all combinations are evaluated. A fixed number of parameter settings is sampled from the specified distributions.
5. The number of parameter settings that are tried is given by n_iter
6. If all parameters are presented as a list, sampling without replacement is performed. If at least one parameter is given as a distribution, sampling with replacement is used. It is highly recommended to use continuous distributions for continuous parameters
7. Randomsearch has higher chance of hitting the right combination thangridsearch.