Lecture ~~6~~ 7

Non-Parametric Regression

Why use non-parametric methods

- Parameterized models include relatively few variables that describe relationships between independent and dependent variables
  - **Pro:** Simple to implement and understand
  - **Con:** Limited predictive power, must make assumptions about the relationship

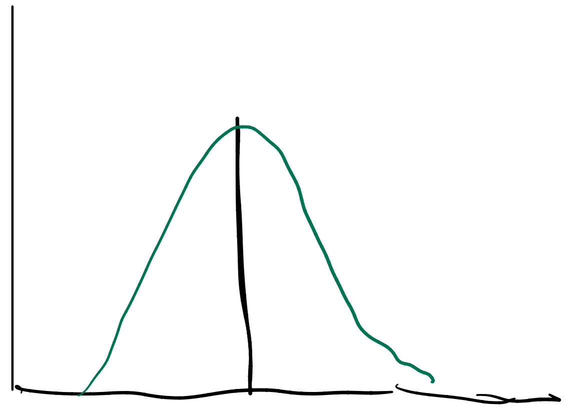
- Non-parameterized models do not assume any form of relationship and instead learn the relationship themselves

Review of Kernel Density Estimation

- Each kernel produces a probability distribution based on its data point

$$\hat{f}(x) = \frac{1}{\text{nh}} \sum_{i=1}^{n} K(z)$$

$$z = \frac{x - X_i}{h}$$



Kernel Regression

- Kernel regression is analogous to kernel density estimation, except we are predicting an **expected value** instead of a **probability density**

$$E(Y|X) = m(x)$$

High if x and x_i are close

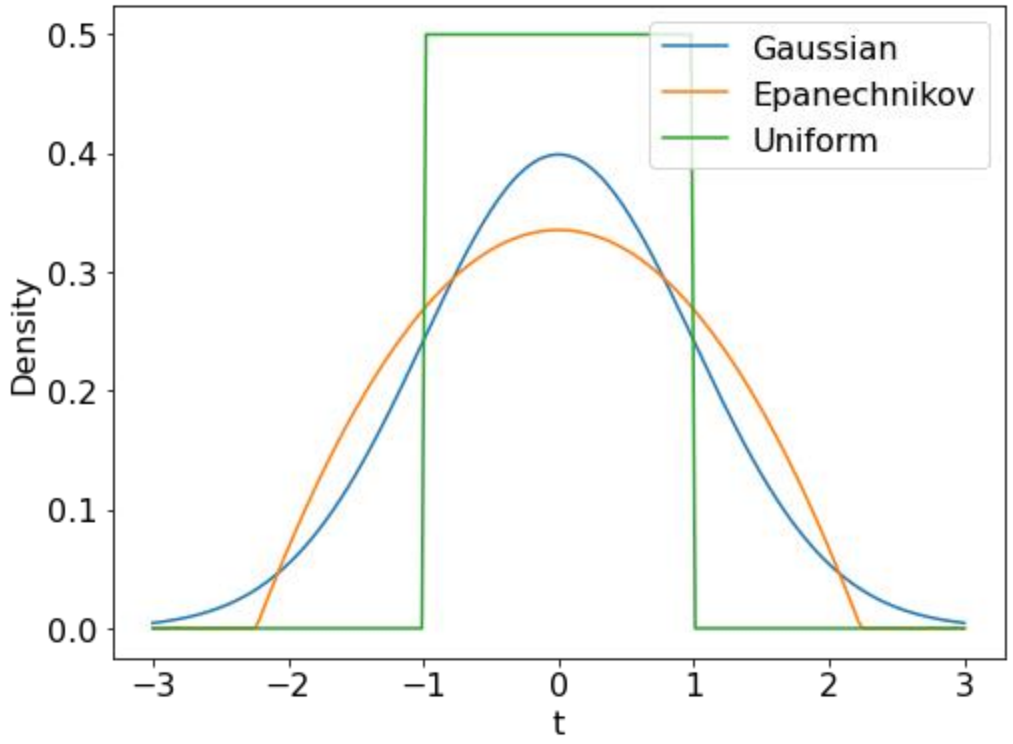
$$\hat{m}_h(x) = \frac{\sum_{i=1}^{n} K_h(x - x_i) * y_i}{\sum_{i=1}^{n} K_h(x - x_i)}$$

point of interest

$$K_h(x - x_i) = \frac{1}{h} K\left(\frac{x - x_i}{h}\right)$$

bandwidth

Kernel Examples



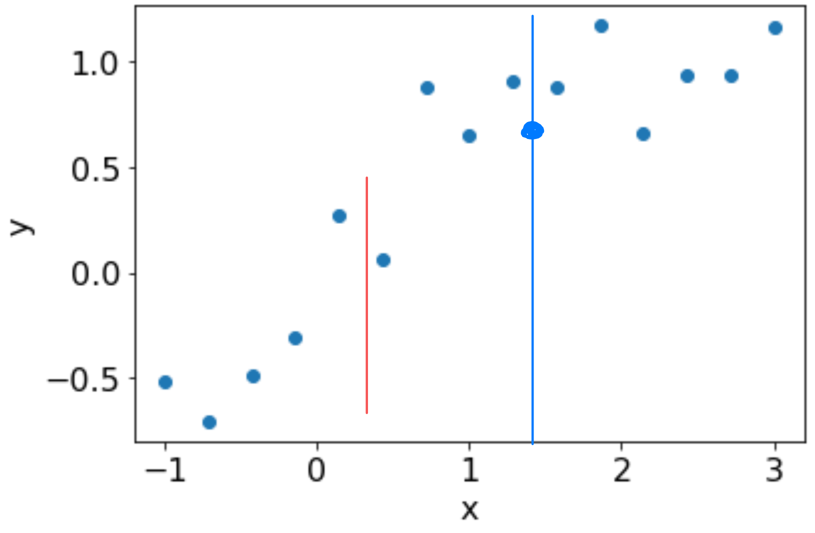
Working an example

- I want to estimate the value of y at x=1.25 based on the data on the right
- I have decided to use a Gaussian kernel with a bandwidth of 1

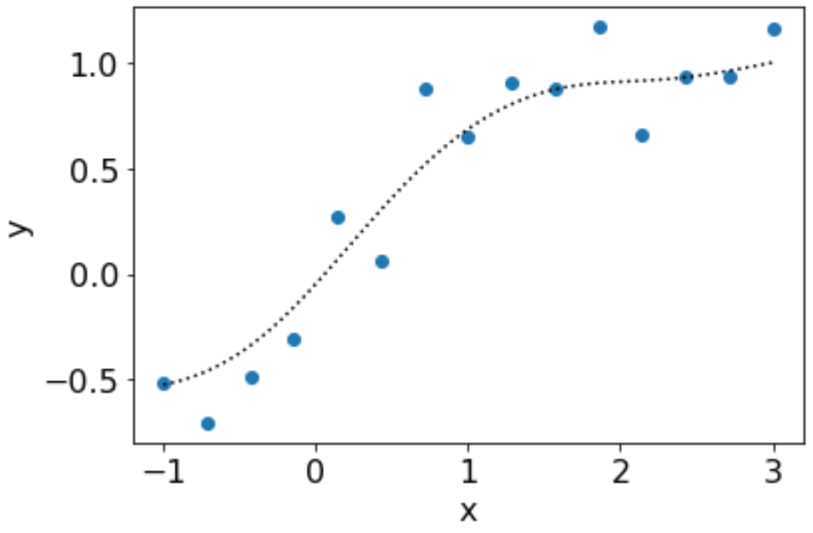
divide

X_i	Y_i	$K\left(\frac{(x-X_i)}{h}\right)$	$K_h(x-X_i) * Y_i$
0.71	0.875	0.346	0.302
1.00	0.647	0.386	0.250
1.28	0.905	0.397	0.361
1.57	0.880	0.379	0.333
1.86	1.172	0.267	0.389
2.14	0.663	0.199	0.177

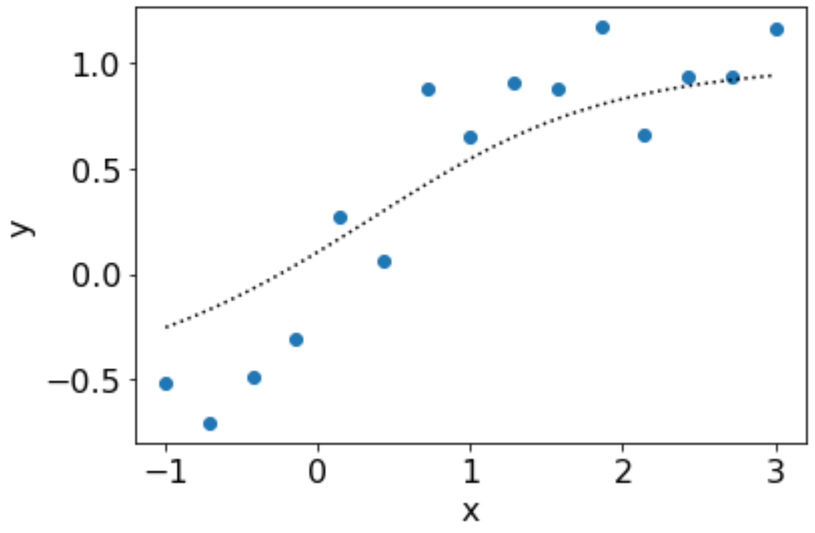
$$m(1.25) = 0.639$$



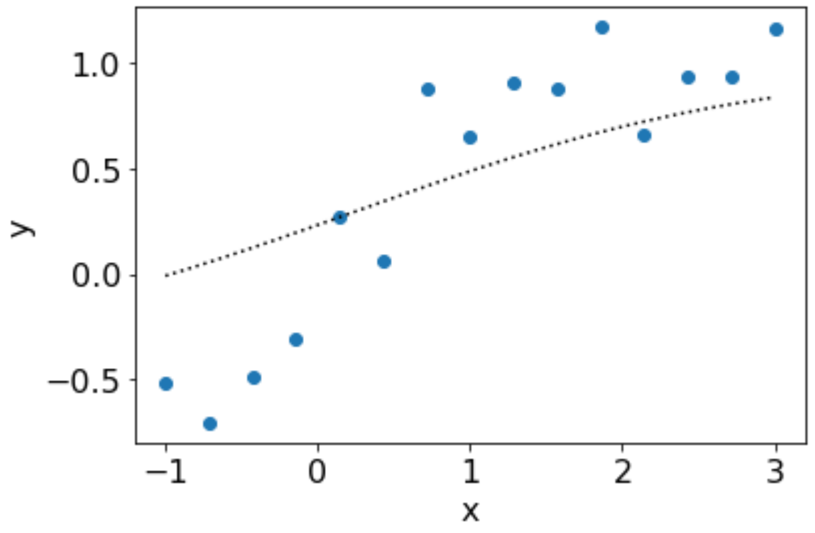
Example solution



H = 0.5



H = 1.0



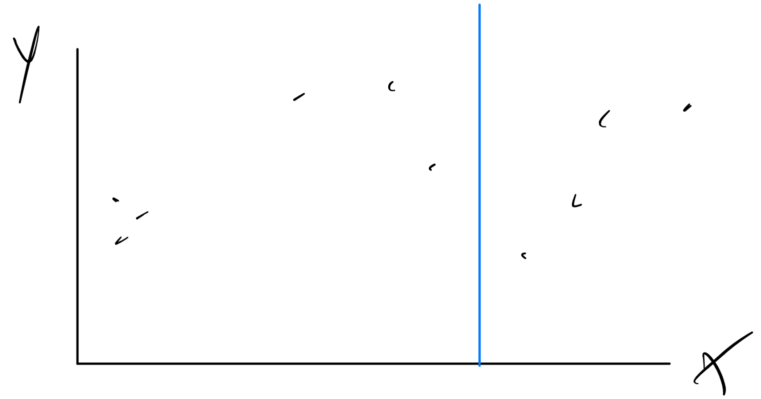
H = 1.5

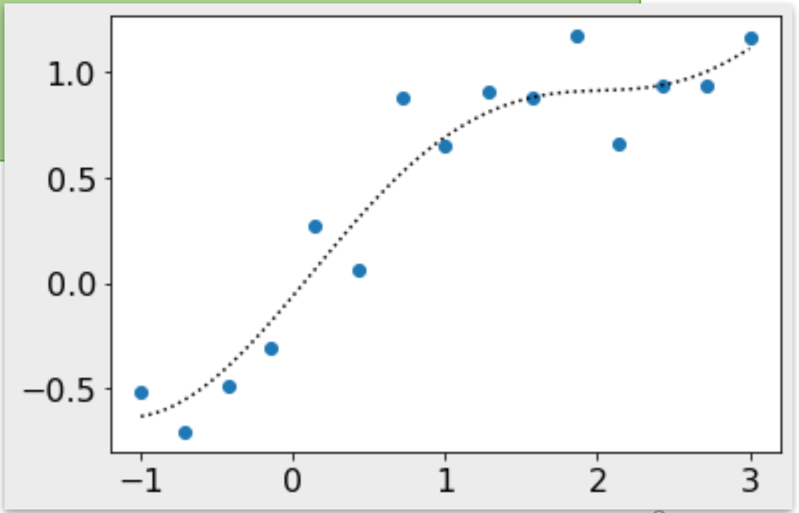
Solving using statsmodels

```
import statsmodels.api as sm

kr = sm.nonparametric.KernelReg(y, x, var_type=['c'], ckertype='gaussian')
y_prediction, _ = kr.fit()

print(kr.bw)
>>> array([0.48250042])

plt.scatter(x,y)
y_pred, _ = kr.fit(xr)
plt.plot(xr, y_pred, color='black', linestyle='dotted')
```

['c', 'o', 'c']

'o'





Kernel regression with categorical variables

- We can also use categorical kernels in kernel regression
  - For unordered variables

$$K(x) = \begin{cases} 1 - \lambda\,; X_i = x \\ \frac{\lambda}{\underbrace{c}_{\text{\# of categories}} - 1}\,; otherwise \end{cases}$$

  - For ordered variables

$$K(x) = \begin{cases} 1 - \lambda\,; X_i = x \\ \frac{1 - \lambda}{2} * \lambda^{|X_i - x|}\,; otherwise \end{cases}$$

Local Regression

- An alternative method of non-parametric regression is **Lo**cally **We**ighted **S**catterplot **S**moothing (LOWESS)
  - Sometimes called **Lo**cally **E**stimated **S**catterplot **S**moothing (LOESS)

- LOWESS fits multiple models to subsets of the data. Each data point is weighted according to its distance from the point of interest

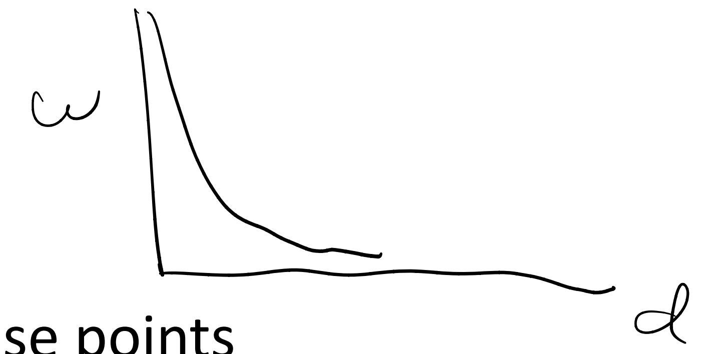
- Each model uses polynomial regression within its data subset

LOESS Steps

For each point of interest (x)

1. Select the *k* nearest points to x
2. Calculate the weight of each neighbour point, typically using the tri-cubic function

$$W = \left(1 - \left|\frac{d}{d_{max}}\right|^3\right)^3$$



3. Perform a weighted linear regression using these points

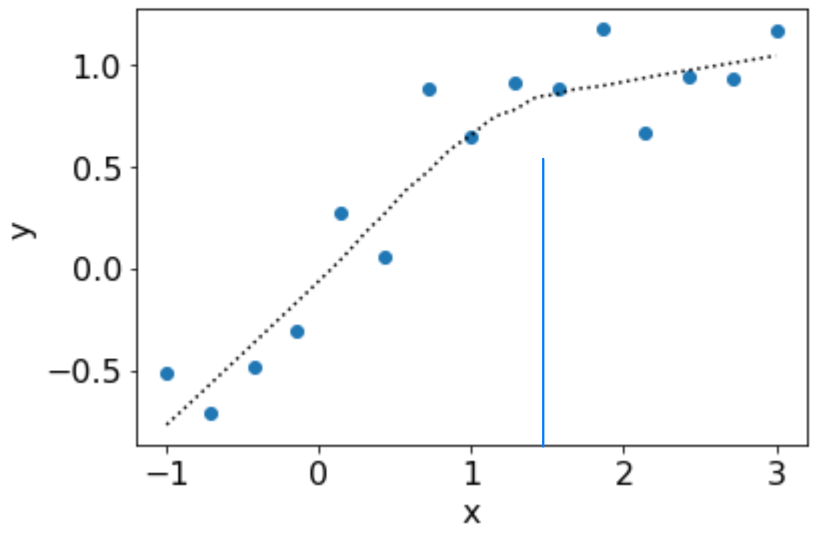
Solving using statsmodels

data

points to estimate

```
y_pred = sm.nonparametric.lowess(y, x, xvals=xr)
```

x_1	x_2	x_3
1	2	3
2	4	6
3	6	NA



Support Vector Regression

- Support Vector Machines are models that seek to find a hyperplane defined by weighted combinations of input values
  - For classification, an SVM seeks to find a plane that separates classes with the maximum distance between the plane and the nearest data points
  - For regression, an SVM seeks to find a plane such that as many data points as possible lie near the plane

- Support Vector Machines rely on a kernel to calculate the weight of each training data point for an inference point

Support Vector Regression

- Support vector regression training can be expressed as a quadratic optimization problem

$$minimize \frac{1}{2}\|w\|^2 + C * \sum_{i=1}^{n}(\xi_i + \xi_i^*)$$

$$y_i - w^T x_i - b \le \epsilon + \xi_i$$

$$w^T x_i + b - y_i \le \epsilon + \xi_i^*$$

$$\xi_i, \xi_i^* \ge 0$$

Support Vector Regression

- It is helpful to solve the previous equation in it’s dual form
  - Don’t worry about primal and dual forms, you will learn about that next block

$$\min \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}(\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*)(x_i^T x_j) + \epsilon \sum_{i=1}^{N}(\alpha_i + \alpha_i^*) + \sum_{i=1}^{N} y_i(\alpha_i - \alpha_i^*)$$

$$\sum_{i=1}^{N}(\alpha_i + \alpha_i^*) = 0$$

$$0 \leq \ \alpha_i \leq C$$
$$0 \leq \ \alpha_i^* \leq C$$

SVR Kernels

- Some problems cannot be described using the linear model above

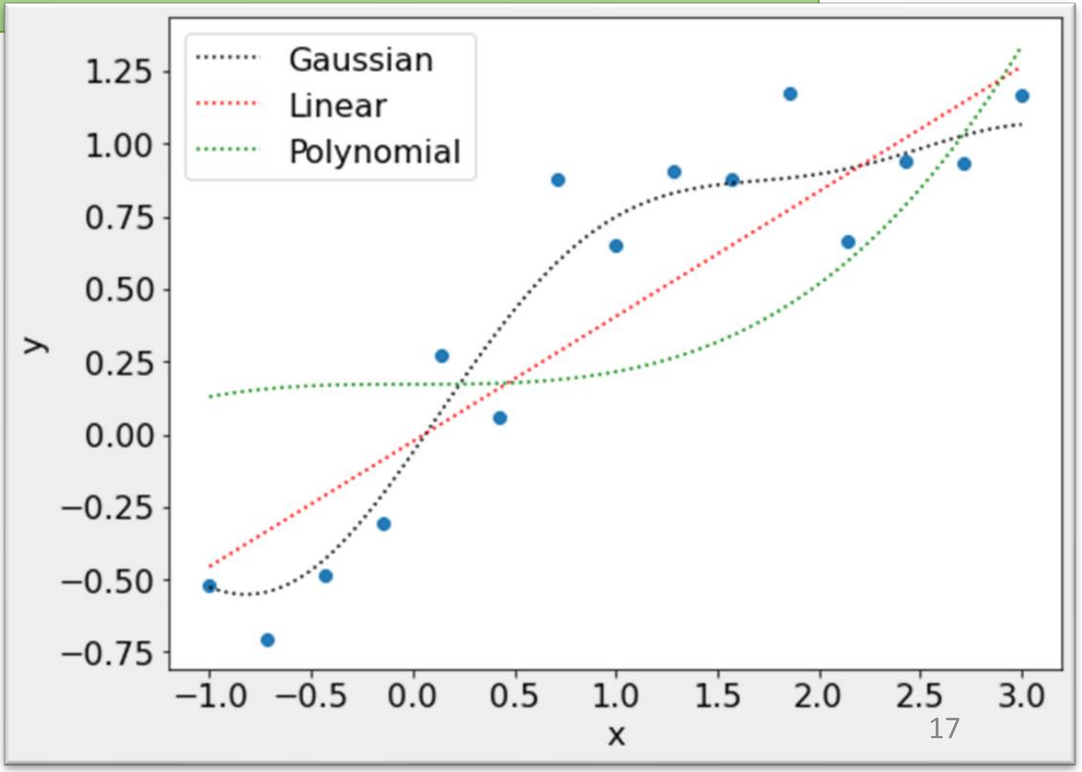
- For these, we can again use a kernel to calculate distance, replacing the dot product distance we used previously

Radial basis function

Solving using Scikit-Learn

```
x_vec = x.reshape(-1,1)
s_rbf = SVR(C=1.0, epsilon=0.1, kernel='rbf').fit(x_vec ,y)
s_linear = SVR(C=1.0, epsilon=0.1, kernel='linear').fit(x_vec ,y)
s_poly = SVR(C=1.0, epsilon=0.1, kernel='poly').fit(x_vec ,y)

y_pred_rbf = s_rbf.predict(xr.reshape(-1,1))
```

from sklearn.svm import SVR



Tree-Based Regression

- Tree-based classification models (e.g. random forest, xgboost, etc.) have equivalent regression formulations
- A tree-based regression model has leaf nodes with individual values of the response variable
- The latest developments in tree-based models are **boosting trees**, which are the current state-of-the-art for tabular prediction

How Boosting Trees Work

- The model is initialized with a single decision tree

$$\hat{y} = F_0(x)$$

- Using the existing model, calculate the residuals of the training set

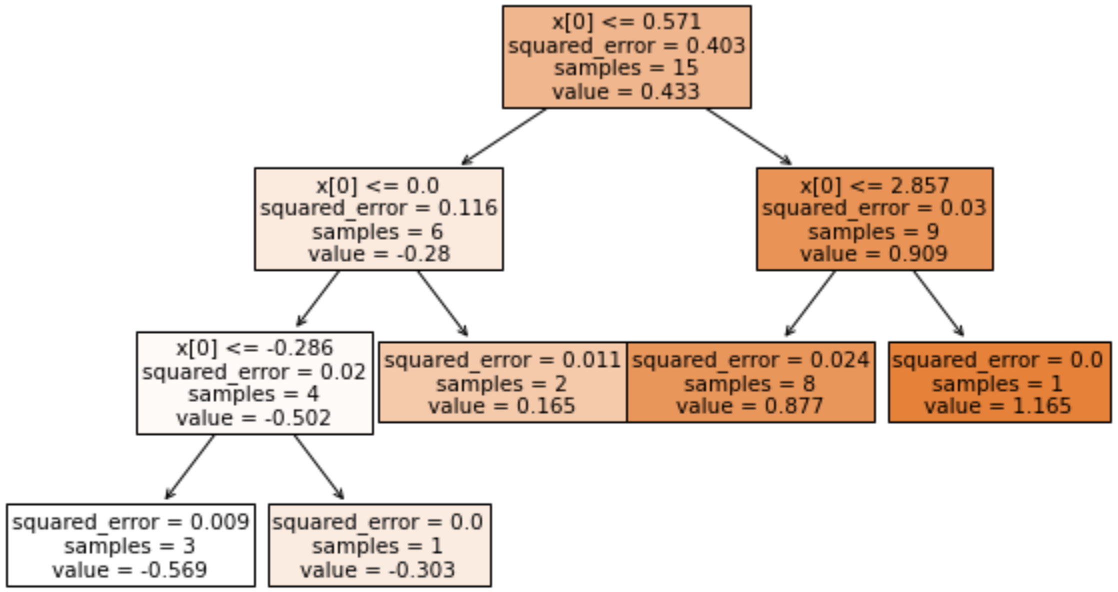
$$E = y - \hat{y}$$

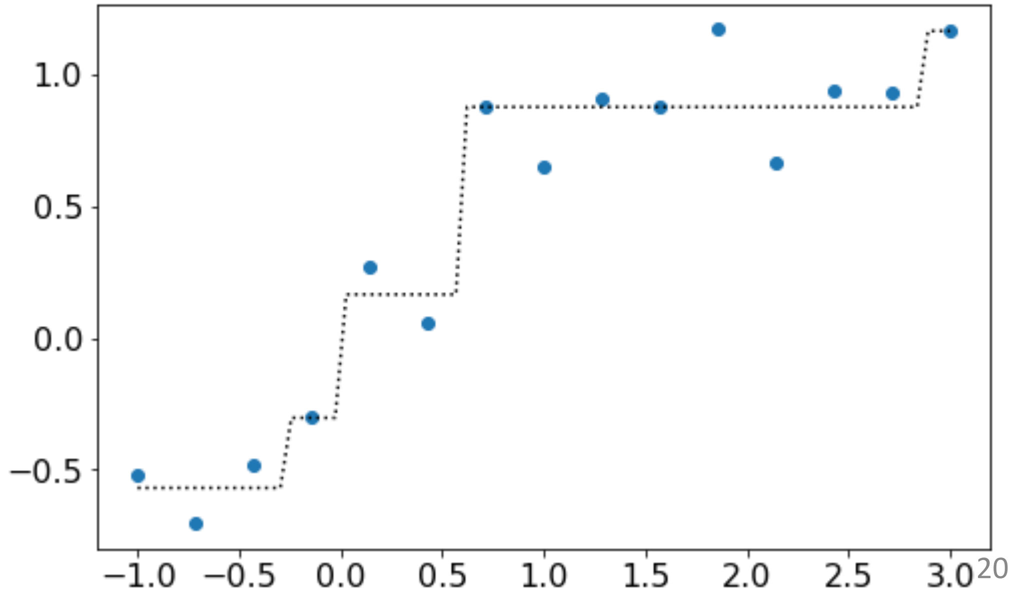
- Fit a new tree to these residuals and add it to the model

$$F_m(x) = F_{m-1}(x) + h_m(x)$$

Solving using Scikit-Learn

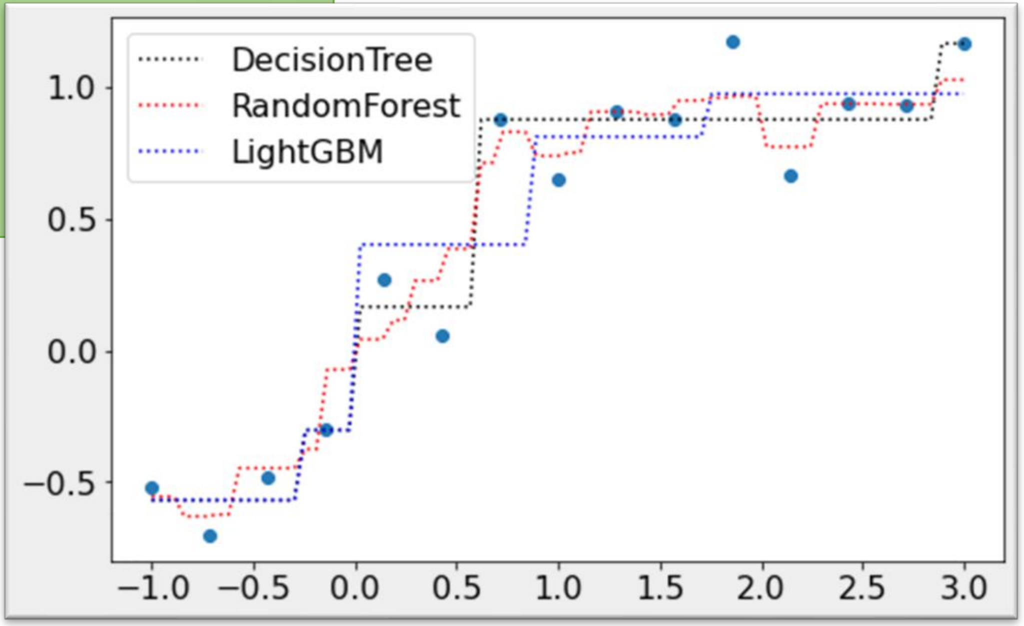
```
from sklearn.tree import DecisionTreeRegressor, plot_tree

tree = DecisionTreeRegressor(max_leaf_nodes=5)
tree.fit(x_vec, y)

plt.figure(figsize=(10,6))
plot_tree(tree, filled=True)
```





Other Solutions

```
from sklearn.ensemble import RandomForestRegressor
forest = RandomForestRegressor(n_estimators=5)
forest.fit(x_vec, y)
y_pred_forest = forest.predict(xr.reshape(-1,1))

from lightgbm import LGBMRegressor
lgbm = LGBMRegressor(min_child_samples=3)
lgbm.fit(x_vec, y)
y_pred_lgbm = lgbm.predict(xr.reshape(-1,1))
```



cardinal → actual space
ordinal → ordered but irregula

Comparison of Methods

Method	Benefits	Drawbacks
Kernel Regression	• Conceptually simple • Easy handling of categorical predictors	• Sensitive to choice of hyperparameters (especially bandwidth)
Local Regression	• Does not require pre-training • Conceptually simple	• Computationally difficult
Support Vector Regression	• Works well in high-dimensional spaces • Tends to generalize well	• Hard to interpret results • Slow to train when using non-linear kernels
Tree-Based ~~Model~~ Regression	• Good performance	• Sensitive to hyperparameters