# Machine Learning: Assignment 1

import numpy as np  
import pandas as pd  
import time  
import gc  
import random  
from sklearn.model\_selection import cross\_val\_score, GridSearchCV, cross\_validate, train\_test\_split  
from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix, f1\_score  
from sklearn.datasets import make\_classification  
from sklearn.svm import SVC  
from sklearn.tree import DecisionTreeClassifier  
from sklearn.neural\_network import MLPClassifier  
from sklearn.ensemble import GradientBoostingClassifier  
from sklearn.preprocessing import StandardScaler, normalize  
from sklearn.decomposition import PCA  
from sklearn.impute import SimpleImputer  
from sklearn.neighbors import KNeighborsClassifier  
from sklearn.model\_selection import validation\_curve  
from sklearn.neural\_network import MLPClassifier  
import seaborn as sns  
import matplotlib.pyplot as plt  
from yellowbrick.model\_selection import LearningCurve, ValidationCurve  
from sklearn.preprocessing import OneHotEncoder

# 1. Data Import, leansing Setup and helper functions

class Data():  
 def dataAllocation(self,path):  
 # df = pd.read\_csv(path)  
 # x\_data = df.iloc[:, :-1]  
 # y\_data = df.iloc[:, -1 ]  
 # return x\_data,y\_data  
 X, y = make\_classification(n\_samples=2000, n\_features=20, n\_informative=10, n\_redundant=0, random\_state=rs)  
 return X, y  
 def trainSets(self,x\_data,y\_data):  
 x\_train, x\_test, y\_train, y\_test = train\_test\_split(x\_data, y\_data, test\_size = 0.3, random\_state = rs, shuffle = True)  
 return x\_train, x\_test, y\_train, y\_test  
  
data = 'data/pima-indians-diabetes.csv'  
rs = 614  
dataset = Data()  
x\_data,y\_data = dataset.dataAllocation(data)  
x\_train, x\_test, y\_train, y\_test = dataset.trainSets(x\_data,y\_data)  
# print("Heatmap for Features")  
# data\_corr = sns.heatmap(pd.DataFrame(x\_train).corr(), cmap='Blues')

# 2. Decision Tree Classifier

class DTClassifier():  
  
 def trainTest(self,x\_train,x\_test, y\_train):  
 df = []  
 for i in range(16):  
 for j in range(20):  
 dt\_clf = DecisionTreeClassifier(max\_depth=i+1, min\_samples\_leaf=j+1)  
 dt\_clf.fit(x\_train, y\_train)  
   
 y\_predict\_train = dt\_clf.predict(x\_train)  
 y\_predict\_test = dt\_clf.predict(x\_test)  
 df.append([i+1, j+1, "Train", f1\_score(y\_train, y\_predict\_train)])  
 df.append([i+1, j+1, "Test", f1\_score(y\_test, y\_predict\_test)])  
  
 return pd.DataFrame(df, columns=["Depth", "Leaf Size", "Sample Type", "F1 Score" ])  
   
 def hyperParameterTuning(self,x\_train,y\_train):  
 param\_grid = {'max\_depth': range(1, 21), 'min\_samples\_leaf': range(1, 20)}  
 tuned = GridSearchCV(estimator = DecisionTreeClassifier(random\_state = rs), param\_grid = param\_grid, cv=10)  
 tuned.fit(x\_train, y\_train)  
 print(tuned.best\_params\_)  
 return tuned.best\_score\_, tuned.best\_params\_

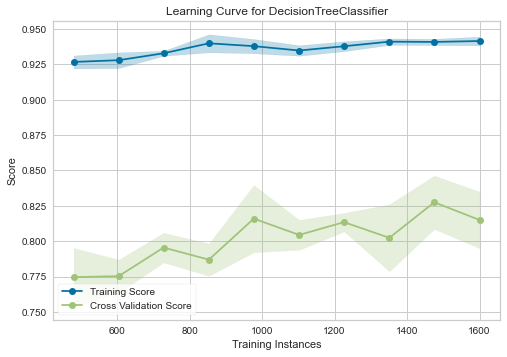
dt = DTClassifier()  
df = dt.trainTest(x\_train, x\_test, y\_train)  
g = sns.FacetGrid(df, hue="Sample Type", col="Depth", height=4, col\_wrap=4)  
g.map(sns.pointplot, "Leaf Size", "F1 Score" )

---------------------------------------------------------------------------  
KeyboardInterrupt Traceback (most recent call last)  
<ipython-input-4-831e8817202b> in <module>  
 2 df = dt.trainTest(x\_train, x\_test, y\_train)  
 3 g = sns.FacetGrid(df, hue="Sample Type", col="Depth", height=4, col\_wrap=4)  
----> 4 g.map(sns.pointplot, "Leaf Size", "F1 Score" )  
  
~/anaconda3/lib/python3.8/site-packages/seaborn/axisgrid.py in map(self, func, \*args, \*\*kwargs)  
 760   
 761 # Draw the plot  
--> 762 self.\_facet\_plot(func, ax, plot\_args, kwargs)  
 763   
 764 # Finalize the annotations and layout  
  
~/anaconda3/lib/python3.8/site-packages/seaborn/axisgrid.py in \_facet\_plot(self, func, ax, plot\_args, plot\_kwargs)  
 844   
 845 # Draw the plot  
--> 846 func(\*plot\_args, \*\*plot\_kwargs)  
 847   
 848 # Sort out the supporting information  
  
~/anaconda3/lib/python3.8/site-packages/seaborn/categorical.py in pointplot(x, y, hue, data, order, hue\_order, estimator, ci, n\_boot, units, seed, markers, linestyles, dodge, join, scale, orient, color, palette, errwidth, capsize, ax, \*\*kwargs)  
 3328 capsize=None, ax=None, \*\*kwargs):  
 3329   
-> 3330 plotter = \_PointPlotter(x, y, hue, data, order, hue\_order,  
 3331 estimator, ci, n\_boot, units, seed,  
 3332 markers, linestyles, dodge, join, scale,  
  
~/anaconda3/lib/python3.8/site-packages/seaborn/categorical.py in \_\_init\_\_(self, x, y, hue, data, order, hue\_order, estimator, ci, n\_boot, units, seed, markers, linestyles, dodge, join, scale, orient, color, palette, errwidth, capsize)  
 1671 orient, color, palette, errwidth=None, capsize=None):  
 1672 """Initialize the plotter."""  
-> 1673 self.establish\_variables(x, y, hue, data, orient,  
 1674 order, hue\_order, units)  
 1675 self.establish\_colors(color, palette, 1)  
  
~/anaconda3/lib/python3.8/site-packages/seaborn/categorical.py in establish\_variables(self, x, y, hue, data, orient, order, hue\_order, units)  
 201   
 202 # Group the numeric data  
--> 203 plot\_data, value\_label = self.\_group\_longform(vals, groups,  
 204 group\_names)  
 205   
  
~/anaconda3/lib/python3.8/site-packages/seaborn/categorical.py in \_group\_longform(self, vals, grouper, order)  
 252 for g in order:  
 253 try:  
--> 254 g\_vals = grouped\_vals.get\_group(g)  
 255 except KeyError:  
 256 g\_vals = np.array([])  
  
~/anaconda3/lib/python3.8/site-packages/pandas/core/groupby/groupby.py in get\_group(self, name, obj)  
 687 raise KeyError(name)  
 688   
--> 689 return obj.\_take\_with\_is\_copy(inds, axis=self.axis)  
 690   
 691 def \_\_iter\_\_(self):  
  
~/anaconda3/lib/python3.8/site-packages/pandas/core/series.py in \_take\_with\_is\_copy(self, indices, axis, \*\*kwargs)  
 840 See the docstring of `take` for full explanation of the parameters.  
 841 """  
--> 842 return self.take(indices=indices, axis=axis, \*\*kwargs)  
 843   
 844 def \_ixs(self, i: int, axis: int = 0):  
  
~/anaconda3/lib/python3.8/site-packages/pandas/core/series.py in take(self, indices, axis, is\_copy, \*\*kwargs)  
 816   
 817 indices = ensure\_platform\_int(indices)  
--> 818 new\_index = self.index.take(indices)  
 819   
 820 if is\_categorical\_dtype(self):  
  
~/anaconda3/lib/python3.8/site-packages/pandas/core/indexes/base.py in take(self, indices, axis, allow\_fill, fill\_value, \*\*kwargs)  
 762 )  
 763 taken = self.values.take(indices)  
--> 764 return self.\_shallow\_copy(taken)  
 765   
 766 def \_assert\_take\_fillable(  
  
~/anaconda3/lib/python3.8/site-packages/pandas/core/indexes/numeric.py in \_shallow\_copy(self, values, \*\*kwargs)  
 109 if values is not None and not self.\_can\_hold\_na:  
 110 # Ensure we are not returning an Int64Index with float data:  
--> 111 return self.\_shallow\_copy\_with\_infer(values=values, \*\*kwargs)  
 112 return super().\_shallow\_copy(values=values, \*\*kwargs)  
 113   
  
~/anaconda3/lib/python3.8/site-packages/pandas/core/indexes/base.py in \_shallow\_copy\_with\_infer(self, values, \*\*kwargs)  
 565 except (TypeError, ValueError):  
 566 pass  
--> 567 return Index(values, \*\*attributes)  
 568   
 569 def \_update\_inplace(self, result, \*\*kwargs):  
  
~/anaconda3/lib/python3.8/site-packages/pandas/core/indexes/base.py in \_\_new\_\_(cls, data, dtype, copy, name, tupleize\_cols, \*\*kwargs)  
 387   
 388 # maybe coerce to a sub-class  
--> 389 if is\_signed\_integer\_dtype(data.dtype):  
 390 return Int64Index(data, copy=copy, dtype=dtype, name=name)  
 391 elif is\_unsigned\_integer\_dtype(data.dtype):  
  
~/anaconda3/lib/python3.8/site-packages/pandas/core/dtypes/common.py in is\_signed\_integer\_dtype(arr\_or\_dtype)  
 882 """  
 883   
--> 884 return \_is\_dtype\_type(arr\_or\_dtype, classes\_and\_not\_datetimelike(np.signedinteger))  
 885   
 886   
  
KeyboardInterrupt:

best\_score, best\_params = dt.hyperParameterTuning(x\_train,y\_train)  
print(best\_score)

{'max\_depth': 10, 'min\_samples\_leaf': 5}  
0.8150000000000001

dt\_tuned = DecisionTreeClassifier(max\_depth=best\_params['max\_depth'], min\_samples\_leaf=best\_params['min\_samples\_leaf'], random\_state=rs)  
  
sizes = np.linspace(0.3, 1.0, 10)  
  
# Instantiate the classification model and visualizer  
visualizer = LearningCurve(  
 dt\_tuned, scoring='f1\_weighted', train\_sizes=sizes, n\_jobs=4  
)  
  
visualizer.fit(x\_data,y\_data) # Fit the data to the visualizer  
visualizer.show()



<matplotlib.axes.\_subplots.AxesSubplot at 0x7ff332eea8b0>

# 3. Support Vector Machine

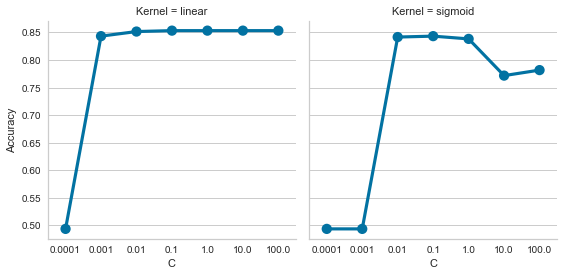
class SupportVectorMachine():  
 def dataPreProcess(self,x\_train,x\_test):  
 scaler = StandardScaler()  
 scaled\_x\_train = scaler.fit\_transform(x\_train)  
 scaled\_x\_test = scaler.transform(x\_test)  
 return scaled\_x\_train, scaled\_x\_test  
   
 def trainTest(self,scaled\_x\_train,scaled\_x\_test, y\_train, y\_test):  
 cs = [x/10000 for x in [1, 10, 100, 1000, 10000, 100000, 1000000]]  
 df = []  
 for c in cs:  
 for k in ["linear", "sigmoid"]:  
 model = SVC(kernel = k, C=c)  
 model.fit(scaled\_x\_train,y\_train)  
 y\_predict\_train = model.predict(scaled\_x\_train)  
 y\_predict\_test = model.predict(scaled\_x\_test)  
 df.append([k, c, accuracy\_score(y\_predict\_test, y\_test)])  
 return pd.DataFrame(df, columns=["Kernel", "C", "Accuracy"])  
   
 def hyperParameterTuning(self,scaled\_x\_train, y\_train):  
 param\_grid = {'C': [x/10000 for x in [1, 10, 100, 1000, 10000, 100000, 1000000]],   
 'kernel': ["linear", "sigmoid"]}   
 svm\_tune = SVC(gamma = "auto")  
 svm\_cv = GridSearchCV(estimator = svm\_tune, param\_grid = param\_grid, n\_jobs=5, return\_train\_score=True)  
 svm\_cv.fit(scaled\_x\_train, y\_train)  
 best\_score = svm\_cv.best\_score\_  
 return best\_score, svm\_cv.best\_params\_

svm = SupportVectorMachine()  
scaled\_x\_train, scaled\_x\_test = svm.dataPreProcess(x\_train,x\_test)  
df = svm.trainTest(scaled\_x\_train,scaled\_x\_test, y\_train, y\_test)  
best\_score, best\_params = svm.hyperParameterTuning(scaled\_x\_train, y\_train)  
print(best\_score, best\_params)

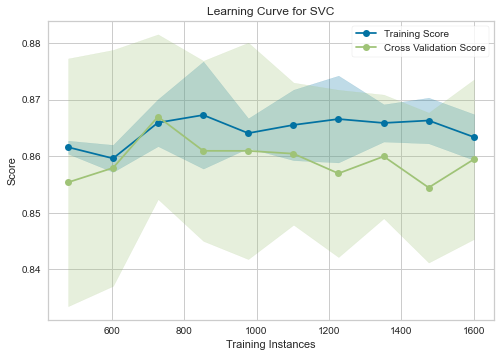
0.8614285714285714 {'C': 0.01, 'kernel': 'linear'}

g = sns.FacetGrid(df, col="Kernel", height=4)  
g.map(sns.pointplot, "C", "Accuracy")

<seaborn.axisgrid.FacetGrid at 0x7ff331e3ea30>



dt\_tuned = SVC(kernel=best\_params['kernel'], C=best\_params['C'])  
  
sizes = np.linspace(0.3, 1.0, 10)  
  
# Instantiate the classification model and visualizer  
visualizer = LearningCurve(  
 dt\_tuned, scoring='f1\_weighted', train\_sizes=sizes, n\_jobs=4  
)  
  
visualizer.fit(x\_data,y\_data) # Fit the data to the visualizer  
visualizer.show()



<matplotlib.axes.\_subplots.AxesSubplot at 0x7ff313fa2340>

# 4. KNN

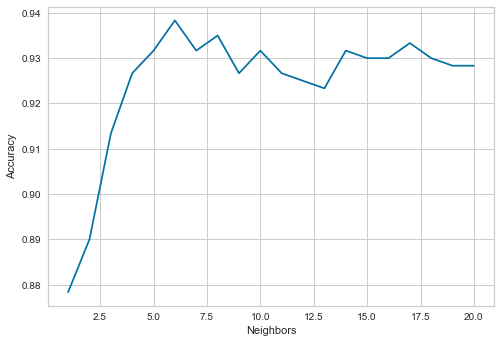
class KNN():  
 def testTrain(self,x\_train,x\_test, y\_train):  
 df = []  
 for i in range(20):  
 model = KNeighborsClassifier(n\_neighbors= i+1)  
 model.fit(x\_train, y\_train)  
 y\_predict\_test = model.predict(x\_test)  
 df.append([i+1, accuracy\_score(y\_predict\_test, y\_test)])  
 return pd.DataFrame(df, columns= ["Neighbors", "Accuracy"])  
   
 def hyperParameterTuning(self,x\_train,y\_train):  
 tuned = GridSearchCV(KNeighborsClassifier(), {"n\_neighbors" : range(1, 21)})  
 tuned.fit(x\_train, y\_train)  
 return tuned.best\_score\_, tuned.best\_params\_

knn = KNN()  
df = knn.testTrain(x\_train,x\_test, y\_train)  
best\_score, best\_params = knn.hyperParameterTuning(x\_train,y\_train)  
print(best\_score, best\_params)

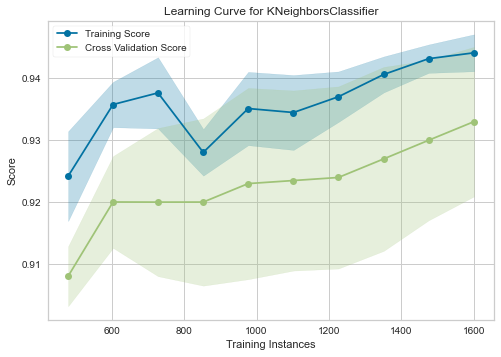
0.9242857142857142 {'n\_neighbors': 10}

sns.lineplot(data=df, x="Neighbors", y="Accuracy")

<matplotlib.axes.\_subplots.AxesSubplot at 0x7fa68a7bae50>



knn\_tuned = KNeighborsClassifier(n\_neighbors=best\_params['n\_neighbors'])  
  
sizes = np.linspace(0.3, 1.0, 10)  
  
# Instantiate the classification model and visualizer  
visualizer = LearningCurve(  
 knn\_tuned, scoring='f1\_weighted', train\_sizes=sizes, n\_jobs=4  
)  
  
visualizer.fit(x\_data,y\_data) # Fit the data to the visualizer  
visualizer.show()



<matplotlib.axes.\_subplots.AxesSubplot at 0x7fa68aa473a0>

# 5. Neural Network

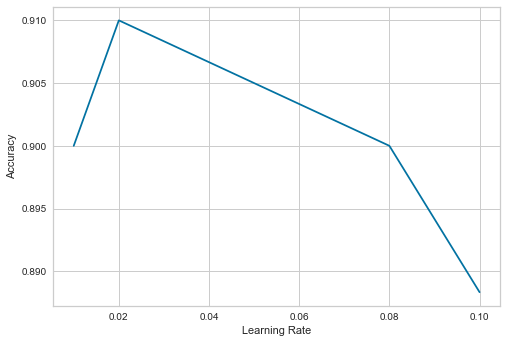
np.linspace(1,150,30).astype('int')

array([ 1, 6, 11, 16, 21, 26, 31, 36, 42, 47, 52, 57, 62,  
 67, 72, 78, 83, 88, 93, 98, 103, 108, 114, 119, 124, 129,  
 134, 139, 144, 150])

class NN():  
 def dataPreProcess(self,x\_train,x\_test):  
 scaler = StandardScaler()  
 scaled\_x\_train = scaler.fit\_transform(x\_train)  
 scaled\_x\_test = scaler.transform(x\_test)  
 return scaled\_x\_train, scaled\_x\_test  
  
 def trainTest(self,scaled\_x\_train,scaled\_x\_test, y\_train, y\_test):  
 df = []  
 for i in [0.01, 0.02, 0.04, 0.08, 0.1]:  
 model = MLPClassifier(max\_iter=300, learning\_rate\_init=i)  
 model.fit(scaled\_x\_train,y\_train)  
 y\_predict\_train = model.predict(scaled\_x\_train)  
 y\_predict\_test = model.predict(scaled\_x\_test)  
 df.append([i, accuracy\_score(y\_predict\_test, y\_test)])  
 return pd.DataFrame(df, columns=["Learning Rate", "Accuracy"])  
   
 def hyperParameterTuning(self,scaled\_x\_train, y\_train):  
 param\_grid = {  
 'hidden\_layer\_sizes': [x\*\*2 for x in range(2, 11)],  
 'learning\_rate\_init': [0.01, 0.02, 0.04, 0.08, 0.1],  
 }  
 tuned = GridSearchCV(MLPClassifier(max\_iter=300), param\_grid = param\_grid, cv=10)  
 tuned.fit(scaled\_x\_train, y\_train)  
 return tuned.best\_score\_, tuned.best\_params\_

nn = NN()  
scaled\_x\_train, scaled\_x\_test = nn.dataPreProcess(x\_train,x\_test)  
df = nn.trainTest(scaled\_x\_train,scaled\_x\_test, y\_train, y\_test)  
sns.lineplot(data=df, x="Learning Rate", y="Accuracy")

<matplotlib.axes.\_subplots.AxesSubplot at 0x7fc23a4ec220>



best\_score, best\_params = nn.hyperParameterTuning(scaled\_x\_train, y\_train)  
print(best\_score, best\_params)

---------------------------------------------------------------------------  
NameError Traceback (most recent call last)  
<ipython-input-25-1c9bb1b94653> in <module>  
----> 1 best\_score, best\_params = nn.hyperParameterTuning(scaled\_x\_train, y\_train)  
 2 print(best\_score, best\_params)  
  
<ipython-input-21-2d6d5c55abb1> in hyperParameterTuning(self, scaled\_x\_train, y\_train)  
 21 'learning\_rate\_init': [0.01, 0.02, 0.04, 0.08, 0.1],  
 22 }  
---> 23 tuned = GridSearchCV(MLPClassifier(max\_iter=300), param\_grid = mlp\_parameters, cv=10)  
 24 tuned.fit(scaled\_x\_train, y\_train)  
 25 return tuned.best\_score\_, tuned.best\_params\_  
  
NameError: name 'mlp\_parameters' is not defined

boost\_tuned = MLPClassifier(max\_iter=300, learning\_rate\_init=best\_params['learning\_rate\_init'], hidden\_layer\_sizes=best\_params['hidden\_layer\_sizes'])  
  
sizes = np.linspace(0.3, 1.0, 10)  
  
# Instantiate the classification model and visualizer  
visualizer = LearningCurve(  
 boost\_tuned, scoring='f1\_weighted', train\_sizes=sizes, n\_jobs=4  
)  
  
visualizer.fit(x\_data,y\_data) # Fit the data to the visualizer  
visualizer.show()

# 6. Boost

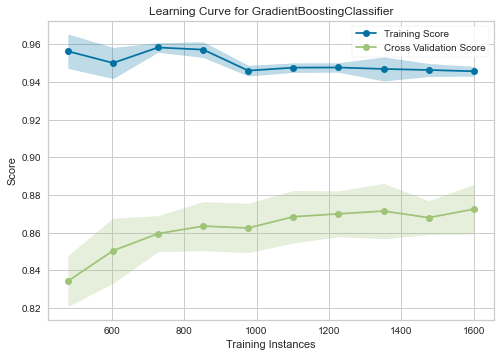
class Boost():  
 def dataPreProcess(self,x\_train,x\_test):  
 scaler = StandardScaler()  
 scaled\_x\_train = scaler.fit\_transform(x\_train)  
 scaled\_x\_test = scaler.transform(x\_test)  
 return scaled\_x\_train, scaled\_x\_test  
  
  
 def trainTest(self,scaled\_x\_train,scaled\_x\_test, y\_train, y\_test):  
 df = []  
 for i in range(50, 251, 5):  
 model = GradientBoostingClassifier(n\_estimators=i, max\_depth=3, min\_samples\_leaf=10)  
 model.fit(scaled\_x\_train,y\_train)  
 y\_predict\_train = model.predict(scaled\_x\_train)  
 y\_predict\_test = model.predict(scaled\_x\_test)  
 df.append([i, accuracy\_score(y\_predict\_test, y\_test)])  
 return pd.DataFrame(df, columns=["Estimators", "Accuracy"])  
   
 def hyperParameterTuning(self,scaled\_x\_train, y\_train):  
 param\_grid = {'n\_estimators': range(50, 151, 40), 'max\_depth': range(2, 4)}  
 tuned = GridSearchCV(GradientBoostingClassifier(), param\_grid, cv=10)  
 tuned.fit(scaled\_x\_train, y\_train)  
 return tuned.best\_score\_, tuned.best\_params\_

boost = Boost()  
scaled\_x\_train, scaled\_x\_test = boost.dataPreProcess(x\_train,x\_test)  
df = boost.trainTest(scaled\_x\_train,scaled\_x\_test, y\_train, y\_test)  
sns.lineplot(data=df, x="Estimators", y="Accuracy")

best\_score, best\_params = boost.hyperParameterTuning(scaled\_x\_train, y\_train)  
print(best\_score, best\_params)

0.8757142857142857 {'max\_depth': 3, 'n\_estimators': 130}

boost\_tuned = GradientBoostingClassifier(n\_estimators=best\_params['n\_estimators'], max\_depth=best\_params['max\_depth'], min\_samples\_leaf=100)  
  
sizes = np.linspace(0.3, 1.0, 10)  
  
# Instantiate the classification model and visualizer  
visualizer = LearningCurve(  
 boost\_tuned, scoring='f1\_weighted', train\_sizes=sizes, n\_jobs=4  
)  
  
visualizer.fit(x\_data,y\_data) # Fit the data to the visualizer  
visualizer.show()



<matplotlib.axes.\_subplots.AxesSubplot at 0x7fc2374704f0>