Unit 8
K-NN and Kernel Methods

EE-UY 4563/EL-GY 9143: INTRODUCTION TO MACHINE LEARNING

Learning Objectives

- ❑ Mathematically describe K-NN and kernel methods
  - For regression and classification
- ❑ Visualize the regression and classification output
- ❑ Select kernel hyper-parameters from cross validation
- ❑ Compute the bias and variance error for estimators

Relation to SVM Lecture

- Unit 8 also has a lecture on SVM
- Beginning 2025, we are not teaching SVM
- Spending more time on Kernel methods and decision trees
- SVMs are not widely-used
- But look at SVM if you are interested

Outline

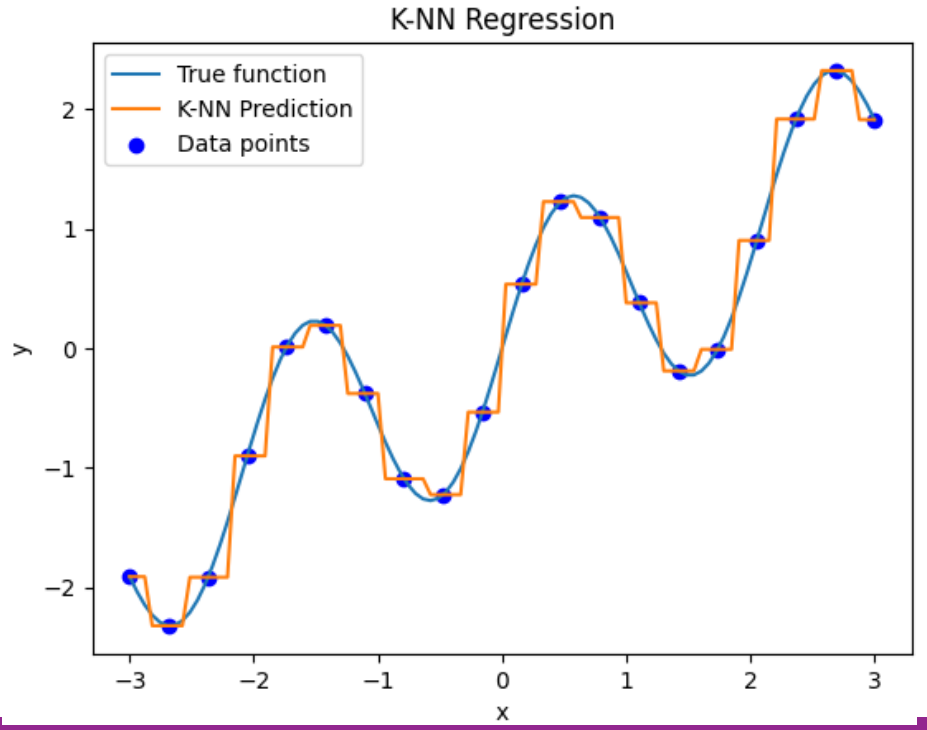
- K-Nearest Neighbors
- Kernel Smoothing
- Kernel Regression
- Example with Climate Data

Nearest Neighbor Regression

- Problem: Given data (x_i, y_i) make prediction for new x_0
- Idea: Similar values of x should have similar y

Nearest neighbor estimation

- Find x_i "closest" to x_0
- Select $\hat{y}_0 = x_i$



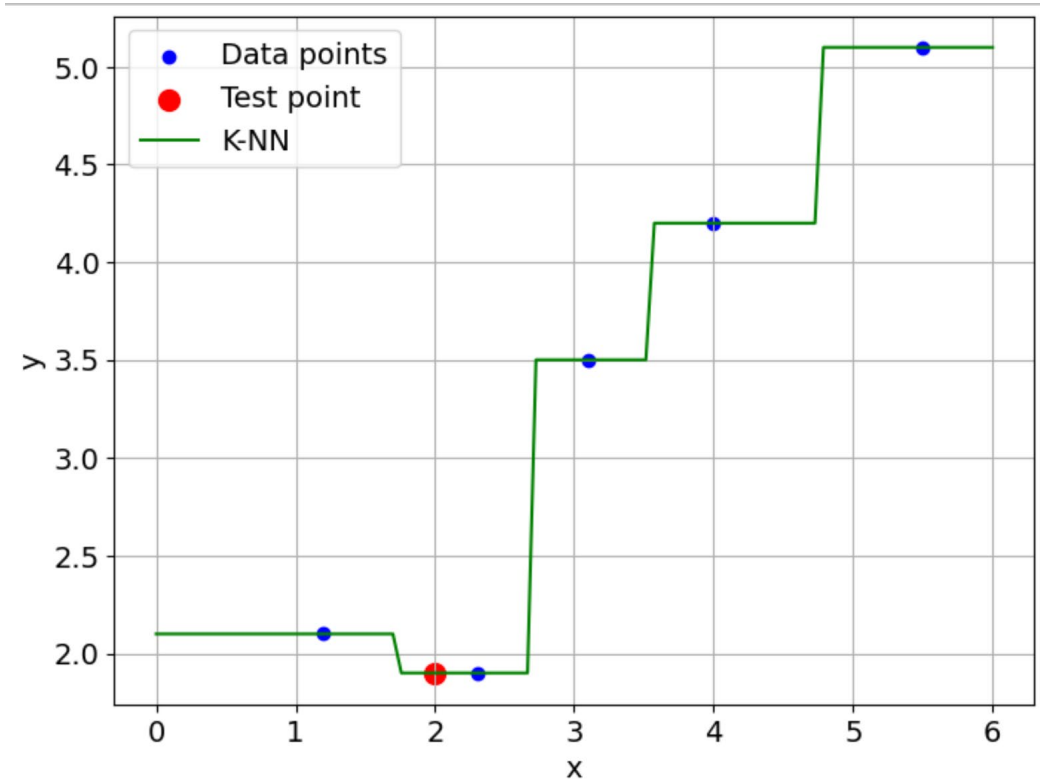
Example

Data

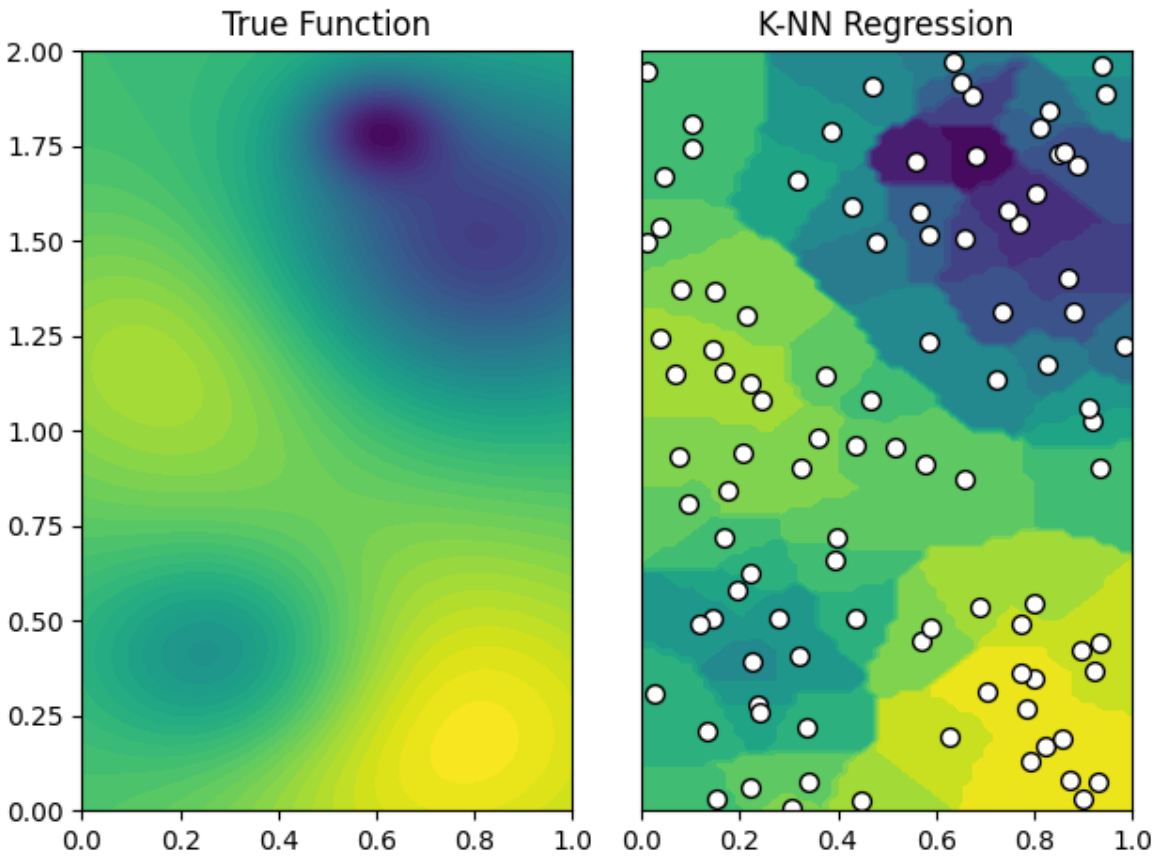
Closest to test point $x_0 = 2$

i	x_i	y_i
1	1.2	2.1
2	2.3	1.9
3	3.1	3.5
4	4.0	4.2
5	5.5	5.1

Prediction $\hat{y}_0 = 1.9$



Nearest Neighbor in 2D



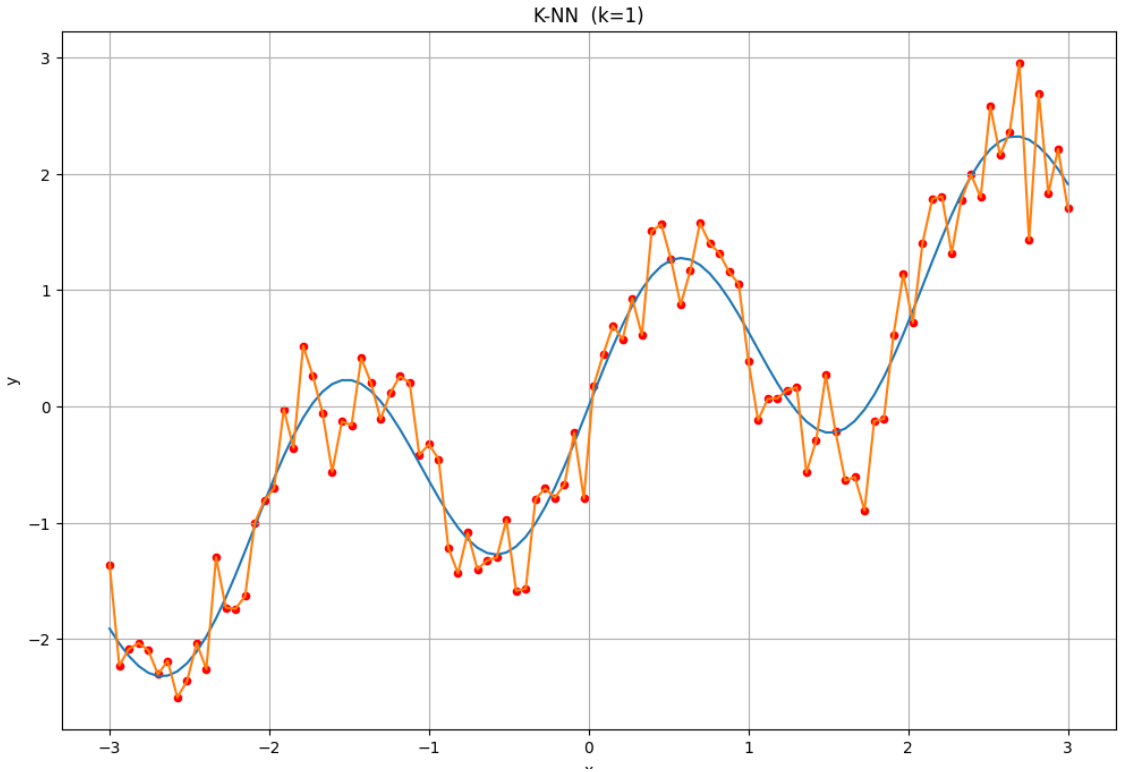
- Gives a “tessellation” of space
- Vornoi regions
  - Each region mapped to a point

K-Nearest Neighbor

- Problem with nearest neighbor:
  - Prediction sensitive to noise
  - Overfits data

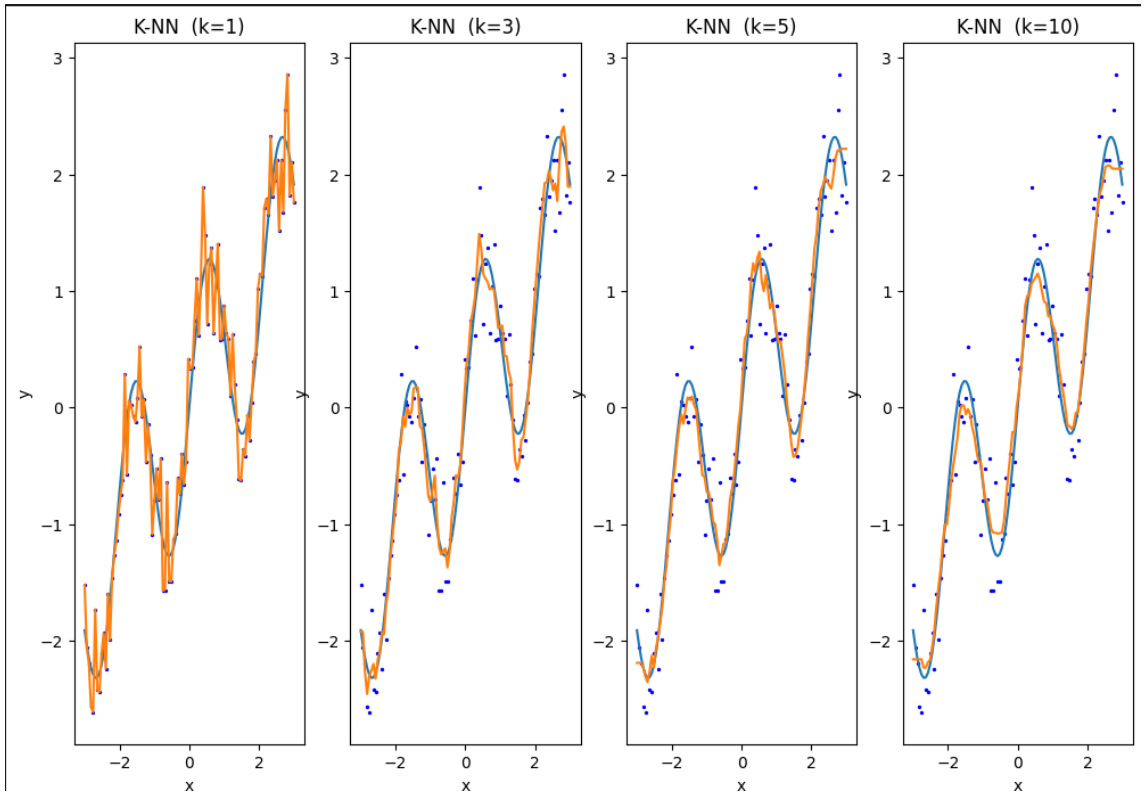
- K-Nearest neighbor:
  - Take average of K closest points

- Increasing K
  - Reduces noise effect (lower variance error)
  - Smooths (increase bias error)



Example of overfitting data

1D Example



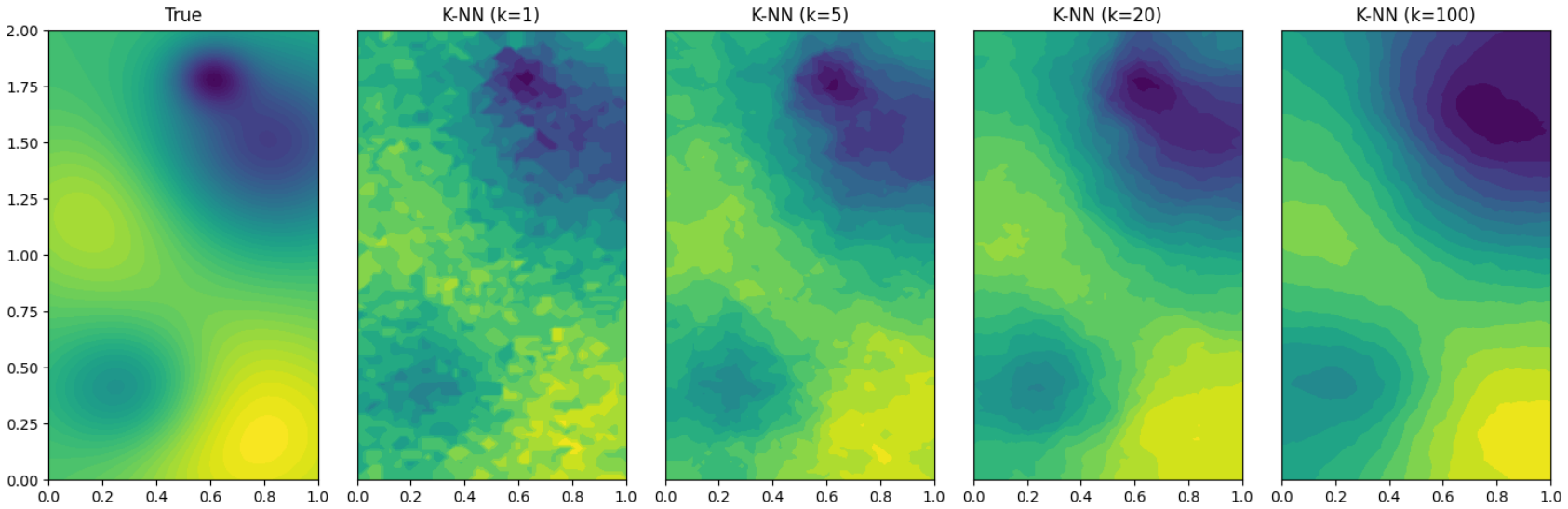
High variance error

Low bias error

Low variance error

High bias error

2D Example



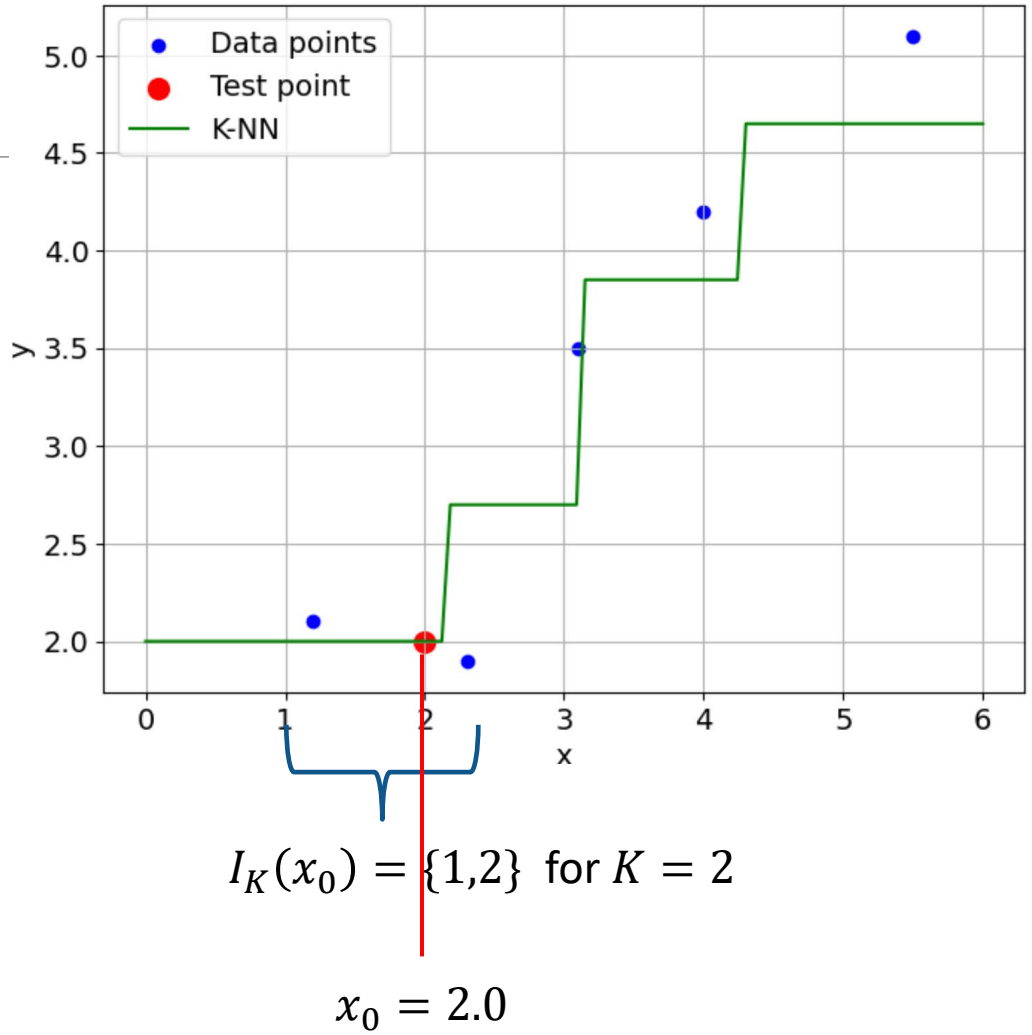
- Simulation details
  - 1000 Data points
  - $y_i = f(x_i) + w_i$
  - $E|w_i|^2 = 0.3^2$

High variance error
Low bias error

Low variance error
High bias error

Error Analysis for K-NN

- Suppose that $y_i = f(x_i) + w_i$,
  - w_i = measurement noise
  - $\mathrm{E}(w_i) = 0,\ E(w_i^2) = \sigma^2$
- Given new test point x_0
- Prediction is:
  - $\hat{y}_0 = \frac{1}{K}\sum_{i \in I_K(x_0)} y_i$,
  - $I_K(x_0) = K$ indices closest to x_0
- What is the bias and variance error?

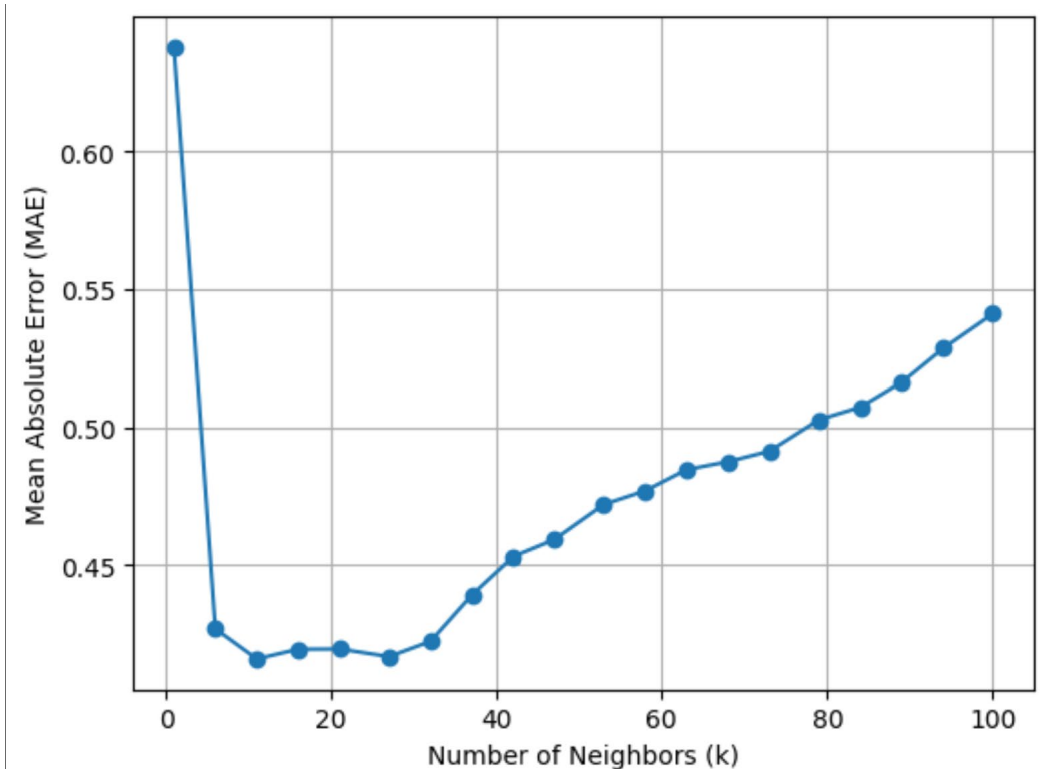


Bias and Variance

- Prediction: $\hat{y}_0 = \frac{1}{K}\sum_{i \in I_K(x_0)} y_i\,,\ y_i = f(x_i) + w_i$
- $E(\hat{y}_0) = \frac{1}{K}\sum_{i \in I_K(x_0)} f(x_i)$
- Bias error: $Bias(x_0) = f(x_0) - \frac{1}{K}\sum_{i \in I_K(x_0)} f(x_i)$
  - Bias will be large when $f(x)$ varies over neighbors
  - Increases with K
- Variance error: $Var(x_0) = E[\hat{y}_0 - E(\hat{y}_0)]^2 = E\left[\frac{1}{K}\sum_{i \in I_K(x_0)} w_i\right]^2 = \frac{\sigma^2}{K}$
  - Decreases with K

Using Cross-Validation

- As usual find optimal K with cross-validation




```
mae = []
for i, k in enumerate(ktest):
    model = KNeighborsRegressor(n_neighbors=k)
    model.fit(xtrain, ytrain)
    yhat = model.predict(xtest)
    err = np.mean(np.abs(yhat - ytest))
    mae.append(err)
    print(f'k={k}, MAE={err:.3f}')
```

Parametric vs. Non-Parametric

Parametric methods

- Ex: Linear regression
- Assume function form
  - Model has parameters
- Fit parameters
- Use model directly

Non-Parametric methods

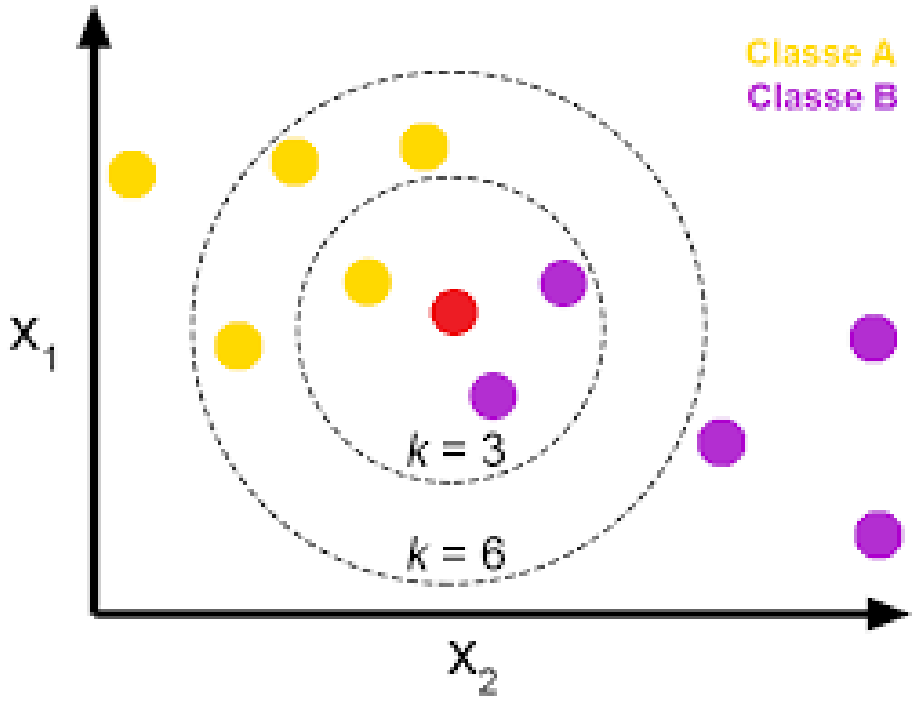
- Ex: K-NN
- No functional assumption
  - No parameters
  - But assumes smoothness
- No fitting
- Use data directly

Computational Cost Comparison

Task	Parametric Method (e.g. linear regression)	Non-Parametric (e.g. K-NN)
Model Fitting	Can be hard (e.g. gradient descent)	None, except hyper-parameters
Inference (after fitting)	Generally easier	Hard Must search through all data

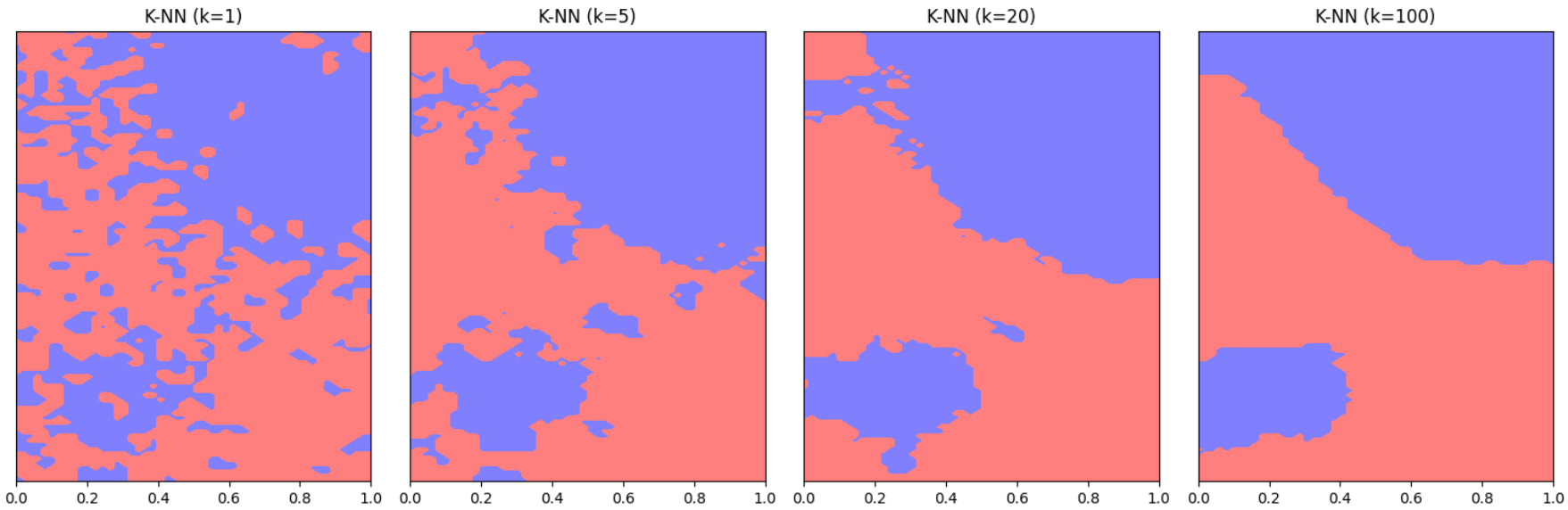
K-NN Classification

- K-NN can also be used for classification
- Data $(\boldsymbol{x}_i, y_i)$ binary labels $y_i \in \{0,1\}$
- Test point x_0
- Find indices of K closest neighbors $I_K(x_0)$
- Predict probability: $p(x_0) = \frac{1}{K}\sum_{i \in I_K(x_0)} y_i$
  - Fraction of neighbors that are class 1
- Class estimate $\hat{y}_0 = \begin{cases} 1 & p(x_0) \geq t \\ 0 & p(x_0) < t \end{cases}$
  - t = threshold. Typically, $t = \frac{1}{2}$



K-NN 2D Classification

- 1000 data points
- Larger K:
  - Smoother decision regions



Outline

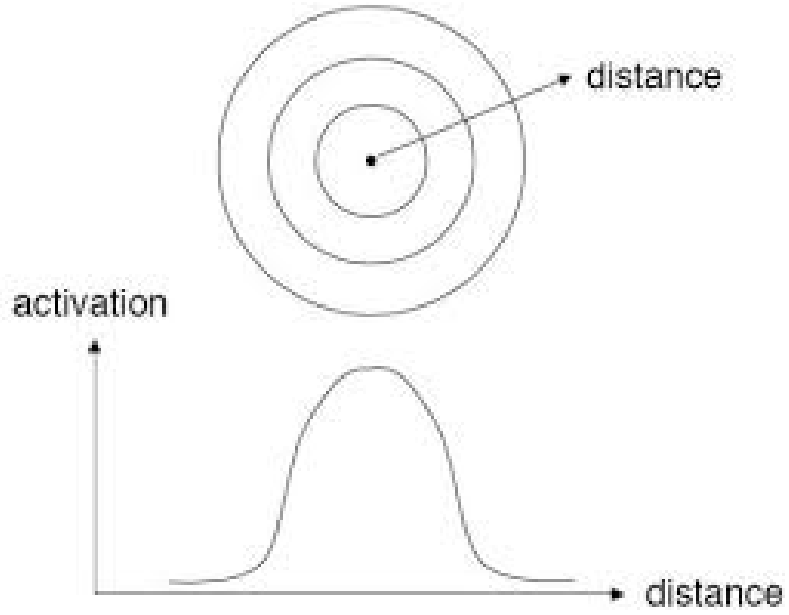
- K-Nearest Neighbors
- Kernel Smoothing
- Kernel Regression
- Example with Climate Data

The Kernel Function

- Problem with K-NN
  - Prediction is not smooth
  - All K points weighted equally

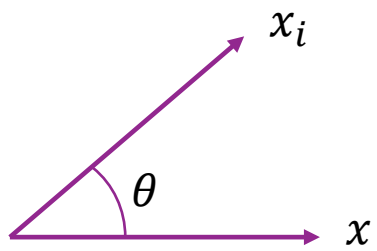
- Kernel function:
  - Function $K(\boldsymbol{x}_i, \boldsymbol{x})$
  - Measures "similarity" between new sample $\boldsymbol{x}$ and trainin

- Typical property
  - $\boldsymbol{x}_i, \boldsymbol{x}$ close $\Rightarrow K(\boldsymbol{x}_i, \boldsymbol{x})$ maximum value
  - $\boldsymbol{x}_i, \boldsymbol{x}$ far $\Rightarrow K(\boldsymbol{x}_i, \boldsymbol{x}) \approx 0$



Common Kernels

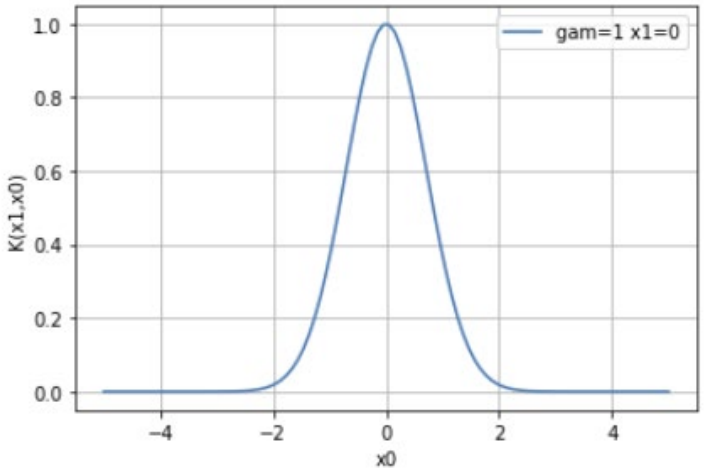
- Linear :
  - $K(x_i, x) = x_i^T x = \|x_i\|\|x\| \cos\theta$
  - Maximum when angle between vectors is small



- Radial basis function:

$$K(x_i, x) = \exp\left[-\gamma\|x - x_i\|^2\right]$$

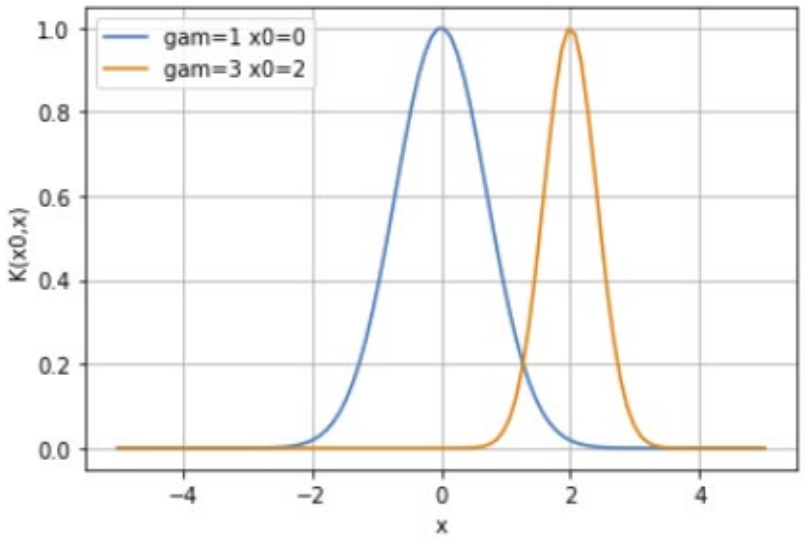
  - $1/\gamma$ indicates width of kernel



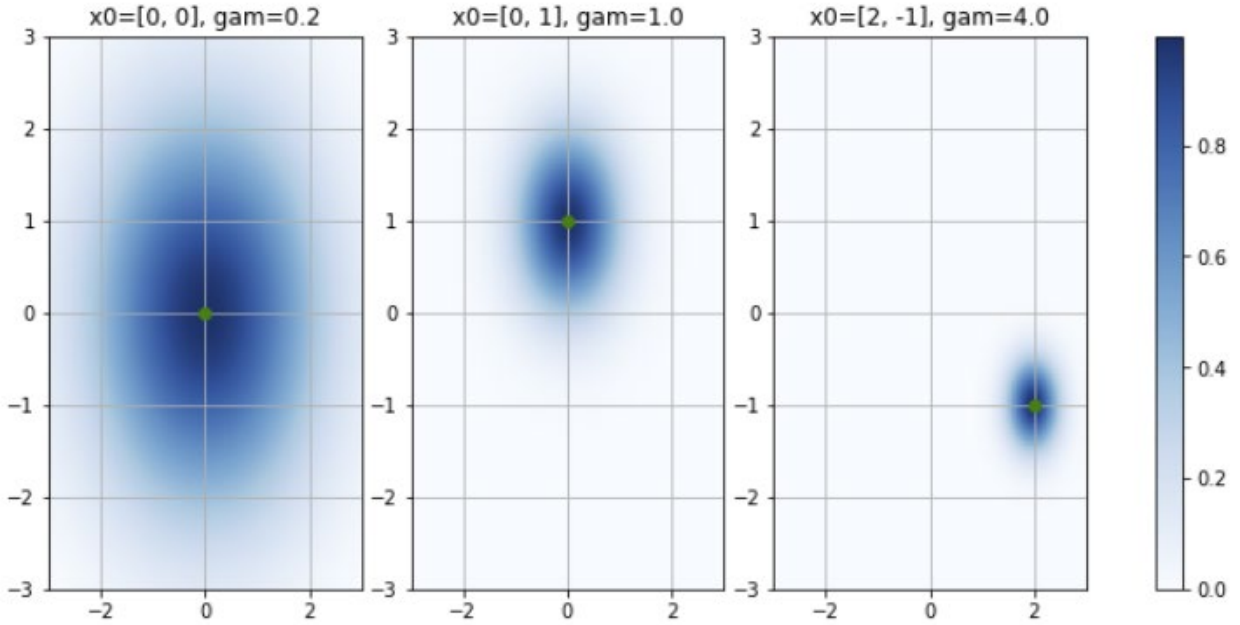
- Polynomial kernel: $K(x_i, x) = \left|x_i^T x\right|^d$
  - Typically d=2

RBF Kernel Examples

- RBF kernel: $K(x_0, x) = \exp\left[-\gamma \|x - x_0\|^2\right]$
  - Peak value of 1 at $x = x_0$
  - Width $\propto \frac{1}{\gamma}$



RBFs in 1D



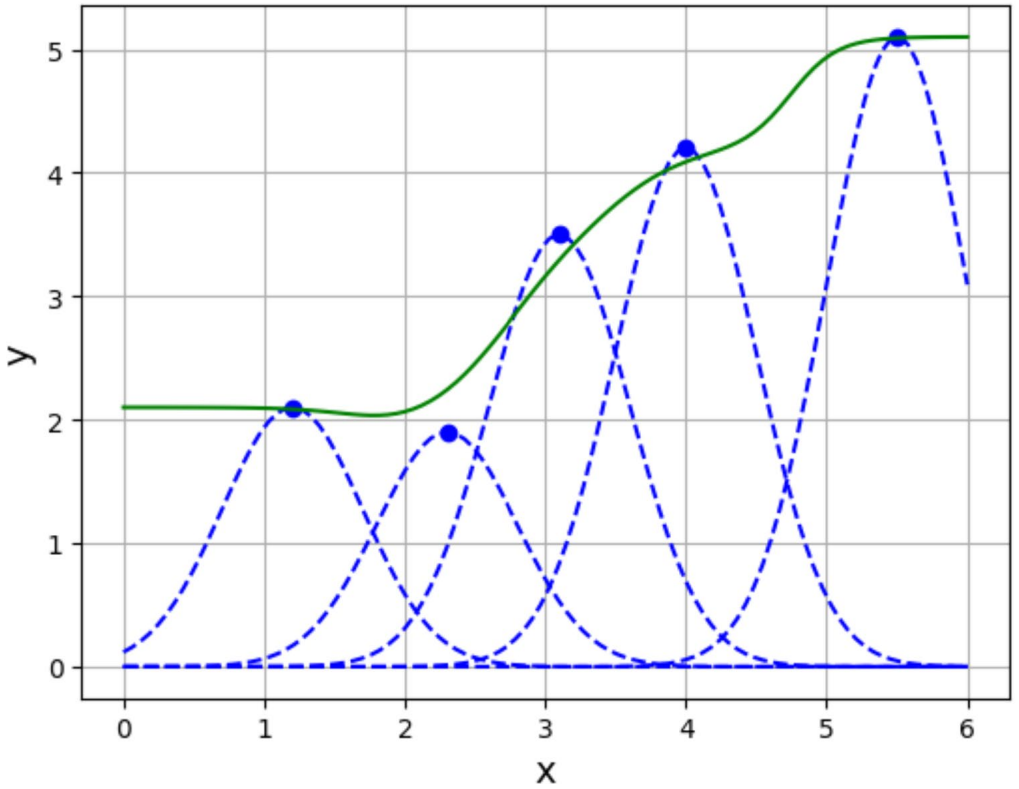
RBFs in 2D

Kernel Smoothing

- Given data $(x_i, y_i), i = 1, \ldots, n$
- Target point x_0
- Compute distance to all data point $K(x_0, x_i)$
- Estimate at x_0

$$\hat{y}_0 = \frac{\sum_i y_i K(x_0, x_i)}{\sum_i K(x_0, x_i)}$$

- Weights each data point by $K(x_0, x_i)$
- Sometimes called Nadaraya-Watson estimator

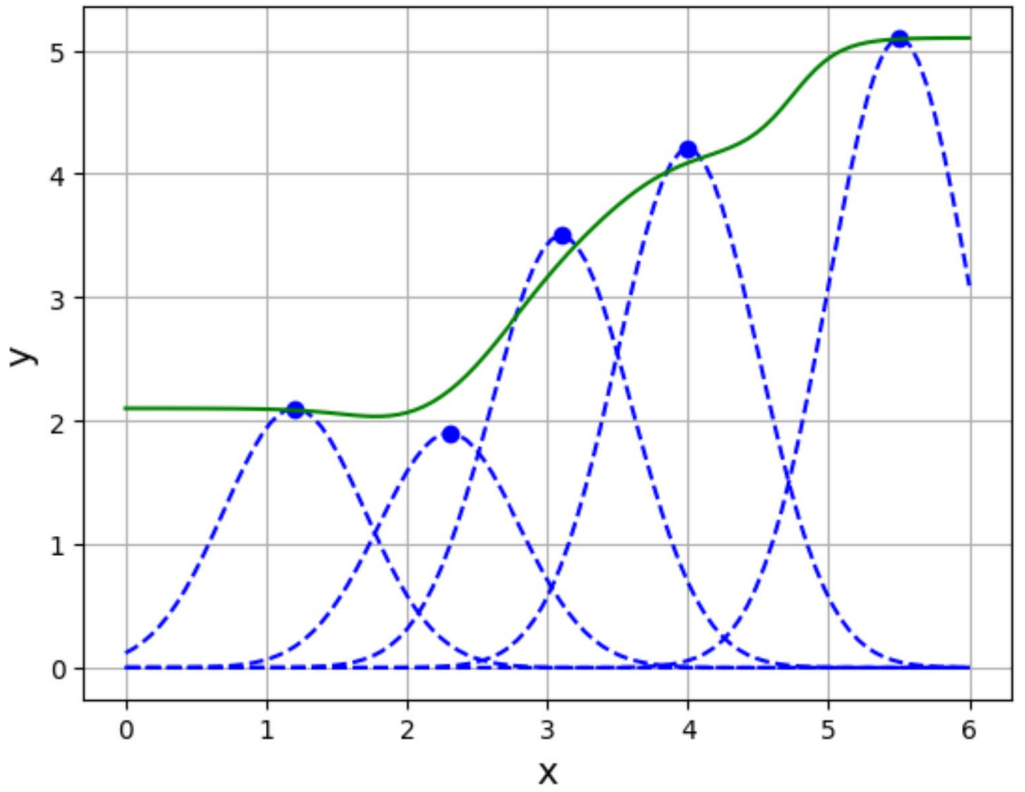


Kernel Smoothing as Weighted Sum

- Kernel smoother at x_0

$$\hat{y}_0 = \frac{\sum_i y_i K(x_0, x_i)}{\sum_i K(x_0, x_i)} = \sum_i y_i \alpha_i$$

- Weights: $\alpha_i = \frac{K(x_0, x_i)}{\sum_j K(x_0, x_j)}$
- Note: $\sum_i \alpha_i = 1, \ \alpha_i \geq 0$
- So, $\hat{y}_0$ is a weighted sum
- Higher weight on points where $K(x_0, x_i)$



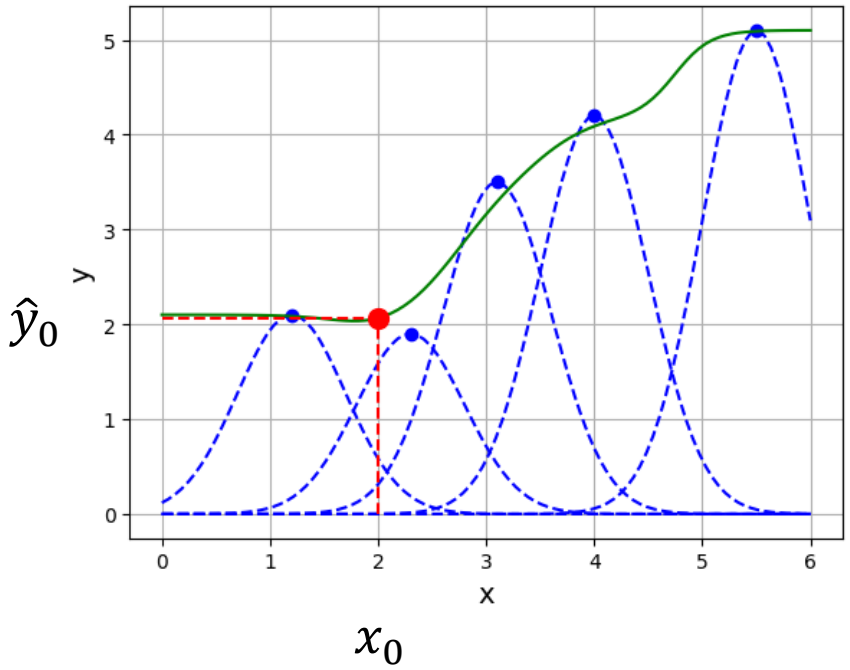
Example with RBF

Test point: $x_0 = 2$

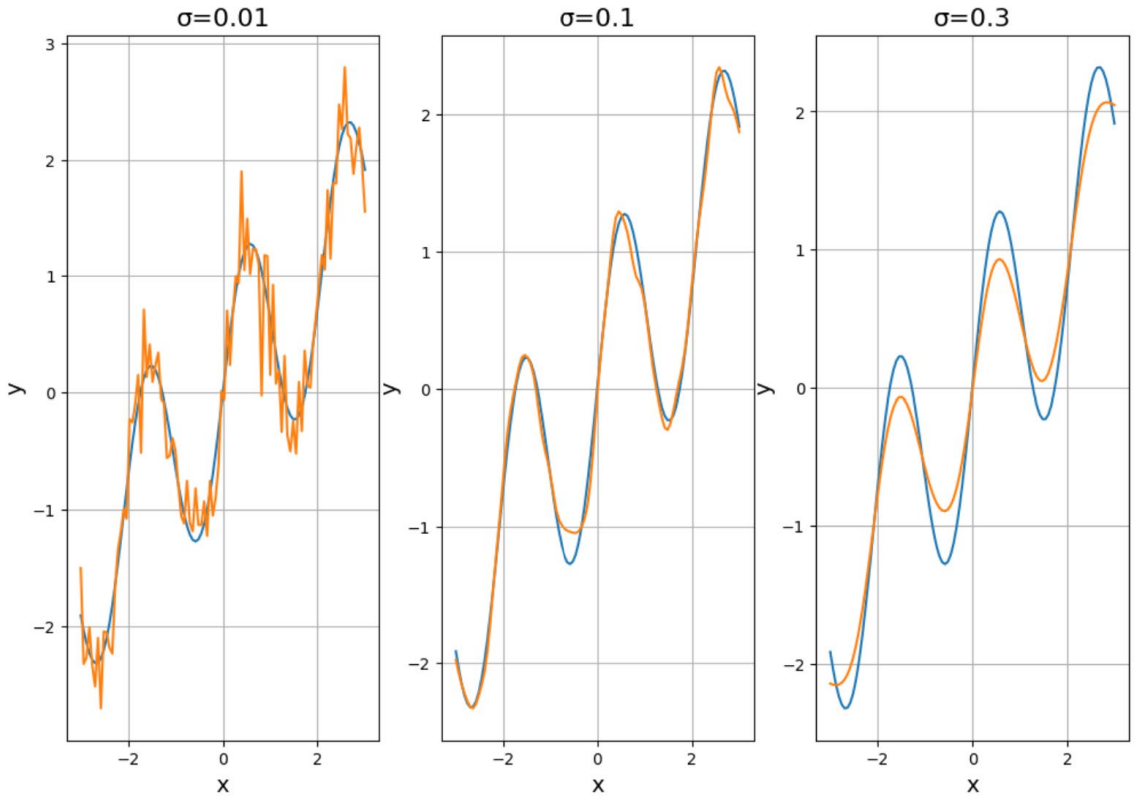
RBF kernel: $\gamma = \frac{1}{\sigma^2}, \sigma = 0.5$

Estimate: $\hat{y}_0 = \sum_{i=1}^{n} \alpha_i x_i \approx 2.06$

i	x_i	y_i	$K(x_0, x_i)$	α_i
1	1.2	2.1	0.27	0.23
2	2.3	1.9	0.84	0.69
3	3.1	3.5	0.089	0.074
4	4.0	4.2	3.3 $(10)^{-4}$	$2.8(10)^{-4}$
5	5.5	5.1	$2.3(10)^{-11}$	$1.9(10)^{-11}$



1D Example



High variance error

Low bias error

Low variance error

High bias error

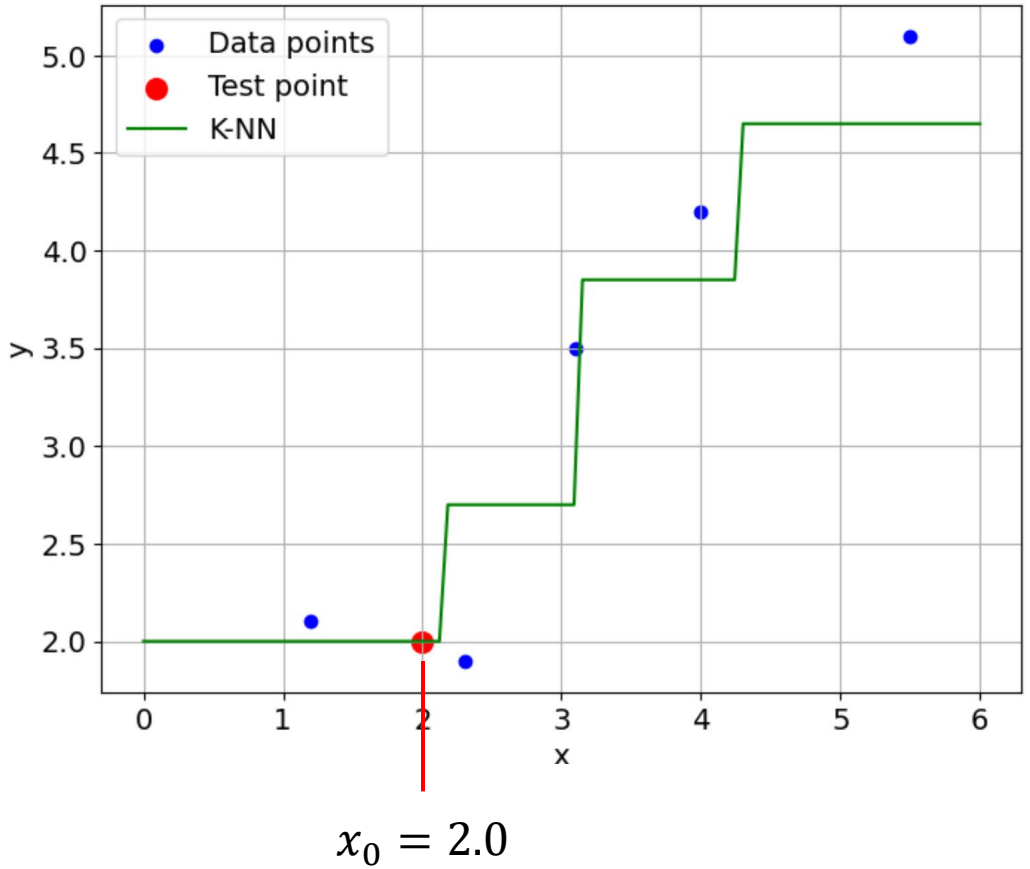
Kernel Smoother in Python

- No built-in function in sklearn
- But you can write it manually

```
def kernel_smoother(xdata, ydata, x, sig, tol=1e-8):
    Dsq = np.sum((xdata[None,:,:] - x[:,None,:])**2, axis=2)
    K = np.exp(-0.5 * Dsq/(sig**2))
    Ksum = K.sum(axis=1, keepdims=True)
    W = K / (Ksum + tol)
    yhat = (W * ydata[None,:]).sum(axis=1)
    return yhat
```

Error Analysis for Kernel

- Suppose that $y_i = f(x_i) + w_i$,
  - w_i = measurement noise
  - $\mathrm{E}(w_i) = 0,\ E(w_i^2) = \sigma^2$
- Given new test point x_0
- Prediction is:
  - $\hat{y}_0 = \sum_i \alpha_i y_i$,
- What is the bias and variance error?



Bias and Variance

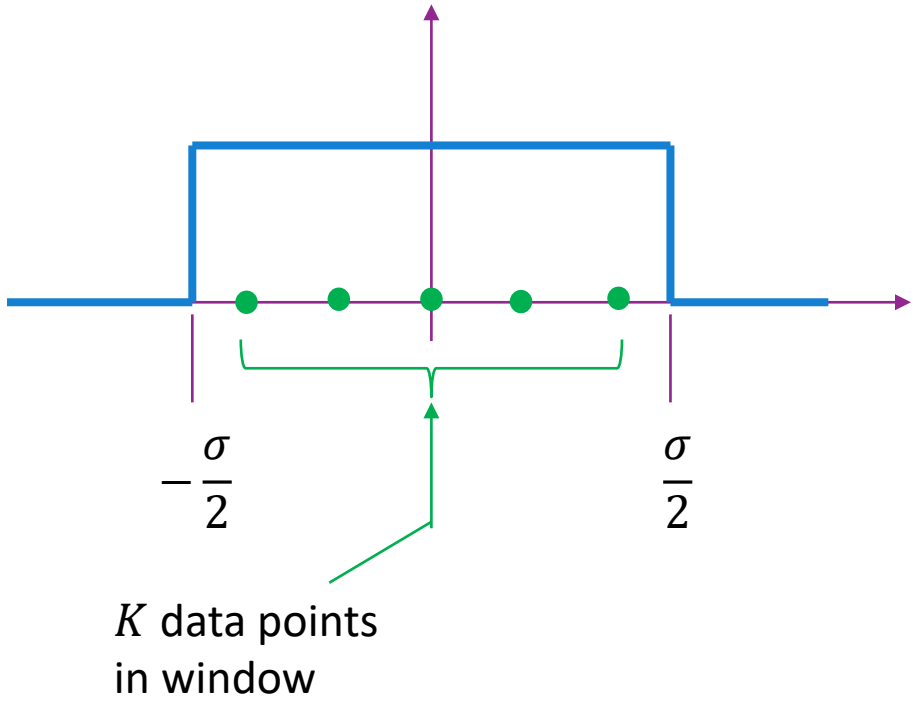
- Prediction: $\hat{y}_0 = \sum_i \alpha_i(x_0) y_i\,,\ y_i = f(x_i) + w_i$
- $E(\hat{y}_0) = \sum \alpha_i(x_0) f(x_i)$
- Bias error: $Bias(x_0) = f(x_0) - \sum \alpha_i(x_0) f(x_i)$
  - Bias will be large when $f(x)$ varies over kernel bandwidth
  - Increases with kernel radius
- Variance error: $Var(x_0) = E[\hat{y}_0 - E(\hat{y}_0)]^2 = E[\sum \alpha_i w_i]^2 = \sigma^2 \sum \alpha_i^2$

Variance Error for a Uniform Kernel

- Example: Uniform kernel

$$K(x, x') = \begin{cases} 1 & \text{if } |x - x'| \le \sigma \\ 0 & \text{else} \end{cases}$$

- Data (x_i, y_i) and test point x_0
- Suppose that M data points $|x_i - x_0| \le \sigma$
- Then: $\alpha_i(x_0) = \frac{K(x_0, x_i)}{\sum K(x_0, x_i)} = \frac{1}{M}$
- Variance is $Var(x_0) = \sigma^2 \sum \alpha_i^2 = \frac{\sigma^2 M}{M^2} = \frac{\sigma^2}{M}$
- Variance decreases with window size



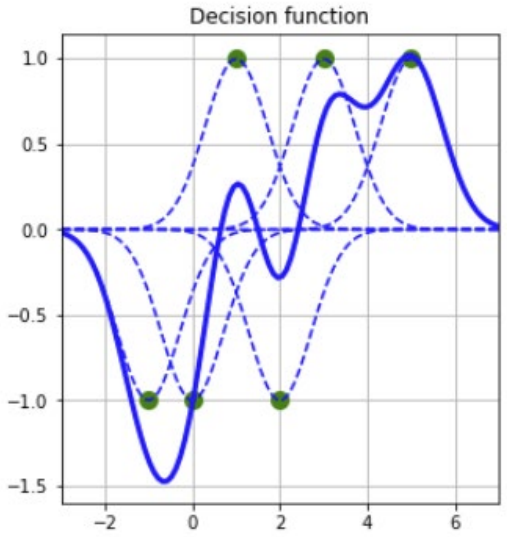
Kernel Classifier

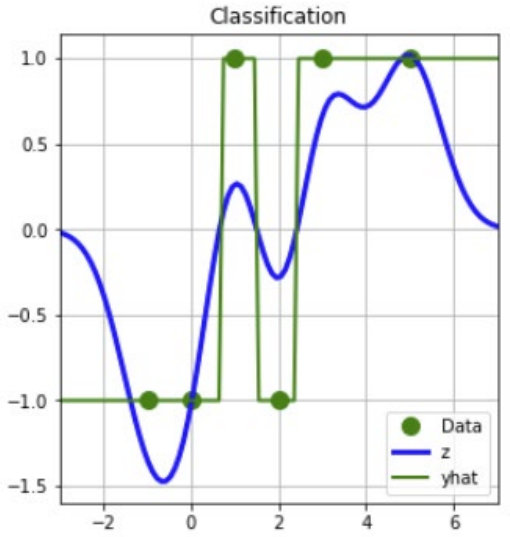
- Kernel can also be used for classification
- Given:
  - Training data (x_i, y_i) with binary labels $y_i = \pm 1$
  - Kernel $K(x_i, x)$
- To classify a new point x:
  - Decision function: $z = \sum_{i=1}^{n} y_i K(x_i, x)$
  - Classify: $\hat{y} = sign(z)$
- Idea:
  - z is large positive when x is close to samples x_i with $y_i = 1$
  - z is large negative when x is close to samples x_i with $y_i = -1$

Example in 1D

- Example data with 6 points (x_i, y_i)
  - RBF kernel: $K(x_i, x) = e^{-\gamma(x_i - x)^2}$, $\gamma = 1$
- Decision function:
  - $z = \sum_{i=1}^{n} y_i K(x_i, x)$
  - Sum of bell curves
  - Positive when near positive samples
  - Negative when near negative samples
- Classification:
  - $\hat{y} = sign(z)$

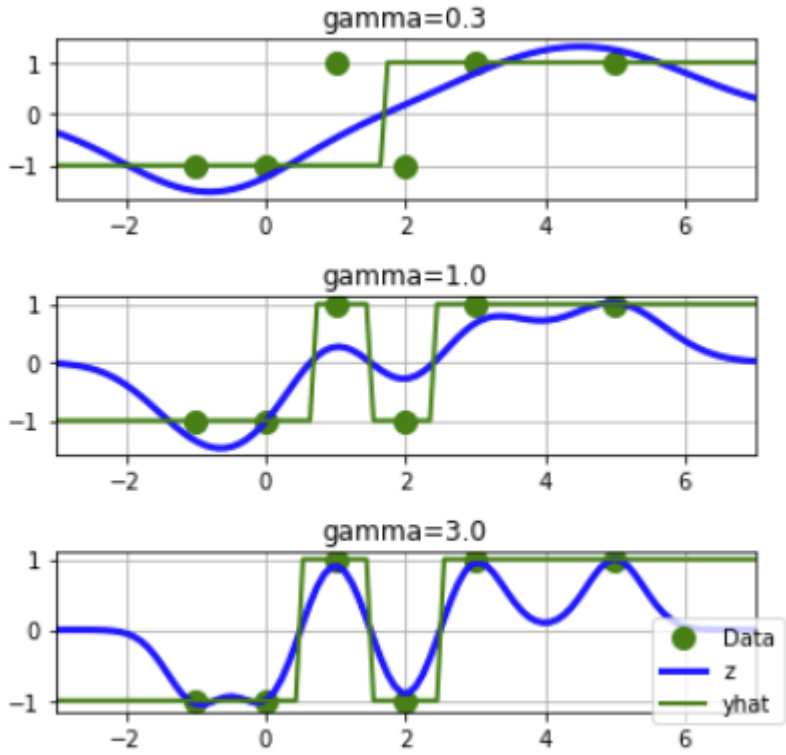
i	1	2	3	4	5	6
x_i	-1	0	1	2	3	5
y_i	-1	-1	1	-1	1	1





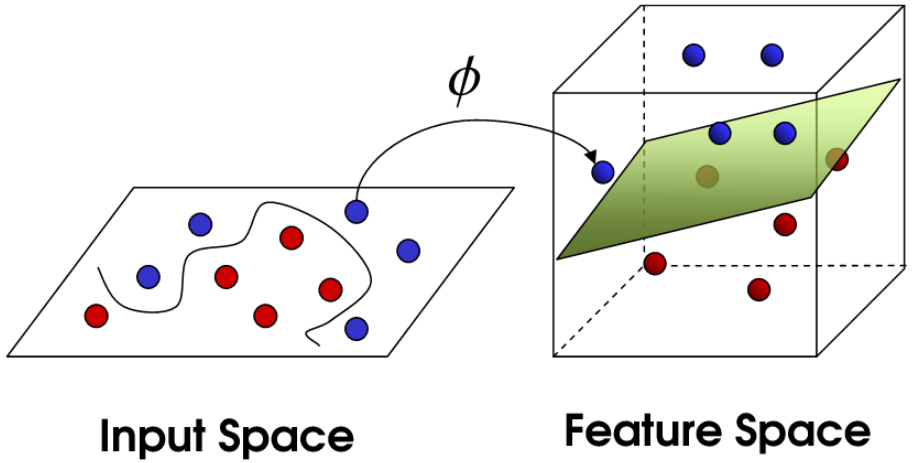
Effect of Gamma

- Same data as before
- RBF kernel: $K(x_i, x) = e^{-\gamma(x_i - x)^2}$
- As γ increases:
  - Decision function $z \approx y_i$ when $x = x_i$
  - Classifier fits training data better
  - Classification region more complex
- As a classifier, higher γ results in:
  - Lower bias error
  - But, higher variance error



Transform Linear Models

- Recall transforms
- Data $(\boldsymbol{x}_i, y_i)$
- Transform data $z_{ij} = \phi_j(x_i), j = 1, \ldots, p$
  - Each $\phi_j(\boldsymbol{x})$ is a "basis" function
- Linear model: $\hat{y} = \sum_{j=1}^{p} w_j \phi_j(\boldsymbol{x})$
- Weights w_j found by linear regression
- Maps original data $\boldsymbol{x}$ to a higher dimensional space $\boldsymbol{z}$



Inner Product Kernel

- Consider a transform $\mathbf{z} = \phi(\boldsymbol{x})$
- Define kernel $K(\boldsymbol{x}, \boldsymbol{x}') = \phi(\boldsymbol{x})^T \phi(\boldsymbol{x}')$
  - Inner product in the transformed space
- $K(\boldsymbol{x}, \boldsymbol{x}')$ has three key properties:
- Positive: $K(\boldsymbol{x}, \boldsymbol{x}) \geq 0$ since $K(\boldsymbol{x}, \boldsymbol{x}) = \|\phi(\boldsymbol{x})\|^2$
- Symmetry: $K(\boldsymbol{x}, \boldsymbol{x}') = \phi(\boldsymbol{x})^T \phi(\boldsymbol{x}') = \phi(\boldsymbol{x}')^T \phi(\boldsymbol{x}) = K(\boldsymbol{x}', \boldsymbol{x})$
- Positive semi-definite property: Given any α_i and $\boldsymbol{x}_i$ $\sum_{i,j} \alpha_i \alpha_j K(\boldsymbol{x}_i, \boldsymbol{x}_j) \geq 0$
  - Why? $\sum_{i,j} \alpha_i \alpha_j K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \sum_{i,j} \alpha_i \alpha_j \phi(\boldsymbol{x}_i)^T \phi\left(\boldsymbol{x}_j\right) = \left\| \sum_j \alpha_j \phi\left(\boldsymbol{x}_j\right) \right\|^2$

“Kernel Trick”

- Consider transformed linear model: $\hat{y} = \hat{f}(\boldsymbol{x}) = \sum_{j=1}^{p} w_j \phi_j(\boldsymbol{x})$
- Suppose w solved with Ridge regression:

$$w = \arg\min_{\mathrm{w}} \sum_{i} \left(y_i - \hat{f}(\boldsymbol{x}_i)\right)^2 + \lambda \|\boldsymbol{w}\|^2$$

- Theorem: For any transformed linear model:

$$\hat{y} = \hat{f}(\boldsymbol{x}) = \sum_{i=1}^{n} \beta_i K(\boldsymbol{x}_i, \boldsymbol{x})$$

  - Coefficients are solutions to $(\boldsymbol{K} + \lambda \boldsymbol{I})\boldsymbol{\beta} = \boldsymbol{y}$
  - $\boldsymbol{K}$ is the matrix with coefficients $K_{ij} = K(\boldsymbol{x}_i, \boldsymbol{x}_j)$

Implications of the Kernel Trick

- In any transformed model: $\hat{y} = \hat{f}(\boldsymbol{x}) = \sum_{i=1}^{n} \beta_i K(\boldsymbol{x}_i, \boldsymbol{x})$
- Similar to Kernel smoothing
  - But parameters β are fit from the data
- No need to transform data $\boldsymbol{x} \mapsto \phi(\boldsymbol{x})$
- Can solve for coefficients directly with Kernel $K(\boldsymbol{x}_i, \boldsymbol{x})$

Proof of Kernel Trick

- Weights are given by: $w = \left(A^T A + \lambda I\right)^{-1} A^T y$ with $A = \begin{bmatrix} \phi(x_1)^T \\ \vdots \\ \phi(x_n)^T \end{bmatrix}$
- From linear algebra $w = A^T \left(AA^T + \lambda I\right)^{-1} y$
- Let $\beta = \left(AA^T + \lambda I\right)^{-1} y$ so $w = A^T \beta$
- For any other data point x, $\hat{y} = \hat{f}(x) = w^T \phi(x) = \beta^T A^T \phi(x)$
- By definition of the kernel:
  - $\left[AA^T\right]_{ij} = \phi(x_i)^T \phi(x_j) = K(x_i, x_j)$ and $\left[A^T \phi(x)\right]_i = K(x_i, x)$
- Hence: $\hat{y} = \hat{f}(x) = w^T \phi(x) = \beta^T A^T \phi(x) = \sum \beta_i K(x_i, x)$
- Also: $(K + \lambda I)\beta = y$

Kernel Representation (Advanced)

- Say a kernel $K(\boldsymbol{x}, \boldsymbol{x}')$ is symmetric and positive semidefinite if:
  - Symmetry: $K(\boldsymbol{x}, \boldsymbol{x}') = K(\boldsymbol{x}', \boldsymbol{x})$
  - Given any α_i and $\boldsymbol{x}_i$ $\quad \sum_{i,j} \alpha_i \alpha_j K(\boldsymbol{x}_i, \boldsymbol{x}_j) \geq 0$
- Sometimes people say positive semidefinite to include symmetric
- Theorem: If kernel $K(\boldsymbol{x}, \boldsymbol{x}')$ is symmetric and positive semidefinite then there exists a mapping $\boldsymbol{x} \mapsto \phi(\boldsymbol{x})$ to a Hilbert space such that

$$K(\boldsymbol{x}, \boldsymbol{x}') = \langle \phi(\boldsymbol{x}), \phi(\boldsymbol{x}') \rangle$$

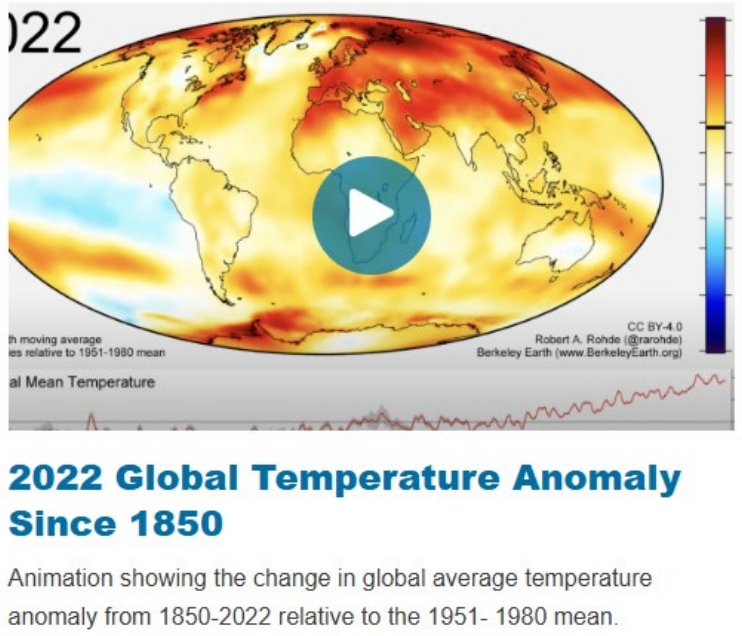
- All kernels are inner products in some transformed space
- The space may be infinite dimensional

Outline

- ❑ K-Nearest Neighbors
- ❑ Kernel Smoothing
- ❑ Kernel Regression
- ❑ Example with Climate Data

Temperature Modeling

- Problem: Interpolate temperature from limited measurements
- Data has natural spatial structure
- Kernel methods apply well
- Data from excellent Berkeley Earth page
- Several other sources as well



Loading the Data

- Get data from data webpage
- Can manually download file
- Or use downloader in jupyter notebook
- File is NetCDF (.nc) file
  - Used for multi-dimensional data

```
<xarray.Dataset> Size: 548MB
Dimensions:      (latitude: 180, longitude: 360, time: 2100, month_number: 12)
Coordinates:
  * longitude    (longitude) float32 1kB -179.5 -178.5 -177.5 ... 178.5 179.5
  * latitude     (latitude) float32 720B -89.5 -88.5 -87.5 ... 87.5 88.5 89.5
  * time         (time) float64 17kB 1.85e+03 1.85e+03 ... 2.025e+03 2.025e+03
Dimensions without coordinates: month_number
Data variables:
    land_mask    (latitude, longitude) float64 518kB ...
    temperature  (time, latitude, longitude) float32 544MB ...
    climatology  (month_number, latitude, longitude) float32 3MB ...
Attributes:
    Conventions:           Berkeley Earth Internal Convention (based on CF-1.5)
    title:                 Native Format Berkeley Earth Surface Temperature A...
    history:               09-Jan-2025 20:35:17
    institution:           Berkeley Earth Surface Temperature Project
    land_source_history:   04-Jan-2025 19:11:11
    ocean_source_history:  06-Jan-2025 11:47:59
    comment:               This file contains Berkeley Earth surface temperat...
```

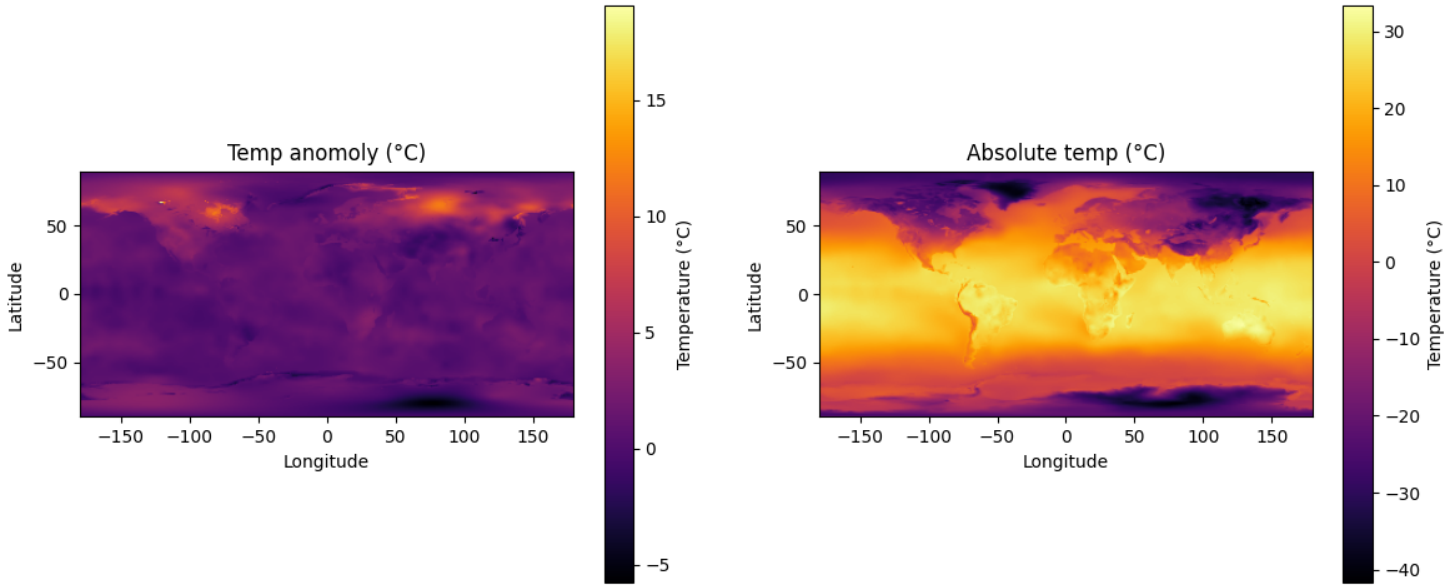
Global Gridded Data

Datasets are also provided in a gridded NetCDF format. Two types of grids are provided, a grid based o a latitude-longitude grid. The equal area grid is the primary data format used in most of our analyses an may be less convenient for many users.

Datasets marked as "Experimental" below are products that are under development have not peer revie give us feedback.

- Global Monthly Land + Ocean
  - Average Temperature with Air Temperatures at Sea Ice (Recommended; 1850 – Recent)
    - Video of Temperature Field
    - Equal Area (~100 MB)
    - 1° x 1° Latitude-Longitude Grid (~400 MB)

Visualize the Temperature



- Berkeley data has climate
  - Over time and space
- We look at one “time slice”
- Temperature anomaly:
  - Difference from average monthly temp
- Temperature anomaly used for fitting
  - Removes known variation

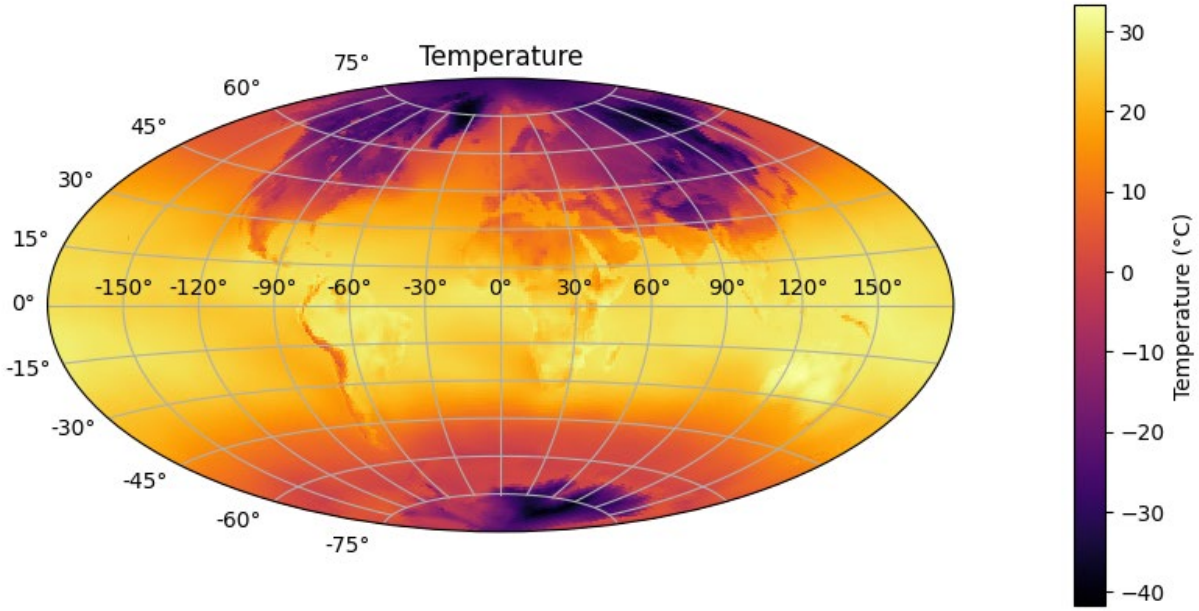
Projections

- You can also view via azimuthal map projection
- Example: Aitoff

```
# Convert coordinates to radians
lon_rad = np.deg2rad(lon)  # shift to [-180, 180] then to radians
lat_rad = np.deg2rad(lat)

# Create meshgrid
lon_grid, lat_grid = np.meshgrid(lon_rad, lat_rad)

fig = plt.figure(figsize=(10, 5))
ax = fig.add_subplot(111, projection='aitoff')
im = ax.pcolormesh(lon_grid, lat_grid, temp_abs, cmap='inferno', shading='auto')
ax.grid(True)
plt.title('Temperature')
plt.colorbar(im, label='Temperature (°C)', pad=0.1)
```



Interpolation with K-NN

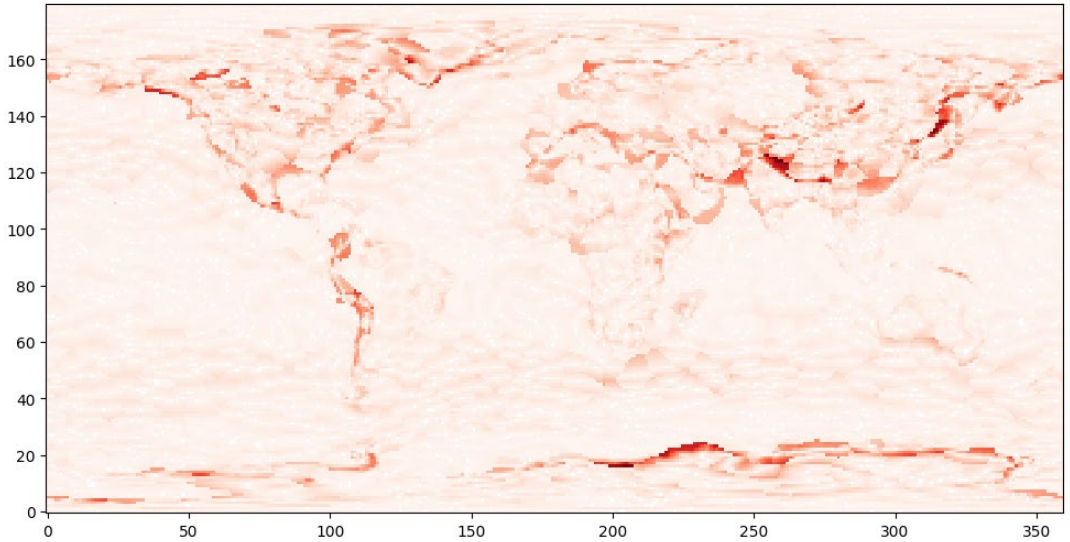
- Take 3% of the data for training
- Interpolate to remaining 90%
- Use (x, y, z) instead of lat-long for distance
  - Ensures distance is measured on sphere

```
# Convert the lat/long to Cartesian coordinates for distance calculation
# We use a unit sphere (R=1) for simplicity
R = 1
x = R * np.cos(np.deg2rad(lat_flat)) * np.cos(np.deg2rad(lon_flat))
y = R * np.cos(np.deg2rad(lat_flat)) * np.sin(np.deg2rad(lon_flat))
z = R * np.sin(np.deg2rad(lat_flat))

# Select a random subset of points for training
p = 0.03  # fraction of points to use for training
num_points = len(temp_flat)
num_train = int(p * num_points)
np.random.seed(0)  # for reproducibility
train_indices = np.random.choice(num_points, num_train, replace=False)
test_indices = np.setdiff1d(np.arange(num_points), train_indices)
X_train = np.vstack((x[train_indices], y[train_indices], z[train_indices])).T
y_train = temp_flat[train_indices]
X_test = np.vstack((x[test_indices], y[test_indices], z[test_indices])).T
y_test = temp_flat[test_indices]
```

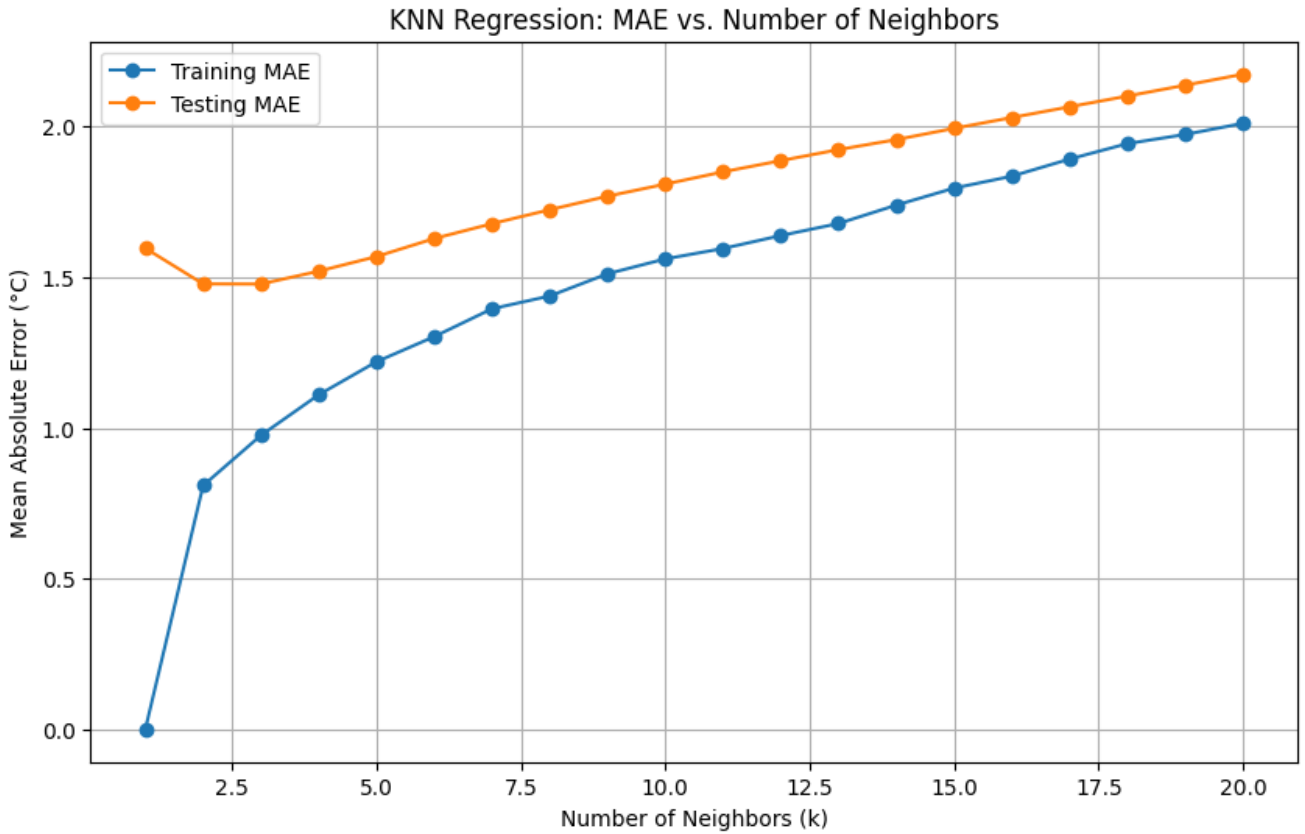
Interpolation with K-NN

- Error with $K = 1$
- MAE = $1.6°$
- Error map to the right
  - Errors largest at mountains and coastlines
  - Locations of rapid variation

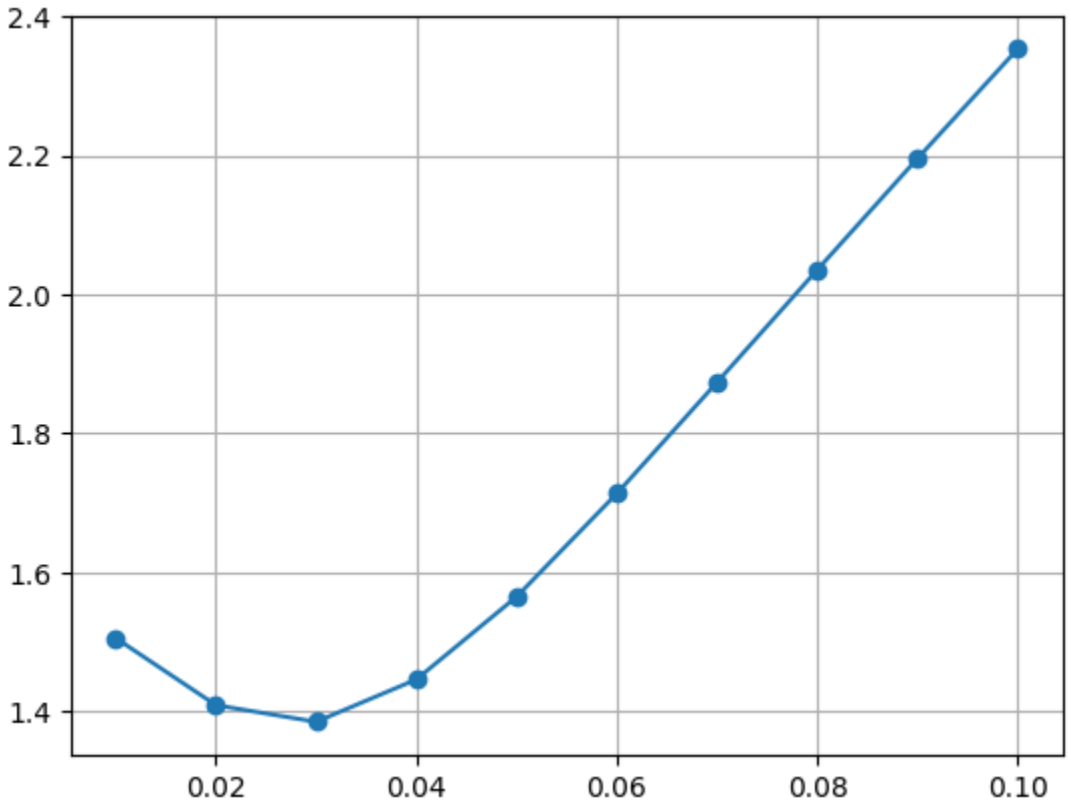


Selecting K

- $K = 3$ is optimal



Kernel Smoother



- Compute MAE vs. bandwidth
- Performs slightly better than K-NN