Scikit-learn Tutorial and Introduction to Model Validation

Justin Silverman

Penn State University

02/04/2021

Table of Contents

Section 1

Overview of Scikit-learn

scikit-learn

- Python Framework for Machine Learning built ontop of NumPy and SciPy using matplotlib. It also plays nicely with Pandas.
- Very well designed UI that supports a wide variety of useful machine learning models as well as model selection, evalution, and dataset preprocessing tools.
- Very well documented. Much of this lecture is adapted from the documentation: https://scikit-learn.org/stable/getting_started.html

Installation

Instructions at https://scikit-learn.org/stable/install.html

Estimators: Fitting

Scikit-learn is built around estimators (functions that estimate something from data).

```
from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier(random_state=0)
X = [[ 1,  2,  3],  # 2 samples, 3 features
     [11, 12, 13]]
y = [0, 1]  # classes of each sample
clf.fit(X, y)
```

Each estimator supports a `fit` method which typically takes two inputs `X` and `y` which shold both be numpy arrays or equivalent array-like data types.

By convention `X` is typically of dimension `(n_samples, n_features)` (*i.e.*, samples are rows, columns are features)

`y` is typically the outcome of interest (*e.g.*, sample labels to be predicted or the value to be predicted using regression)

Estimators: Predicting

Once the estimator is fitted, it can be used for predicting target values of new data. You don't need to re-train the estimator:

```
# predict classes of the training data
clf.predict(X)
# predict classes of new data
```

```
## array([0, 1])
```

```
clf.predict([[4, 5, 6], [14, 15, 16]])
```

```
## array([0, 1])
```

Transformers and pre-processors

Data transformation and pre-processing is also considered an estimator in Scikit-learn.

These estimators have `fit` and `transform` methods.

Consider the scaling transformation[1]

$$z = \frac{x - \text{mean}(x)}{\text{sd}(x)}$$

```
from sklearn.preprocessing import StandardScaler
X = [[0, 15],
     [1, -10]]
StandardScaler().fit(X).transform(X)
```

```
## array([[-1.,  1.],
##        [ 1., -1.]])
```

[1]This has to actually be fit to data, especially you will want to use the same mean and standard deviation when you transform the training set.

Transforming Individual Features

Use the `ColumnTransformer` object.

```
import pandas as pd
X = pd.DataFrame(
    {'city': ['London', 'London', 'Paris', 'Sallisaw'],
     'title': ["His Last Bow", "How Watson Learned the Trick",
               "A Moveable Feast", "The Grapes of Wrath"],
     'expert_rating': [5, 3, 4, 5],
     'user_rating': [4, 5, 4, 3]})

from sklearn.compose import ColumnTransformer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.preprocessing import OneHotEncoder
column_trans = ColumnTransformer(
    [('city_category', OneHotEncoder(dtype='int'),['city']),
     ('title_bow', CountVectorizer(), 'title')],
    remainder='drop')

column_trans.fit(X)
```

```
## ColumnTransformer(transformers=[('city_category', OneHotEncoder(dtype='int'),
##                                   ['city']),
##                                  ('title_bow', CountVectorizer(), 'title')])
```

```
column_trans.get_feature_names()
```

```
## ['city_category__x0_London', 'city_category__x0_Paris', 'city_category__x0_Sallisaw', 'title_bow__bow', 'tit
```

```
column_trans.transform(X).toarray()
```

```
## array([[1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0],
##        [1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0],
##        [0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
##        [0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1]])
```

Data Quality Issue

- Data in the real world is dirty
  - Missing: lacking attribute values, lacking certain attributes of interest
    - e.g., occupation=" " (missing data)
  - Noisy: containing noise, errors, or outliers
    - e.g., Salary="−10" (an error)
  - Inconsistent: containing discrepancies in codes or names, e.g.,
    - Age="43", Birthday="03/07/1997"
    - Was rating "1,2,3", now rating "A, B, C"
    - discrepancy between duplicate records

Attributes:

1. npreg – Number of times pregnant
2. glucose – Plasma glucose concentration
3. bp – Blood pressure
4. skin – Triceps skinfold thickness
5. bmi – Body mass index
6. ped – Diabetes pedigree function
7. age – Age

	npreg	glu	bp	skin	bmi	ped	age
1	6	148	72	35	33.6	0.627	50
2	1	85	66	29	26.6	0.351	31
3	1	89	6600	23	28.1	0.167	21
4	3	78	50	32	31	0.248	26
5	2	197	70	45	30.5	0.158	53
6	5	166	72	19	25.8	0.587	51
7	0	118	84	47	45.8	0.551	31
8	one	103	30	38	43.3	0.183	33
9	3	126	88	41	39.3	0.704	27
10	9	119	80	35	29	0.263	29
11	1	97	66	15	23.2	0.487	22
12	5	109	75	26	36	0.546	60
13	3	88	58	11	24.8	0.267	22
14	10	122	78	31	27.6	0.512	45
15	4		60	33	24	0.966	33
16	9	102	76	37	32.9	0.665	46
17	2	90	68	42	38.2	0.503	27
18	4	111	72	47	37.1	1.39	56
19	3	180	64	25	34	0.271	26
20	7	106	92	18		0.235	48
21	9	171	110	24	45.4	0.721	54

Preprocessing

- Handle Missing Values
  - Ignore the records with missing values
  - Estimate missing values

- Remove Outliers
  - Find and remove those values that are significantly different from the others

- Resolve conflicts
  - Merge information from different data sources
  - Find duplicate records and identify the correct information

	npreg	glu	bp	skin	bmi	ped	age
1	6	148	72	35	33.6	0.627	50
2	1	85	66	29	26.6	0.351	31
			6600				
4	3	78	50	32	31	0.248	26
5	2	197	70	45	30.5	0.158	53
6	5	166	72	19	25.8	0.587	51
7	0	118	84	47	45.8	0.551	31
8	1	103	30	38	43.3	0.183	33
9	3	126	88	41	39.3	0.704	27
10	9	119	80	35	29	0.263	29
11	1	97	66	15	23.2	0.487	22
12	5	109	75	26	36	0.546	60
13	3	88	58	11	24.8	0.267	22
14	10	122	78	31	27.6	0.512	45
15	4	97	60	33	24	0.966	33
16	9	102	76	37	32.9	0.665	46
17	2	90	68	42	38.2	0.503	27
18	4	111	72	47	37.1	1.39	56
19	3	180	64	25	34	0.271	26
20	7	106	92	18	39	0.235	48
21	9	171	110	24	45.4	0.721	54

Feature Selection

- Some features are good, others not so good
- Even relevant attributes can also be harmful if they mislead a learning algorithm
- What kinds of features should be removed?
  - Irrelevant features
  - Redundant features

Task: Predicting diabetes based on the medical history of the patient

	npreg	glu	bp	skin	bmi	ped	age	income
1	6	148	72	35	33.6	0.627	50	
2	1	85	66	29	26.6	0.351	31	
3	1	89	6600	23	28.1	0.167	21	
4	3	78	50	32	31	0.248	26	
5	2	197	70	45	30.5	0.158	53	
6	5	166	72	19	25.8	0.587	51	
7	0	118	84	47	45.8	0.551	31	
8	one	103	30	38	43.3	0.183	33	
9	3	126	88	41	39.3	0.704	27	
10	9	119	80	35	29	0.263	29	
11	1	97	66	15	23.2	0.487	22	
12	5	109	75	26	36	0.546	60	
13	3	88	58	11	24.8	0.267	22	
14	10	122	78	31	27.6	0.512	45	
15	4		60	33	24	0.966	33	
16	9	102	76	37	32.9	0.665	46	
17	2	90	68	42	38.2	0.503	27	
18	4	111	72	47	37.1	1.39	56	
19	3	180	64	25	34	0.271	26	
20	7	106	92	18		0.235	48	
21	9	171	110	24	45.4	0.721	54	

Filter Methods

- Features are “filtered” out based on criteria X
- For example:
  - Features with too many missing values
  - Features with too little variation in their values
  - Features with too little correlation with a target class feature

Data ID	F1	F2	F3
1	0	0	1
2	0	1	0
3	1	0	0
4	0	1	1
5	0	1	0
6	0	1	1

```
>>> from sklearn.feature_selection import VarianceThreshold
>>> X = [[0, 0, 1], [0, 1, 0], [1, 0, 0], [0, 1, 1], [0, 1, 0], [0, 1, 1]]
>>> sel = VarianceThreshold(threshold=(.8 * (1 - .8)))
>>> sel.fit_transform(X)
array([[0, 1],
       [1, 0],
       [0, 0],
       [1, 1],
       [1, 0],
       [1, 1]])
```

Pipelines

Transformers and estimators (predictors) can be combined together into a single unifying object: a `Pipeline`.

The pipeline offers the same API as a regular estimator: (*e.g.*, it has `fit` and `predict` methods)

```
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import make_pipeline
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

# create a pipeline object
pipe = make_pipeline(
    StandardScaler(),
    LogisticRegression(random_state=0)
)

# load the iris dataset and split it into train and test sets
X, y = load_iris(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

# fit the whole pipeline
pipe.fit(X_train, y_train)
```

Section 2

Introduction to Model Evaluation

Generalization Error

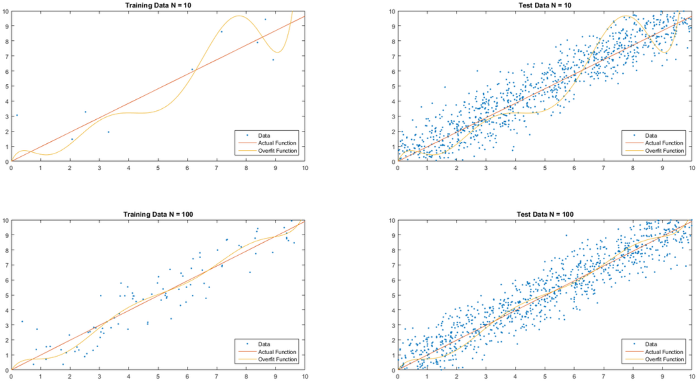


Figure 1: Image from Wikipedia: "Generalization Error"

Generalization Error can be Disasterous

Imagine a hospital using a machine learning algorithm to diagnose lung cancer from CT images only to later realize that the model was overfit to the training data.

Methods for Avoiding Generalization Error

1. Holdout
   - Split your data into two parts "Training" and "Testing". (Typically $\approx$ 70%/30% Training/Testing split)
   - Train your model on the Training Set
   - Evaluate the trained model on the Testing set
2. Cross Validation
   - Partition data into k disjoint subsets
   - For each subset, train on the other k-1 subsets and evaluate trained model on the remaining subset
3. Repeated addition of noise to data and checking that model outputs are robust.

Holdout

```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split

breast_cancer = load_breast_cancer()

# create X (features) and y (response)
X = breast_cancer.data
y = breast_cancer.target

# split data with train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

print('# data=', len(X))
```

```
## # data= 569
```

```
print('# training data =', len(X_train))
```

```
## # training data = 455
```

```
print('# testing data =', len(X_test))
```

```
## # testing data = 114
```

k-fold Cross Validation

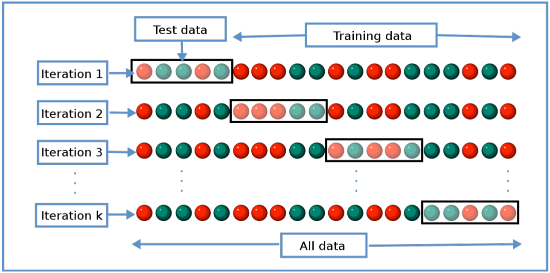


Figure 2: Image from Wikipedia: "Cross Validation (Statistics)"

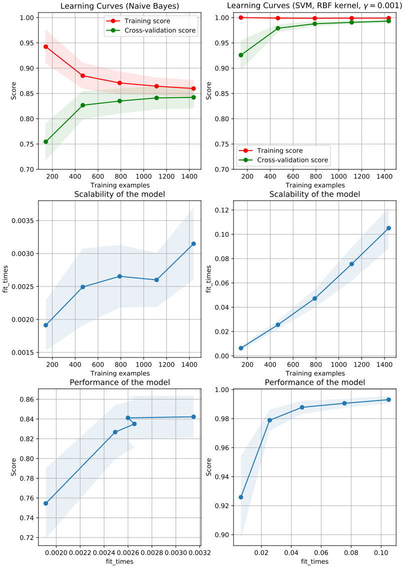
```
from sklearn.datasets import make_regression
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_validate

X, y = make_regression(n_samples=1000, random_state=0)
lr = LinearRegression()

result = cross_validate(lr, X, y)  # defaults to 5-fold CV
result['test_score']  # r_squared score is high because dataset is easy
```

```
## array([1., 1., 1., 1., 1.])
```

Learning Curves



Learning Curves

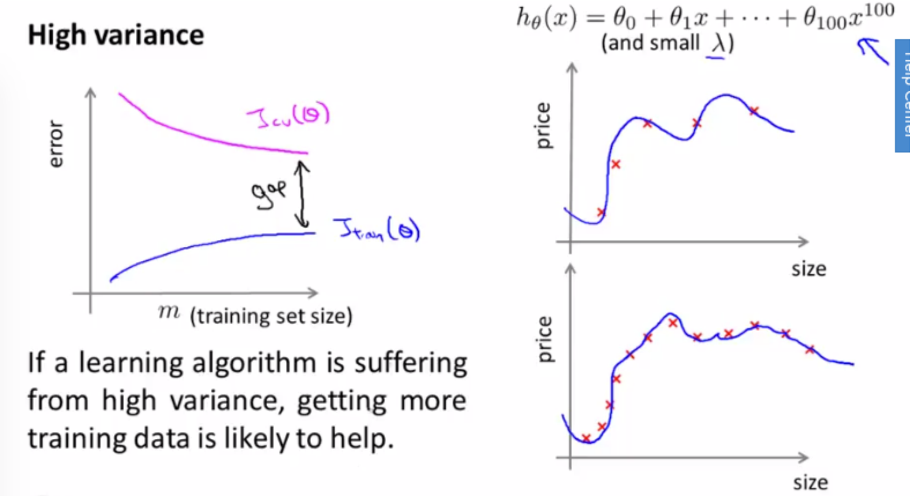


Figure 3: From Andrew Ng, Machine Learning

Learning Curves

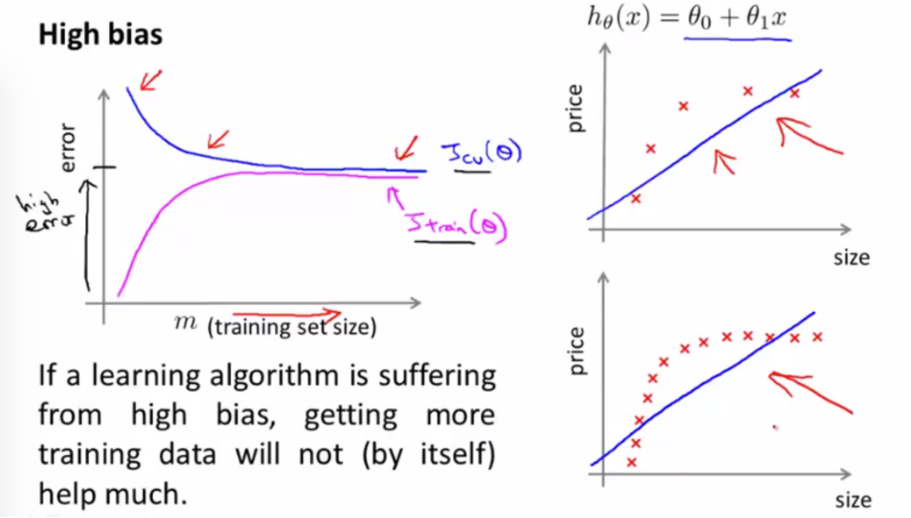


Figure 4: From Andrew Ng, Machine Learning

Caution: Transforming Entire Dataset Before Splitting into Testing/Training Sets

Just as it is important to test a predictor on data held-out from training, preprocessing (such as standardization, feature selection, etc.) and similar data transformations similarly should be learnt from a training set and applied to held-out data for prediction.

```
from sklearn import preprocessing
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.4, random_state=0)
scaler = preprocessing.StandardScaler().fit(X_train) # Learning Scaler
X_train_transformed = scaler.transform(X_train) # Apply to Training
clf = svm.SVC(C=1).fit(X_train_transformed, y_train)
X_test_transformed = scaler.transform(X_test) # Applying to Testing
clf.score(X_test_transformed, y_test)
```

A Pipeline makes it easier to compose estimators, providing this behavior under cross-validation:

```
from sklearn.pipeline import make_pipeline
clf = make_pipeline(preprocessing.StandardScaler(), svm.SVC(C=1))
cross_val_score(clf, X, y, cv=cv)
```

Caution: Data issues causing generalization issues

Always understand how your data was generated.

At this point your model looks like it is generalizing well (e.g., doing well on holdout or cross-validation testing).

But your model may still fail to generalize in practice because of issues with your dataset.

Examples

- The data a collaborator sent you has been cleaned. The model they ultimately want will be applied to data that has not been cleaned.
- The data you have was all collected by a single lab. You want your model to be useable by other labs.

Parameter Searches

All estimators have parameters that you must pick. You can often improve on the default values. But how?

Parameter Searches

All estimators have parameters that you must pick. You can often improve on the default values. But how?

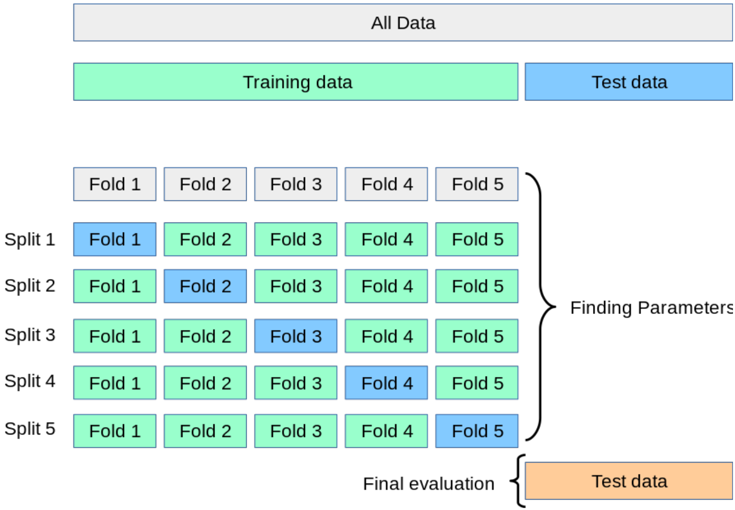


Figure 5: Gridsearch Workflow

Randomized Search

In the following example, we randomly search over the parameter space of the parameters `n_estimators` and `max_depth` of a random forest with a `RandomizedSearchCV` object.

```
from sklearn.datasets import fetch_california_housing
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import train_test_split
from scipy.stats import randint

X, y = fetch_california_housing(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

# define the parameter space that will be searched over
param_distributions = {'n_estimators': randint(1, 5),
                       'max_depth': randint(5, 10)}

# now create a searchCV object and fit it to the data
search = RandomizedSearchCV(estimator=RandomForestRegressor(random_state=0),
                            n_iter=5,
                            param_distributions=param_distributions,
                            random_state=0)
search.fit(X_train, y_train)
```

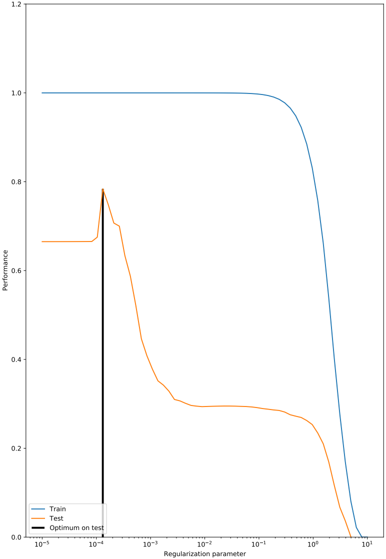
```
search.best_params_
# the search object now acts like a normal random forest estimator
# with max_depth=9 and n_estimators=4
```

```
## {'max_depth': 9, 'n_estimators': 4}
```

```
search.score(X_test, y_test)
```

```
## 0.735363411343253
```

Learning Curves for Model Parameters



Housing Price Regression Example

7.2.1. Boston house prices dataset

Data Set Characteristics:

Number of Instances:	506
Number of Attributes:	13 numeric/categorical predictive. Median Value (attribute 14) is usually the target.
Attribute Information (in order):	• CRIM per capita crime rate by town • ZN proportion of residential land zoned for lots over 25,000 sq.ft. • INDUS proportion of non-retail business acres per town • CHAS Charles River dummy variable (= 1 if tract bounds river; 0 otherwise) • NOX nitric oxides concentration (parts per 10 million) • RM average number of rooms per dwelling • AGE proportion of owner-occupied units built prior to 1940 • DIS weighted distances to five Boston employment centres • RAD index of accessibility to radial highways • TAX full-value property-tax rate per $10,000 • PTRATIO pupil-teacher ratio by town • B 1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town • LSTAT % lower status of the population • MEDV Median value of owner-occupied homes in $1000's
Missing Attribute Values:	None
Creator:	Harrison, D. and Rubinfeld, D.L.

Figure 6: Boston Housing Dataset

We want to be able to predict housing prices.

```
from sklearn.datasets import load_boston
import pandas as pd
data = load_boston()

X = pd.DataFrame(data.data)
y = pd.DataFrame(data.target)
X.columns = data.feature_names

# Brief EDA
X.isna().sum().sum()
```

```
## 0
```

```
X.describe().transpose()[["mean", "std", "min", "max"]]
```

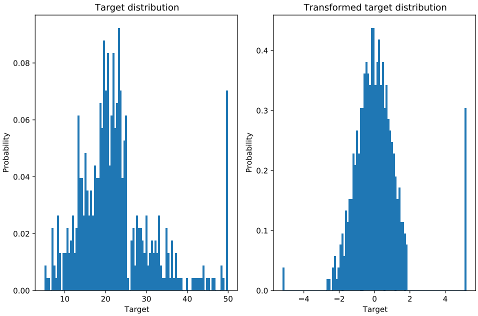
```
##                 mean         std        min       max
## CRIM        3.613524    8.601545    0.00632   88.9762
## ZN         11.363636   23.322453    0.00000  100.0000
## INDUS      11.136779    6.860353    0.46000   27.7400
## CHAS        0.069170    0.253994    0.00000    1.0000
## NOX         0.554695    0.115878    0.38500    0.8710
## RM          6.284634    0.702617    3.56100    8.7800
## AGE        68.574901   28.148861    2.90000  100.0000
## DIS         3.795043    2.105710    1.12960   12.1265
## RAD         9.549407    8.707259    1.00000   24.0000
## TAX       408.237154  168.537116  187.00000  711.0000
## PTRATIO    18.455534    2.164946   12.60000   22.0000
## B         356.674032   91.294864    0.32000  396.9000
## LSTAT      12.653063    7.141062    1.73000   37.9700
```

Housing Price Regression Example

Lets look at what we are trying to predict a bit closer

```
from sklearn.preprocessing import QuantileTransformer, quantile_transform

y_trans = quantile_transform(y,
                             n_quantiles=300,
                             output_distribution='normal',
                             copy=True).squeeze()
```



Boston housing data: distance to employment centers

Housing Price Regression Example

```python
from sklearn import preprocessing, linear_model, pipeline, model_selection
from sklearn.compose import ColumnTransformer, TransformedTargetRegressor

### SETUP
# Create Column Transformer
column_trans = ColumnTransformer([
  ("quantile_numeric",QuantileTransformer(n_quantiles=300,output_distribution='normal'),
    ["ZN", "INDUS", "AGE", "B", "LSTAT"]),
  ("passthrough_boolean", "passthrough", ["CHAS"]),
  ("standard_numeric", preprocessing.StandardScaler(),
    ["CRIM", "NOX", "RM", "DIS", "RAD", "TAX", "PTRATIO"])
    ], remainder="passthrough")

# Make Transformed Target Regressor
regr_trans = TransformedTargetRegressor(
    regressor=linear_model.Ridge(),
    transformer=QuantileTransformer(n_quantiles=300,
                                output_distribution='normal'))

# Make Pipeline
pipe = pipeline.make_pipeline( column_trans, regr_trans )

### END SETUP
```

Housing Price Regression Example

```
from scipy.stats import uniform
from sklearn.metrics import r2_score, mean_squared_error

# Test Train Split
X_train, X_test, y_train, y_test = model_selection.train_test_split(X,y,test_size=0.2)

# Tune alpha by randomized CV
# Chosen by default performance measure for Ridge - R^2
paramdist = {'regressor__alpha': uniform(0.1, 0.9)}
search = model_selection.RandomizedSearchCV(estimator = regr_trans, n_iter=10,
                                            param_distributions=paramdist)
search.fit(X_train, y_train)
```

```
print("Best Parameter: ", search.best_params_)

# Evaluate Performance of Model on Held Out Training Set
```

```
## Best Parameter:  {'regressor__alpha': 0.8212732241467374}
```

```
y_pred = search.predict(X_test)

# Evaluate Performance with R^2 Performance Metric
r2_score(y_test, y_pred)
```

```
## 0.717535051372932
```

```
mean_squared_error(y_test, y_pred)
```

```
## 27.238578531357597
```