Unit 8
Support Vector Machines

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

PROF. SUNDEEP RANGAN, WITH MODIFICATION BY YAO WANG

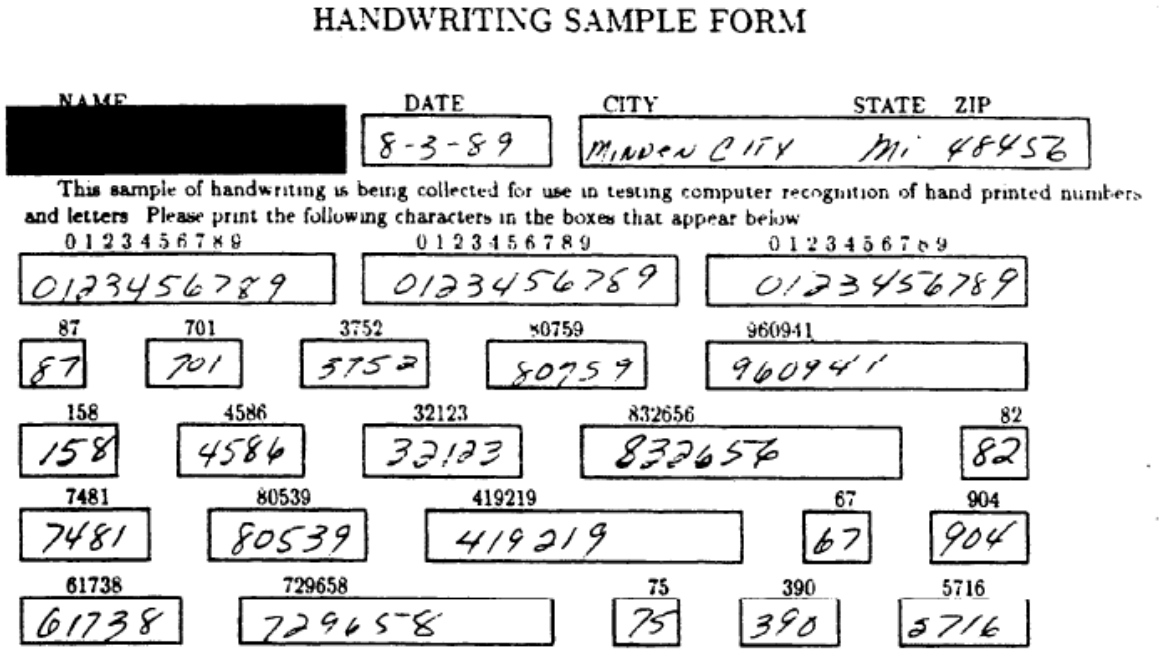
Learning Objectives

- ❑ Interpret weights in linear classification of images
- ❑ Describe why linear classification for images does not work
- ❑ Define the margin in linear classification
- ❑ Describe the SVM classification problem.
- ❑ Write equations for solutions of constrained optimization using the Lagrangian.
- ❑ Describe a kernel SVM problem for non-linear classification
- ❑ Implement SVM classifiers in python
- ❑ Select SVM parameters from cross-validation

Outline

- Motivating example: Recognizing handwritten digits
  - Why logistic regression doesn't work well.
- Maximum margin classifiers
- Support vector machines
- Kernel trick
- Constrained optimization

MNIST Digit Classification



HANDWRITING SAMPLE FORM

NAME [redacted] DATE 8-3-89 CITY MINDEN CITY STATE MI ZIP 48456

This sample of handwriting is being collected for use in testing computer recognition of hand printed numbers and letters. Please print the following characters in the boxes that appear below

0 1 2 3 4 5 6 7 8 9 | 0 1 2 3 4 5 6 7 8 9 | 0 1 2 3 4 5 6 7 8 9

0123456789 | 0123456789 | 0123456789

87 | 701 | 3752 | 80759 | 960941

158 | 4586 | 32123 | 832656 | 82

7481 | 80539 | 419219 | 67 | 904

61738 | 729658 | 75 | 390 | 5716

- Problem: Recognize hand-written digits
- Original problem:
  - Census forms
  - Automated processing
- Classic machine learning problem
- Benchmark

From Patrick J. Grother, NIST Special Database, 1995

A Widely-Used Benchmark

- We will look at SVM today
- Not the best algorithm
- But quite good
- …and illustrates the main points

Classifiers [edit]

This is a table of some of the machine learning methods used on the database and their error rates, by type of classifier:

Type	Classifier	Distortion	Preprocessing	Error rate (%)
Linear classifier	Pairwise linear classifier	None	Deskewing	7.6[9]
K-Nearest Neighbors	K-NN with non-linear deformation (P2DHMDM)	None	Shiftable edges	0.52[14]
Boosted Stumps	Product of stumps on Haar features	None	Haar features	0.87[15]
Non-Linear Classifier	40 PCA + quadratic classifier	None	None	3.3[9]
Support vector machine	Virtual SVM, deg-9 poly, 2-pixel jittered	None	Deskewing	0.56[16]
Neural network	2-layer 784-800-10	None	None	1.6[17]
Neural network	2-layer 784-800-10	elastic distortions	None	0.7[17]
Deep neural network	6-layer 784-2500-2000-1500-1000-500-10	elastic distortions	None	0.35[18]
Convolutional neural network	Committee of 35 conv. net, 1-20-P-40-P-150-10	elastic distortions	Width normalizations	0.23[8]

Downloading MNIST

```
import tensorflow as tf

(Xtr,ytr),(Xts,yts) = tf.keras.datasets.mnist.load_data()

print('Xtr shape: %s' % str(Xtr.shape))
print('Xts shape: %s' % str(Xts.shape))

ntr = Xtr.shape[0]
nts = Xts.shape[0]
nrow = Xtr.shape[1]
ncol = Xtr.shape[2]
```

```
Xtr shape: (60000, 28, 28)
Xts shape: (10000, 28, 28)
```

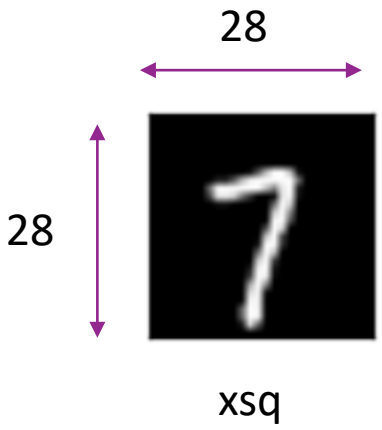
- MNIST data is available in many sources
  - Note: It has been removed from sklearn
- Tensorflow version:
  - 60000 training samples
  - 10000 test samples
- Each sample is a 28 x 28 images
- Grayscale: Pixel values $\in \{0,1,\ldots,255\}$
  - 0 = Black and
  - 255 = White

Matrix and Vector Representation

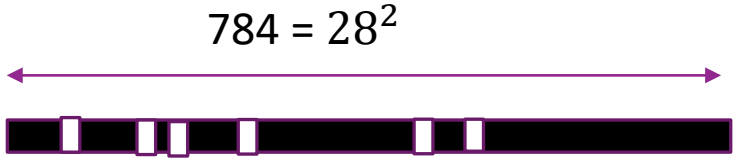
- For this demo, we reshape data from $N \times 28 \times 28$ to $N \times 784$
- But, you can easily go back and forth
- Also, scale the pixel values from -1 to 1

```
npix = nrow*ncol
Xtr = 2*(Xtr/255 - 0.5)
Xtr = Xtr.reshape((ntr,npix))

Xts = 2*(Xts/255 - 0.5)
Xts = Xts.reshape((nts,npix))
```




```
x = xsq.ravel()
```




```
xsq = x.reshape((28,28))
```

$$S = \mathrm{Mat}(x) = \begin{bmatrix} s_{11} & \cdots & s_{1,28} \\ \vdots & \vdots & \vdots \\ s_{28,1} & \cdots & s_{28,28} \end{bmatrix}$$

$$x = \mathrm{vec}(S) = [x_1 \quad \cdots \quad x_{784}]$$

Displaying Images in Python

4 random images in the dataset

A human can classify these easily

```
def plt_digit(x):
    nrow = 28
    ncol = 28
    xsq = x.reshape((nrow,ncol))
    plt.imshow(xsq,  cmap='Greys_r')
    plt.xticks([])
    plt.yticks([])

# Convert data to a matrix
X = mnist.data
y = mnist.target

# Select random digits
nplt = 4
nsamp = X.shape[0]
Iperm = np.random.permutation(nsamp)

# Plot the images using the subplot command
for i in range(nplt):
    ind = Iperm[i]
    plt.subplot(1,nplt,i+1)
    plt_digit(X[ind,:])
```

Key command

Sample permutation is necessary for this dataset, as the original data is ordered by digits

Try a Logistic Classifier

```
ntr1 = 5000
Xtr1 = Xtr[Iperm[:ntr1],:]
ytr1 = ytr[Iperm[:ntr1]]
```

- Train on 5000 samples
  - To reduce training time.
  - In practice want to train with ~40k
- Select correct solver (lbfgs)
  - Others can be very slow. Even this will take minutes

```
from sklearn import linear_model
logreg = linear_model.LogisticRegression(verbose=10, solver='lbfgs',\
                                         multi_class='multinomial',max_iter=500)
logreg.fit(Xtr1,ytr1)
```

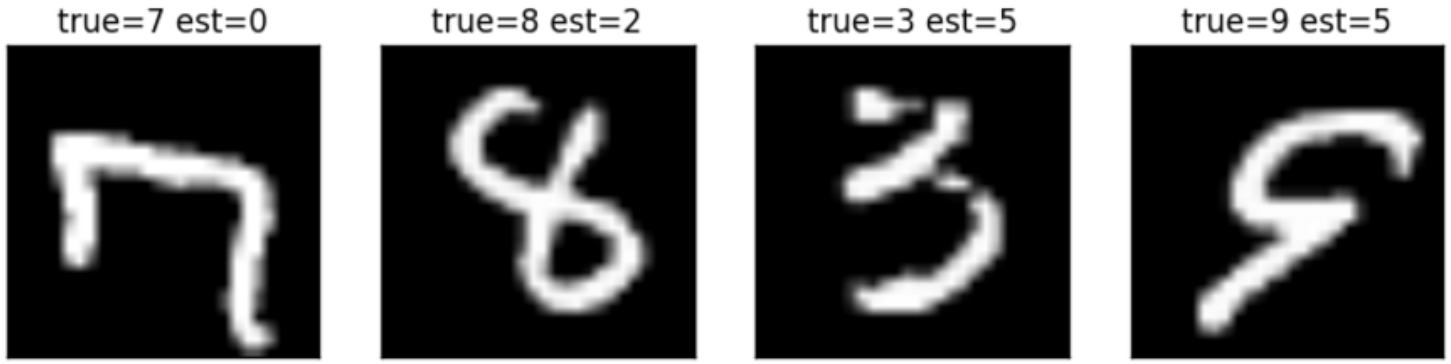
```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:758:
e. Increase the number of iterations.
  "of iterations.", ConvergenceWarning)
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:  1.3min remaining:    0.0s
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:  1.3min finished
```

Performance

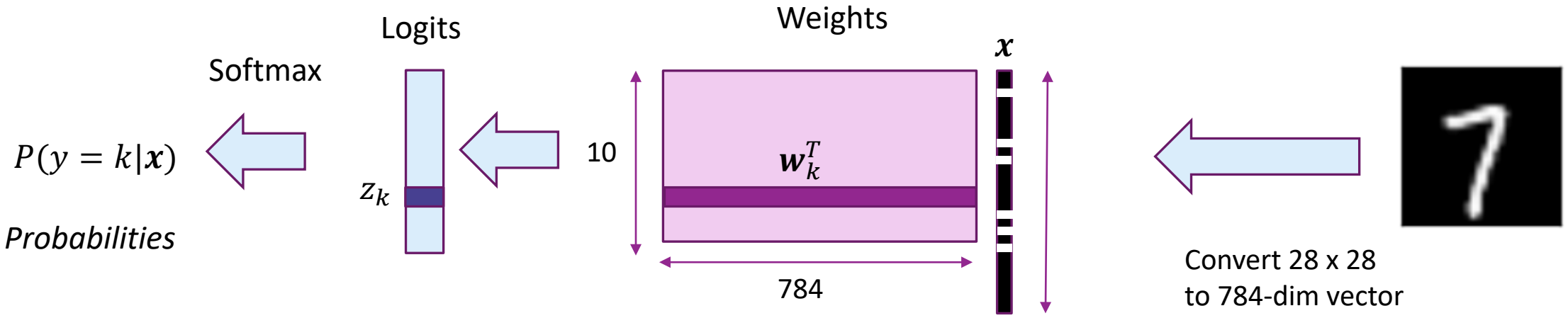
- Accuracy = 89%. Very bad
- Some of the errors seem like they should have been easy to spot
- What went wrong?

```
nts1 = 5000
Iperm_ts = np.random.permutation(nts)
Xts1 = Xts[Iperm_ts[:nts1],:]
yts1 = yts[Iperm_ts[:nts1]]
yhat = logreg.predict(Xts1)
acc = np.mean(yhat == yts1)
print('Accuaracy = {0:f}'.format(acc))
```

```
Accuaracy = 0.891000
```



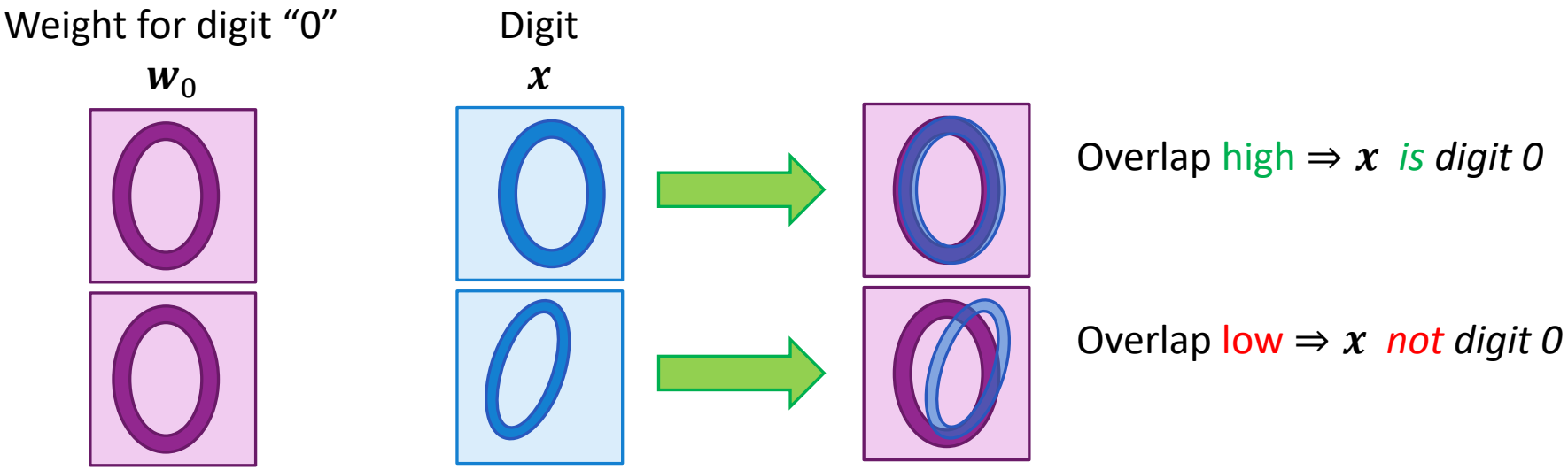
Recap: Logistic Classifier



- Each logit $z_k = \boldsymbol{w}_k^T \boldsymbol{x}$ = inner product with weight $\boldsymbol{w}_k$ with digit $\boldsymbol{x}$, $k = 0, \ldots, 9$
- Will select $\hat{y} = \arg\max_k P(y = k|x) = \arg\max_k z_k$
  - Output z_k which is largest
- When is z_k large?

Interpreting the Logistic Classifier Weights

- A logit $z_k = \boldsymbol{w}_k^T \boldsymbol{x}$ is high when there is high overlap between $\boldsymbol{w}_k$ with digit $\boldsymbol{x}$
  - Visualize each weight as an image
  - Suppose pixels are 0 or 1
  - $z_k = \boldsymbol{w}_k^T \boldsymbol{x} = \sum_i w_{ki} x_i$ = number of pixels that overlap with $\boldsymbol{w}_k$ and $\boldsymbol{x}$
- Conclusion: Small variations in digits can cause low overlap

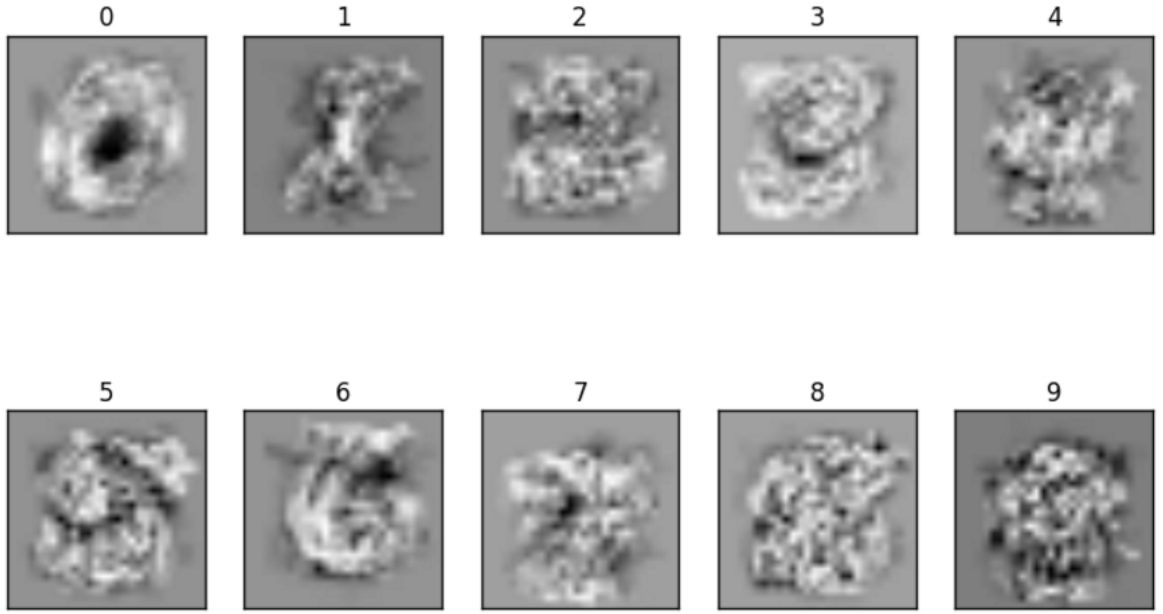


Example with Actual Digits

- Take weight $\boldsymbol{w}$ from a random digit "2"
- Inner products $z = \boldsymbol{w}^T \boldsymbol{x}$ are only slightly higher for other digits "2"
- Cannot tell which digit is correct from the inner product $z = \boldsymbol{w}^T \boldsymbol{x}$

Visualizing the Weights

- Optimized weights of the classifier
- Blurry versions of image to try to capture rotations, translations, …



Problems with Logistic Classifier

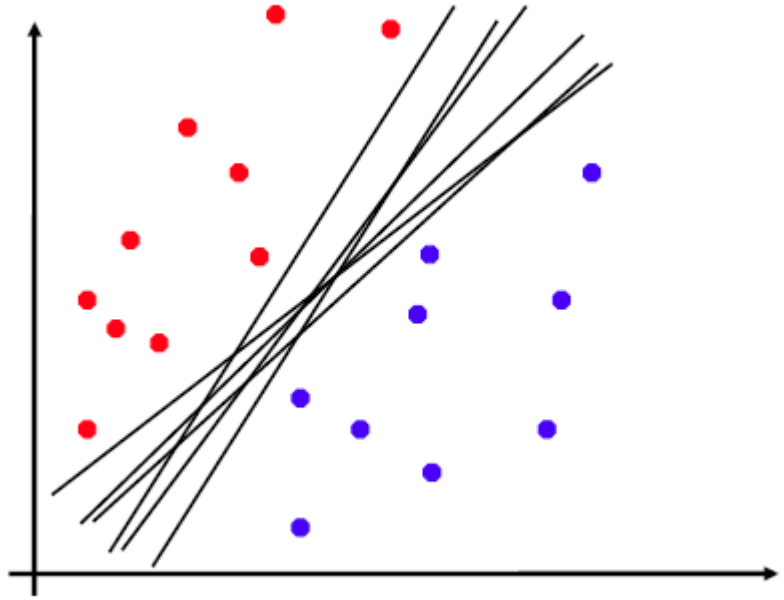
- Linear weighting cannot capture many deformities in image
  - Rotations
  - Translations
  - Variations in relative size of digit components
- Can be improved with preprocessing
  - E.g. deskewing, contrast normalization, many methods
- Is there a better classifier?

Outline

- Motivating example: Recognizing handwritten digits
  - Why logistic regression doesn't work well.
- Maximum margin classifiers
- Support vector machines
- Kernel trick
- Constrained optimization

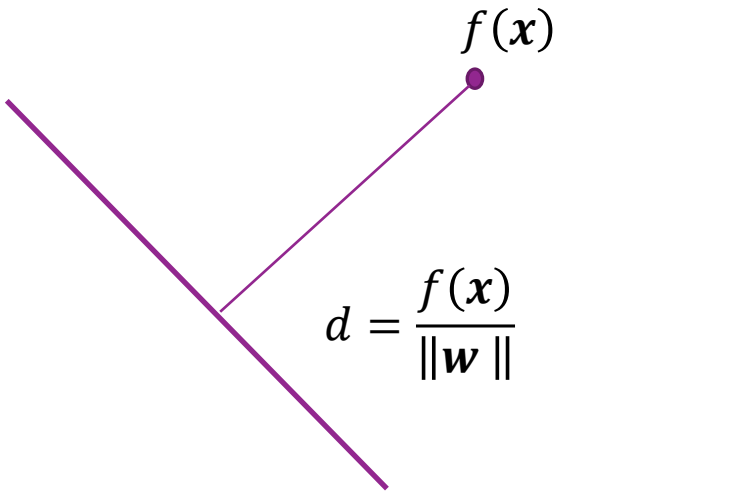
Non-Uniqueness of Separating Plane

- Linearly separable data:
  - Can find a separating hyper-plane as a linear classifier.
- Separating hyper-plane is not unique
  - Fig. on right: Many separating planes
- Which one is optimal?



Hyperplane Basics

- Linear function: $f(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x} + b, \boldsymbol{x} \in R^d$
- Hyperplane in d-dimensional: $f(\boldsymbol{x}) = 0$
- Parameters:
  - Weight $\boldsymbol{w}$ and bias b
  - Unique up to scaling:
  - $(b, \boldsymbol{w})$ and $(\alpha b, \alpha \boldsymbol{w})$ define the same plane.
  - For unique definition, we can require $\|\boldsymbol{w}\|$=1.
- Distance of any point **x** to the hyperplane:
  - $d = f(\boldsymbol{x})/\|\boldsymbol{w}\|$, where $f(\boldsymbol{x}) = b + \boldsymbol{w}^T\boldsymbol{x}$.
  - See ESL Sec. 4.5.
  - ESL: Hastie, Tibshirani, Friedman, "The Elements of Statistical Learning". 2nd Ed. Springer.

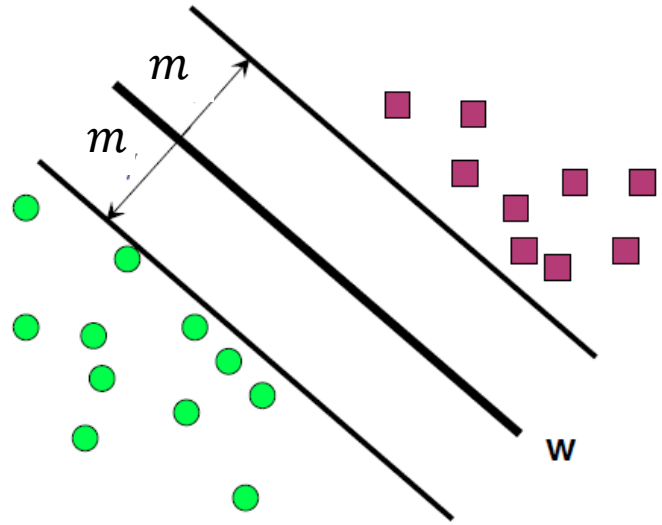


Linear Separability and Margin

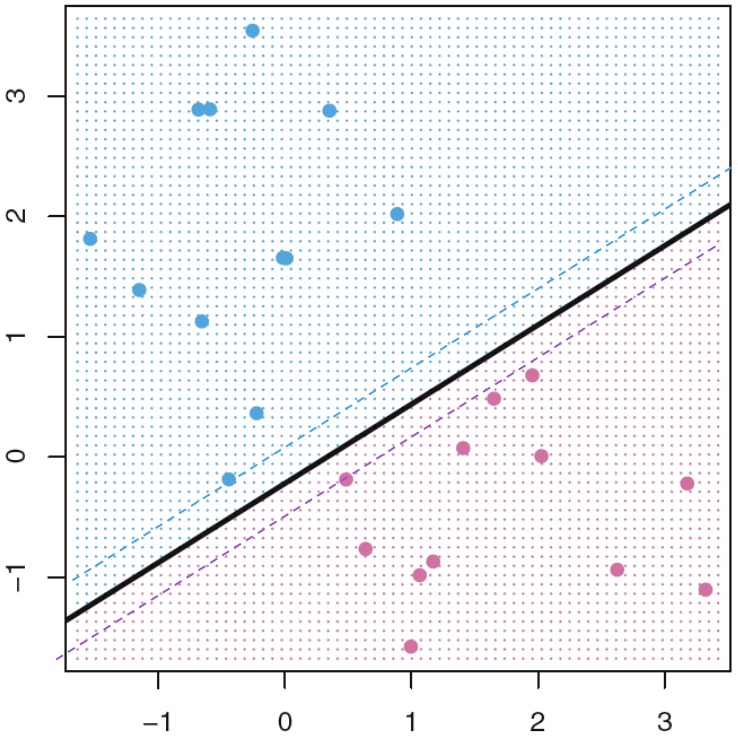
- Given training data $(\boldsymbol{x}_i, y_i), i = 1, \ldots, N$
  - Binary class label: $y_i = \pm 1$
- Suppose it is separable with parameters $(\boldsymbol{w}, b)$
- There must exist a $\gamma > 0$ s.t.:
  - $b + w_1 x_{i1} + \cdots w_d x_{id} > \gamma$ when $y_i = 1$
  - $b + w_1 x_{i1} + \cdots w_d x_{id} < -\gamma$ when $y_i = -1$
- Single equation form:

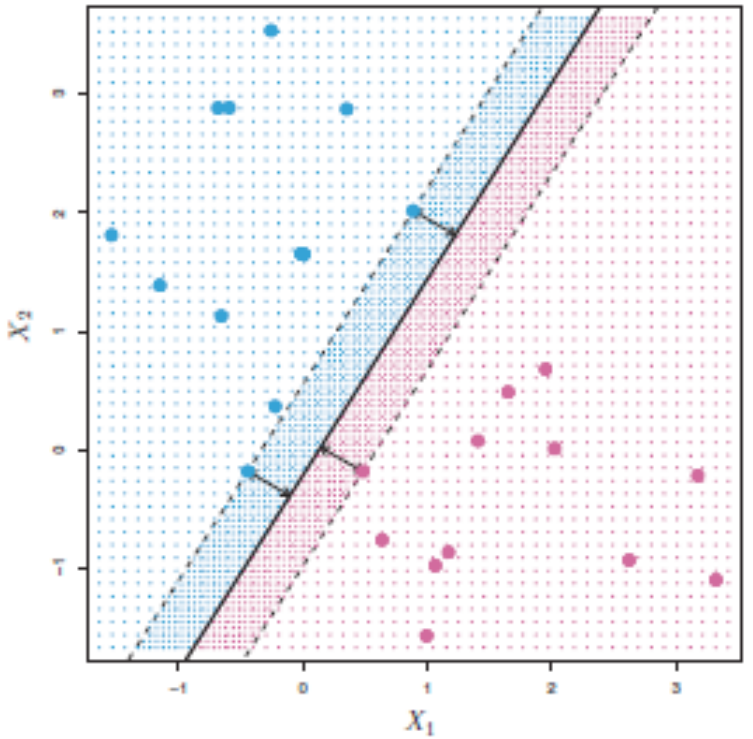
$$y_i(b + w_1 x_{i1} + \cdots w_d x_{id}) > \gamma \text{ for all } i = 1, \ldots, N$$

- Margin: $\mathrm{m} = \frac{\gamma}{\|\boldsymbol{w}\|}$: minimal distance of a sample to the plane
  - γ is the minimum value satisfying the above constraints



Which separating plane is better ?





From Fig. 9.2 and Fig. 9.3 in ISL.

Maximum Margin Classifier

- For the classifier to be more robust to noise, we want to maximize the margin!

- Define maximum margin classifier

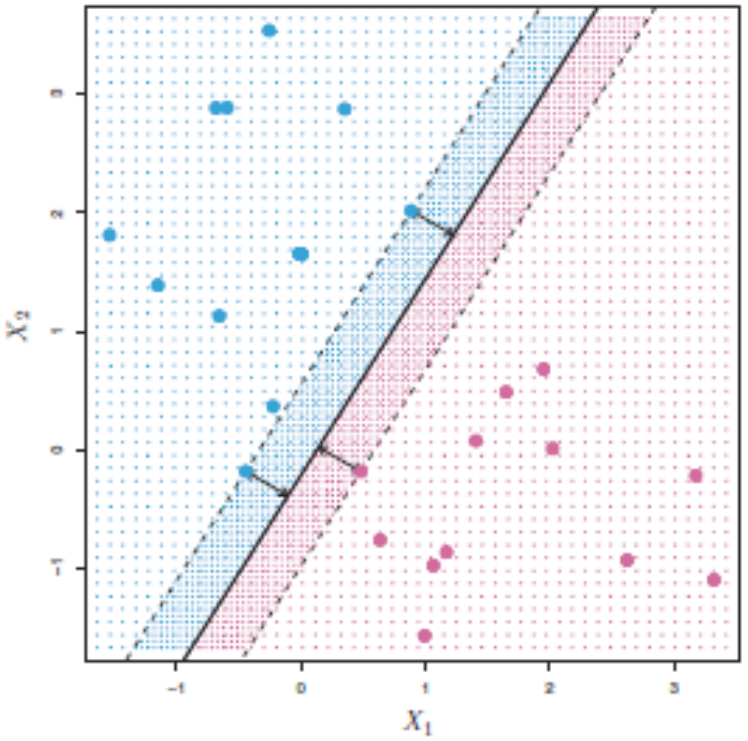
$$\max_{w,\gamma} \gamma$$ ← Maximizes the margin

  - Such that $y_i(b + \boldsymbol{w}^T \boldsymbol{x}) \geq \gamma \quad \text{for all } i$ ← Ensures all points are correctly classified
  - $\sum_{j=1}^{d} w_j^2 \leq 1$ ← Scaling on weights

- Called a constrained optimization
  - Objective function and constraints
  - More on this later.

- See closed form solution in Sec. 4.5.2 in ESL. Note notation difference.

Visualizing Maximum Margin Classifier



- Fig. 9.3 of ISL
- Margin determined by closest points to the line
  - The maximal margin hyperplane represents the mid-line of the widest "slab" that we can insert between two classes
- In this figure, there are 3 points at the margin

ISL: James, Witten, Hastie, Tibshirani, An Introduction to Statistical Learning, Springer. 2013.

Problems with MM classifier

- Data is often not perfectly separable
  - Only want to correctly separate most points

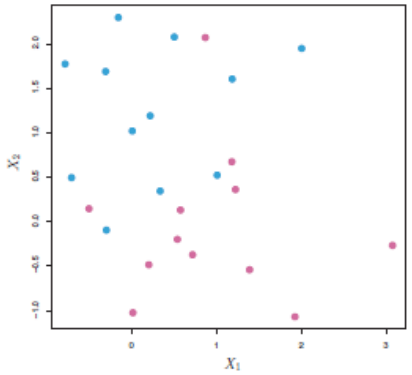


Fig. 9.4

- MM classifier is not robust
  - A single sample can radically change line

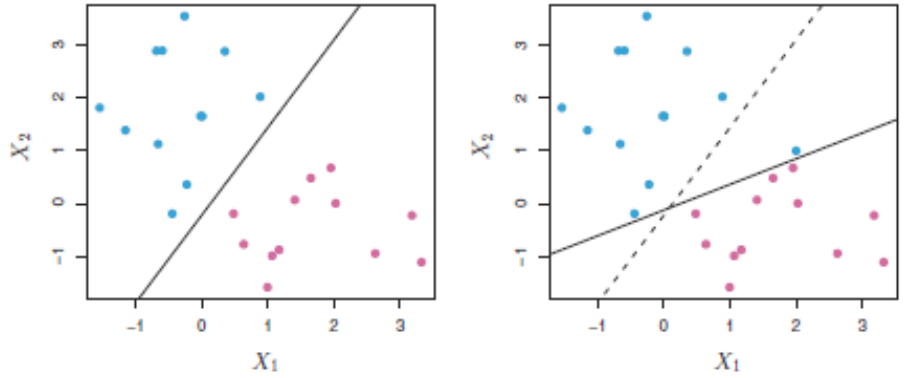


Fig. 9.5

In-Class Exercise

- Found in github site: svm_inclass.ipynb

Problem 1. Margin

For the points below with binary labels:

- Create a scatter plot of the points with different markers for the two classes
- Find the weight and bias of the classifier that separates the two classes
- Compute the distance to the classifier boundary for the points
- Find the margin

```
1 X = np.array([[0.5,0.5], [1,0.5],[0.5,1.75], [0.75,2.75], [1.1,2.2], [2,1], [3,1.5]])
2 y = np.array([1,1,1,0,0,0,0])
```

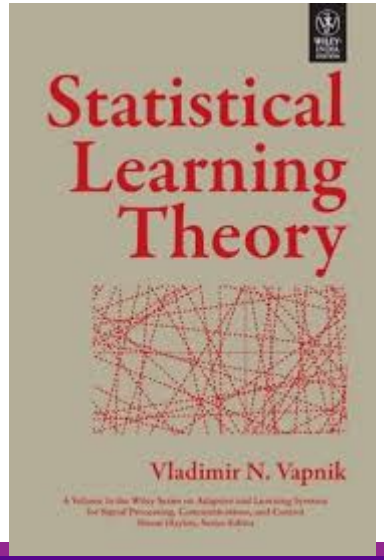
Outline

- Motivating example: Recognizing handwritten digits
  - Why logistic regression doesn't work well.
- Maximum margin classifiers
- Support vector machines
- Kernel trick
- Constrained optimization

Support Vector Machine

- Support Vector Machine (SVM)
  - Vladimir Vapnik, 1963
  - But became widely-used with kernel trick, 1993
  - More on this later

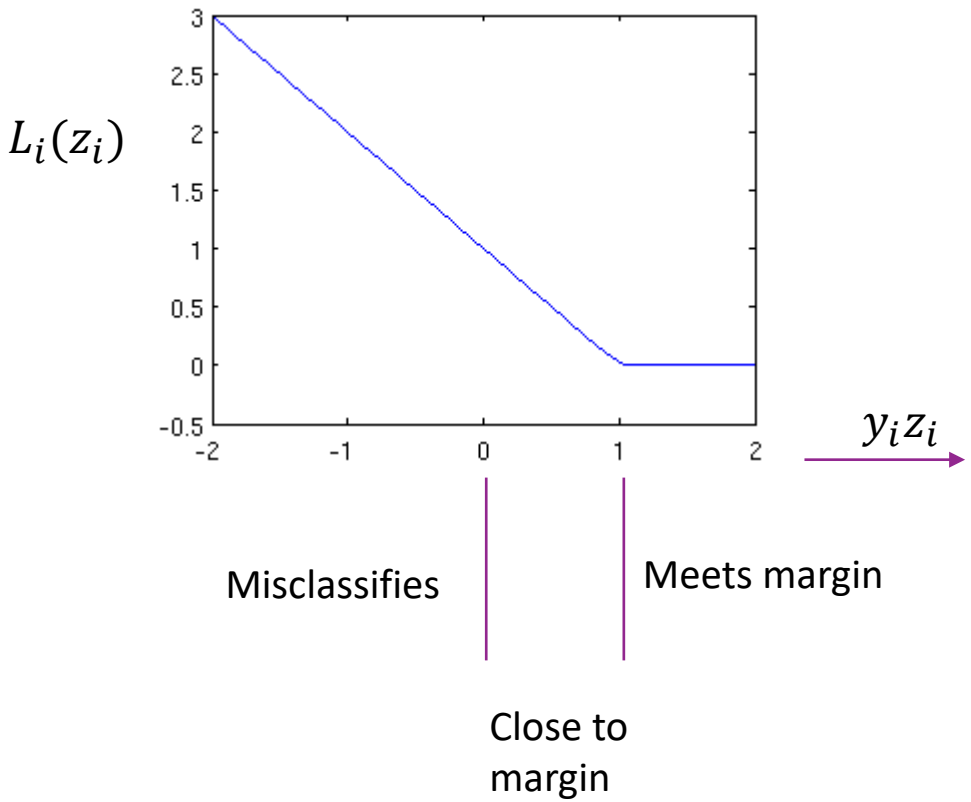
- Got best results on character recognition



- Key idea: Allow “slack” in the classification
  - Support vector classifier (SVC): Directly use raw features. Good when the original feature space is roughly linearly separable
  - Support vector machine (SVM): Map the raw features to some other domain through a kernel function

Hinge Loss

- Fix $\gamma = 1$
- Want ideally: $y_i(\boldsymbol{w}^T\boldsymbol{x} + b) \geq 1$ for all samples i
  - Equivalently, $y_i z_i \geq 1, \quad z_i = b + \boldsymbol{w}^T\boldsymbol{x}$
- But perfect separation may not be possible
- Define hinge loss or soft margin:
  - $L_i(\boldsymbol{w}, b) = \max(0, 1 - y_i z_i)$
- Starts to increase as sample is misclassified:
  - $y_i z_i \geq 1 \Rightarrow$ Sample meets margin target, $L_i(w) = 0$
  - $y_i z_i \in [0,1) \Rightarrow$ Sample margin too small, small loss
  - $y_i z_i \leq 0 \Rightarrow$ Sample misclassified, large loss



SVM Optimization

- Given data $(\boldsymbol{x}_i, y_i)$
- Optimization $\min_{w,b} J(\boldsymbol{w}, b)$

$$J(\boldsymbol{w}, b) = C \sum_{i=1}^{N} \max(0, 1 - y_i(\boldsymbol{w}^T \boldsymbol{x}_i + b)) + \frac{1}{2} \|\boldsymbol{w}\|^2$$

C controls final margin

Hinge loss term
Attempts to reduce
Misclassifications

margin=$1/\|\boldsymbol{w}\|$

- Constant $C > 0$ will be discussed below
- Note: ISL book uses different naming conventions.
  - We have followed convention in sklearn

Alternate Form of SVM Optimization

- Equivalent optimization:

$$\min J_1(\boldsymbol{w}, b, \boldsymbol{\epsilon}), \qquad J_1(\boldsymbol{w}, b, \boldsymbol{\epsilon}) = C\sum_{i=1}^{N} \epsilon_i + \frac{1}{2}\|\boldsymbol{w}\|^2$$

- Subject to constraints:

$$y_i(\boldsymbol{w}^T \boldsymbol{x}_i + b) \geq 1 - \epsilon_i \text{ for all } i = 1, \ldots, N$$

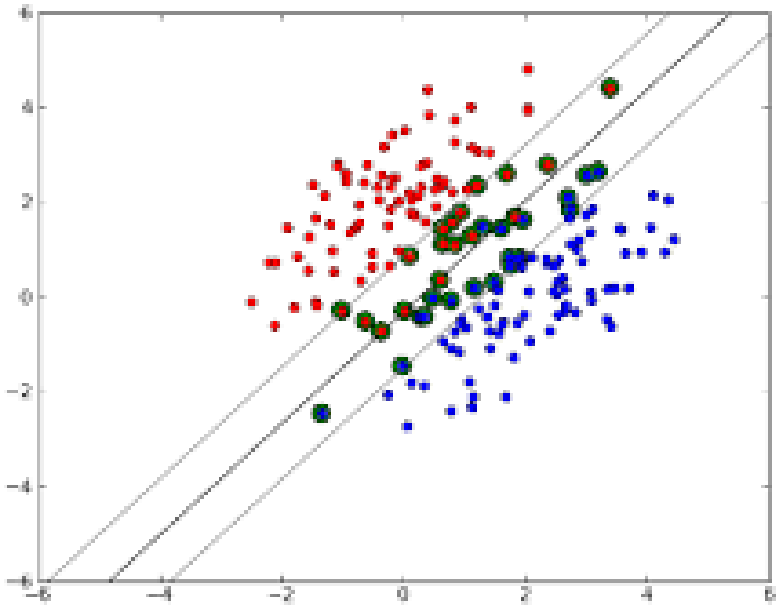
  - ϵ_i = amount sample i misses margin target

- Sometimes write as $J_1(\boldsymbol{w}, b, \boldsymbol{\epsilon}) = C\|\boldsymbol{\epsilon}\|_1 + \frac{1}{2}\|\boldsymbol{w}\|^2$

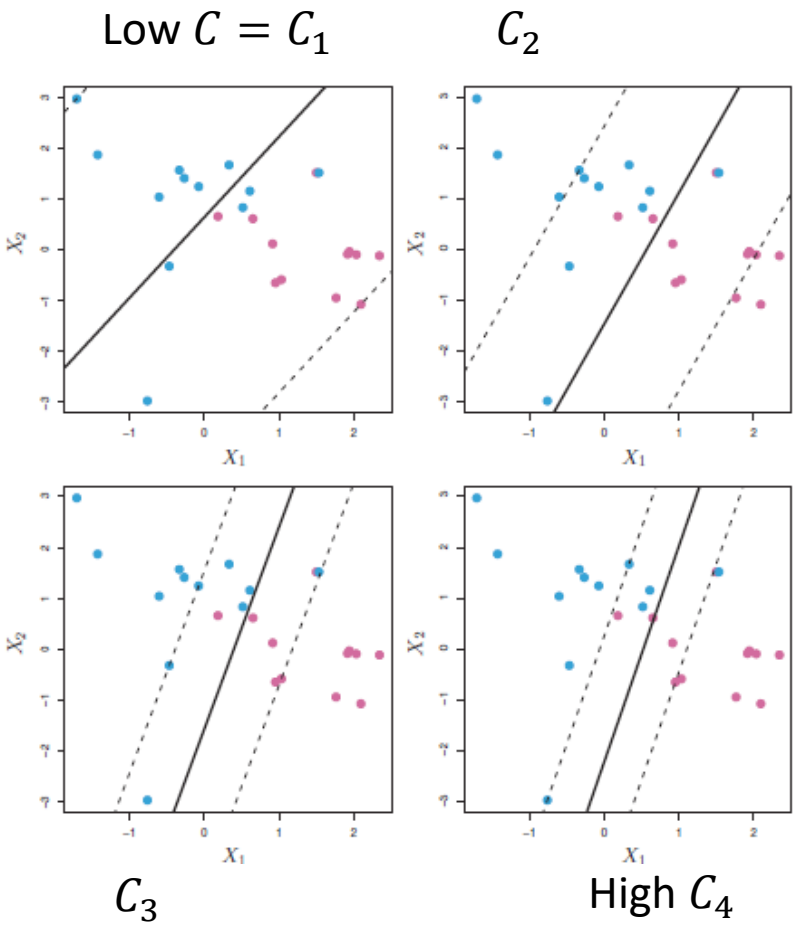
  - $\|\boldsymbol{\epsilon}\|_1 = \sum_{i=1}^{N} \epsilon_i$ called the "one-norm"
  - Generally one-norm would have absolute sign over ϵ_i.
  - But in this case, when the constraint is met, ϵ_i>=0.

Support Vectors

- Support vectors: Samples that either:
  - Are exactly on margin: $y_i(\boldsymbol{w}^T \boldsymbol{x}_i + b) = 1$
  - Or, on wrong side of margin: $y_i(\boldsymbol{w}^T \boldsymbol{x}_i + b) \leq 1$
- Changing samples that are not SVs
  - Does not change solution
  - Provides robustness



Illustrating Effect of C

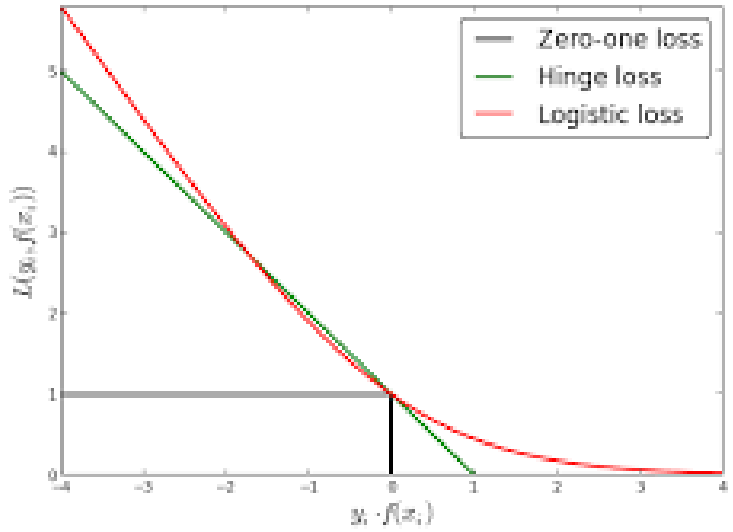


- Fig. 9.7 of ISL
  - Note: C has opposite meaning in ISL than python
  - Here, we use python meaning
- Low C:
  - Leads to large margin
  - But allow many violations of margin.
  - Many more SVs
  - Reduces variance by using more samples
- Large C:
  - Leads to small margin
  - Reduce number of violations, and fewer SVs.
  - Highly fit to data. Low bias, higher variance
  - More chance to overfit

Relation to Logistic Regression

- Logistic regression also minimizes a loss function:

$$J(\boldsymbol{w}, b) = \sum_{i=1}^{N} L_i(w, b)\,, \qquad L_i(w, b) = \ln P(y_i|\boldsymbol{x}_i) = -\ln(1 + e^{-y_i z_i})$$



In-Class Exercise

Problem 2. Minimizing the Hinge Loss

For the data below, first create a scatter plot of the points with different markers for the two classes. You should see that the data is not linearly separable.

Then, consider a set of classifiers:

```
yhat = sign(z),    z = w.dot(x)+b
```

Use the the `w` below, plot the hinge loss as a function of the bias `b` where the hinge loss is:

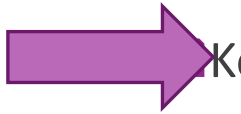
```
J  = sum( maximum(0, 1-ypm*z) )
```

Here `ypm=2*y-1` so that it is a value +1 or -1. Find the `b` that minimizes the hinge loss and plot the boundary of the classifier.

```
1 X = np.array([[0.5,0.5], [1,0.5],[0.5,1.75], [2,2], [0.75,0.75], [0.75,2.75], [1.1,2.2], [2,1], [3,1.5]])
2 y = np.array([1,1,1,1,0,0,0,0,0])
3
4 w = np.array([1.5, 1])
5 w = w / np.linalg.norm(w)
```

Outline

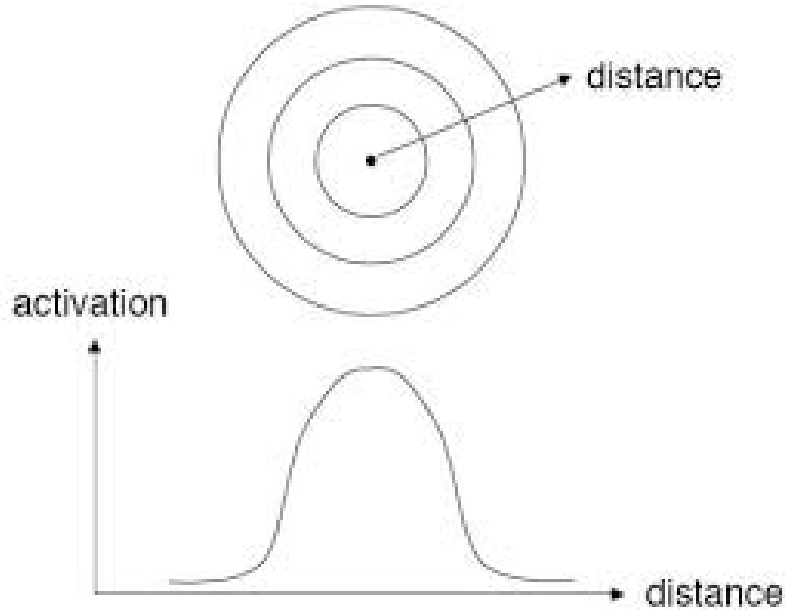
- Motivating example: Recognizing handwritten digits
  - Why logistic regression doesn't work well.
- Maximum margin classifiers
- Support vector machines
- Kernel trick
- Constrained optimization



The Kernel Function

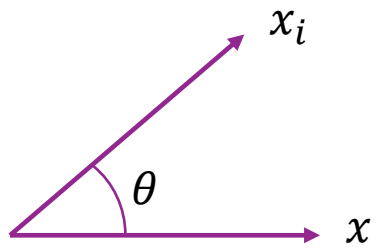
- Kernel function:
  - Function $K(\boldsymbol{x}_i, \boldsymbol{x})$
  - Key function for SVMs and kernel classifiers
  - Measures "similarity" between new sample $\boldsymbol{x}$ and training sample $\boldsymbol{x}_i$

- 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

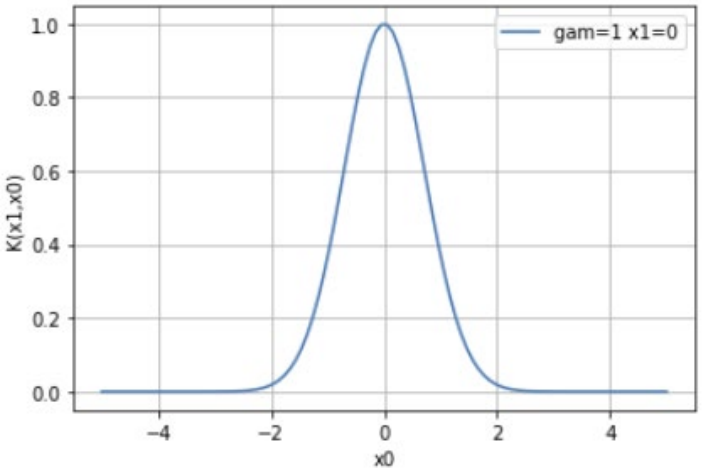
- Linear SVM:
  - $K(x_i, x) = x_i^T x = \|x_i\|\|x\| \cos\theta$
  - Maximum when angle between vectors is small



- Radial basis function:

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

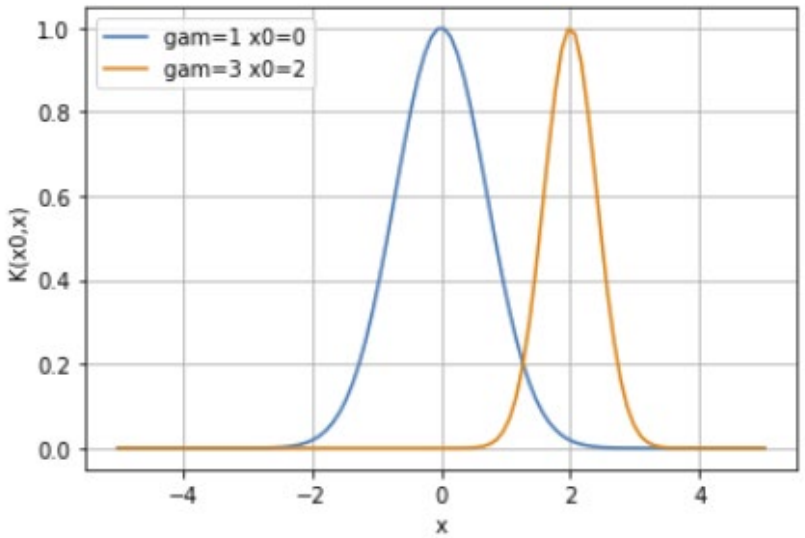
  - $1/\gamma$ indicates width of kernel



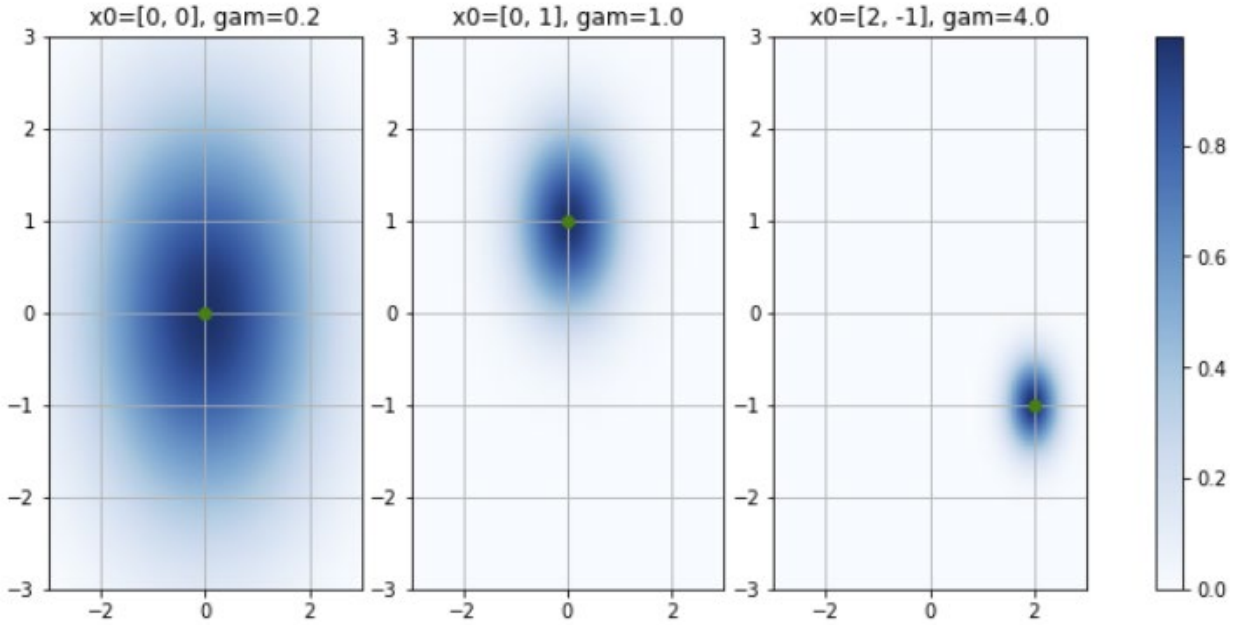
- Polynomial kernel: $K(x_i, x) = \left|x_i^T x\right|^d$
  - Typically d=2

RBF Kernel Examples

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



RBFs in 1D



RBFs in 2D

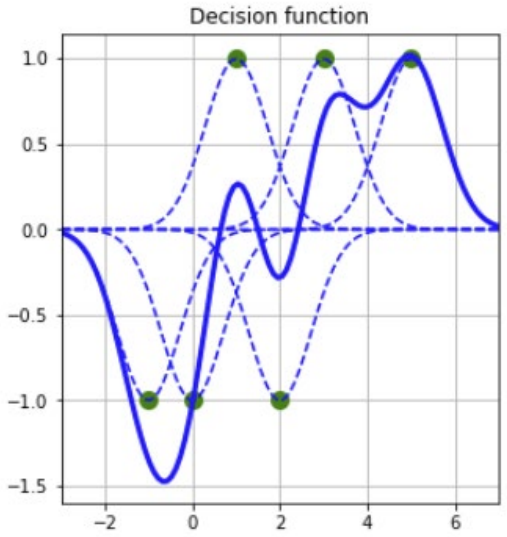
Kernel Classifier

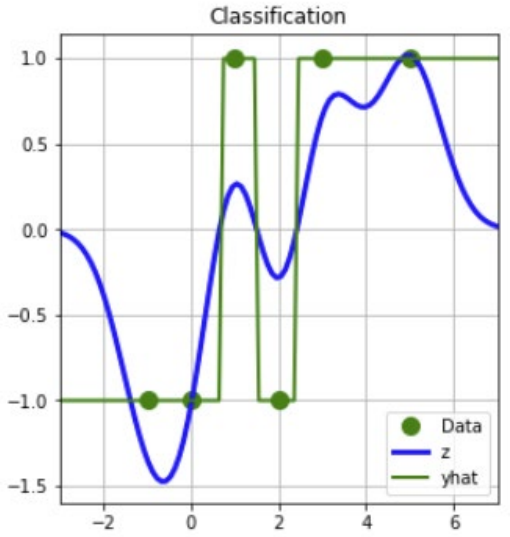
- Given:
  - Training data (x_i, y_i) with binary labels $y_i = \pm 1$
  - Kernel $K(x_i, x)$
- To classify a new point x:
  - Decision function: $z = \sum_{i=1}^{n} y_i K(x_i, x)$
  - Classify: $\hat{y} = sign(z)$
- Idea:
  - z is large positive when x is close to samples x_i with $y_i = 1$
  - z is large negative when x is close to samples x_i with $y_i = -1$
- Kernel classifiers are a subject on their own
  - We just mention them here to explain connection to SVMs

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:
  - $\hat{y} = sign(z)$

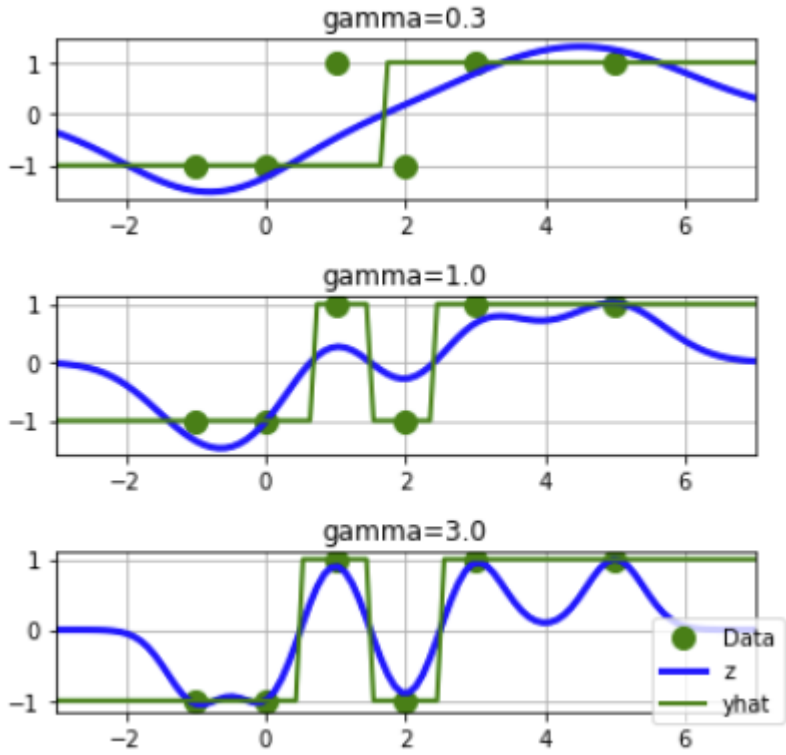
i	1	2	3	4	5	6
x_i	-1	0	1	2	3	5
y_i	-1	-1	1	-1	1	1





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



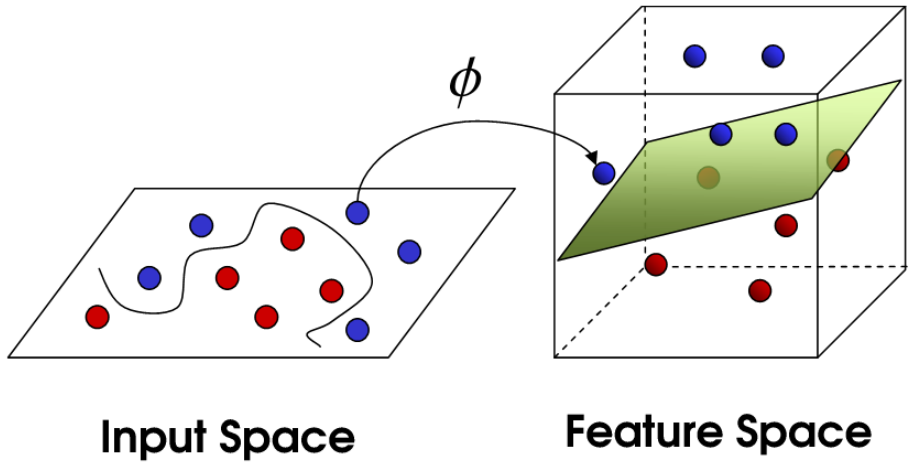
SVMs with Non-Linear Transformations

- Non-linear transformation:
  - Replace $\boldsymbol{x}$ with $\phi(\boldsymbol{x})$
  - Enables more rich, non-linear classifiers
  - Examples: polynomial classification

$$\phi(x) = [1, x, x^2, \dots, x^{d-1}]$$

- Tries to find separation in a feature space

- Kernel trick in SVMs:
  - Makes applying non-linear transformations easy



SVM with the Transformation

- Consider SVM model with $\boldsymbol{x}$ replaced by $\phi(\boldsymbol{x})$
- Minimize SVM cost function as before (i.e. Hinge loss + inverse margin)
- Theorem: The optimal weight is of the form:

$$\boldsymbol{w} = \sum_{i=1}^{N} \alpha_i y_i \phi(\boldsymbol{x}_i)$$

  - $\alpha_i \geq 0$ for all i
  - $\alpha_i > 0$ if and only if sample i is a support vector
  - Will show this fact later using results in constrained optimization

- Consequence: The linear discriminant on any other sample x is:

$$z = b + \boldsymbol{w}^T \phi(\boldsymbol{x}) = b + \sum_{i=1}^{N} \alpha_i y_i \phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x})$$

$K(\boldsymbol{x}_i, \boldsymbol{x})$ = "kernel"

Kernel Form of the SVM Classifier

- SVM classifier can be written with the kernel $K(\boldsymbol{x}_i, \boldsymbol{x})$ and values $\alpha_i \geq 0$:

$$z = b + \sum_{i=1}^{N} \alpha_i y_i K(\boldsymbol{x}_i, \boldsymbol{x}),$$

Decision function

$$\hat{y} = \text{sign}(z) = \begin{cases} 1 & \text{if } z > 0 \\ -1 & \text{if } z < 0 \end{cases}$$

Classification decision

- Key point: SVM classifier is approximately Kernel classifier
- But there are two differences:
  - Weights $\alpha_i \geq 0$ on the samples (the weights are only non-zero on the SVs)
  - A bias term b (can be positive or negative)

“Kernel Trick” and Dual Parameterization

- Kernel form of SVM classifier (previous slide):

$$z = b + \sum_{i=1}^{N} \alpha_i y_i K(\boldsymbol{x}_i, \boldsymbol{x}),$$
$$\hat{y} = \text{sign}(z)$$

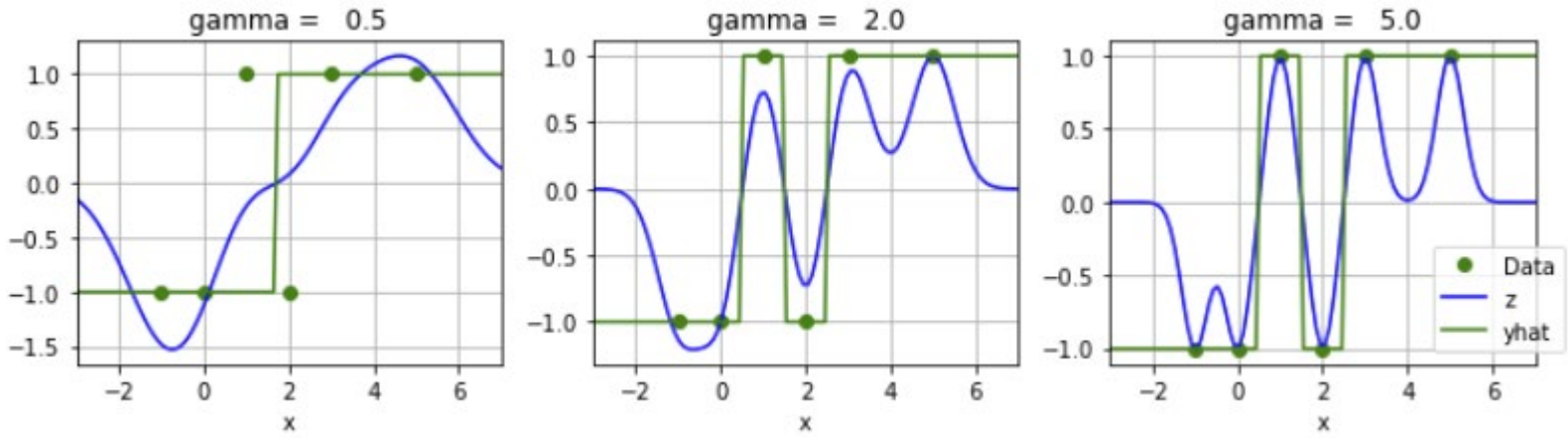
- Dual parameters: $\alpha_i \geq 0, i = 1, \ldots, N$
  - Called the dual parameters due to constrained optimization – see next section

- Kernel trick:
  - Directly solve the parameters $\boldsymbol{\alpha}$ instead of the weights $\boldsymbol{w}$
  - Can show that the optimization only needs the kernel $K(\boldsymbol{x}_i, \boldsymbol{x})$
  - Does not need to explicitly use $\phi(\boldsymbol{x})$

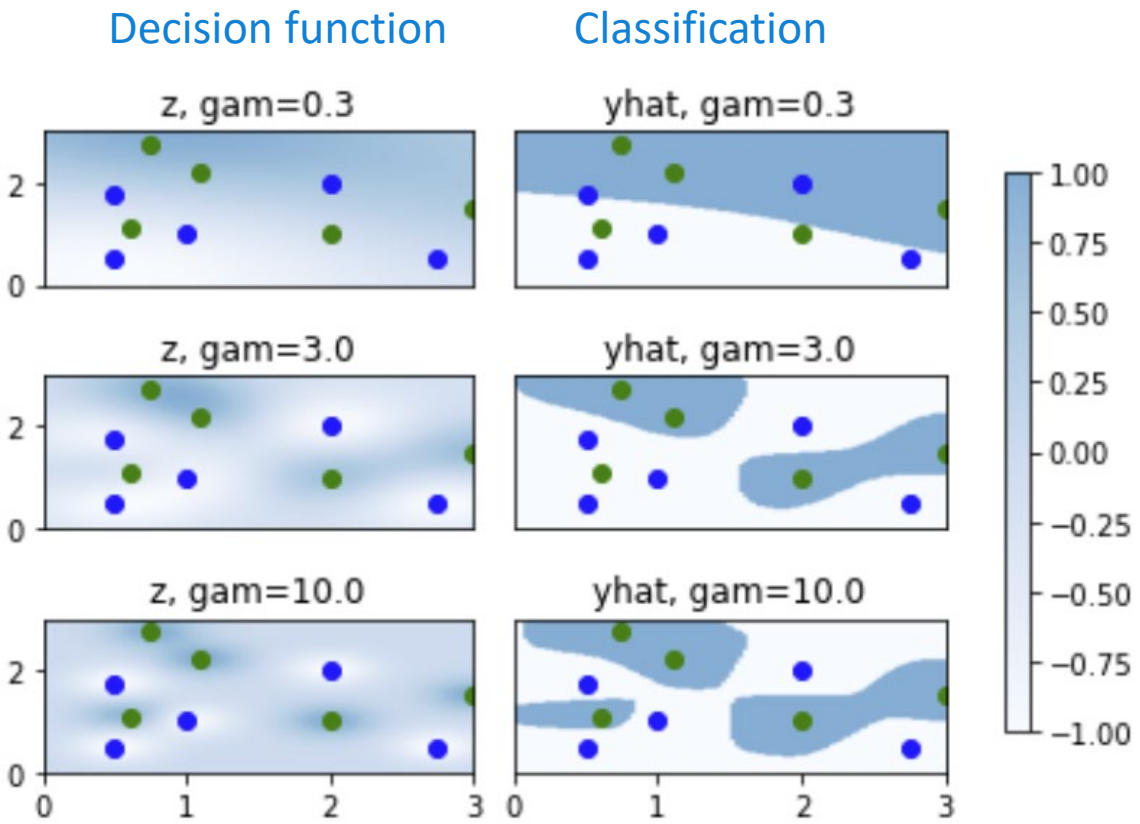
SVM Example in 1D

- Same data as in the Kernel classifier example
- Fit SVM with RBF with different γ
- Similar trends as kernel classifier: As γ increases
  - z "fits" data (x_i, y_i) closer
  - Leads to more complex decision regions.
  - Enables nonlinear decision regions

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



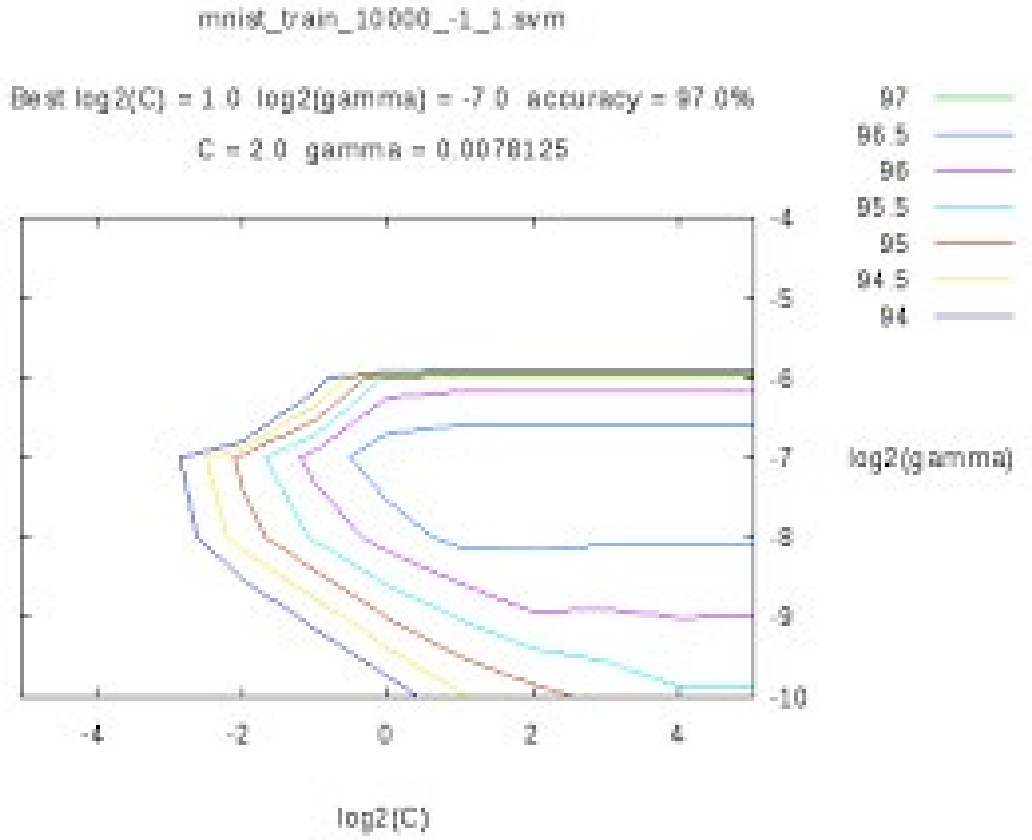
Example in 2D



- Example:
  - 10 data points with binary labels
  - Fit SVM with $C = 1$ and RBF
  - $\gamma = 0.3$, 3 and 10
- Plot:
  - z= linear discriminant
  - $\hat{y} = sign(z) =$ classification decision
- Observe: As γ increases
  - Fits training data better
  - More complex decision region

Parameter Selection

- For SVMs with RBFs we need to select:
  - Parameter $C > 0$ in the loss function
  - Kernel width $\gamma > 0$
- Higher C or γ
  - Fewer SVs
  - Classifiers averages over smaller set
  - Lower bias, but higher variance
- Typically select via cross-validation
  - Try out different (C, γ) pairs
  - Find which one provides highest accuracy on test set
- Python can automatically do grid search



http://peekaboo-vision.blogspot.com/2010/09/mnist-for-ever.html

Multi-Class SVMs

- Suppose there are K classes
- One-vs-one:
  - Train $\binom{K}{2}$ SVMs for each pair of classes
  - Test sample assigned to class that wins "majority of votes"
  - Best results but very slow
- One-vs-rest:
  - Train K SVMs: train each class k against all other classes
  - Pick class with highest z_k
- Sklearn has both options

MNIST Results

- Run classifier
- Very slow
  - Several minutes for 40,000 samples
  - Slow in training and test
  - Major drawback of SVM
- Accuracy ≈ 0.984
  - Much better than logistic regression
- Can get better with:
  - pre-processing
  - More training data
  - Optimal parameter selection

```
from sklearn import svm

# Create a classifier: a support vector classifier
svc = svm.SVC(probability=False,  kernel="rbf", C=2.8, gamma=.0073,verbose=10)
```

```
svc.fit(Xtr,ytr)
```

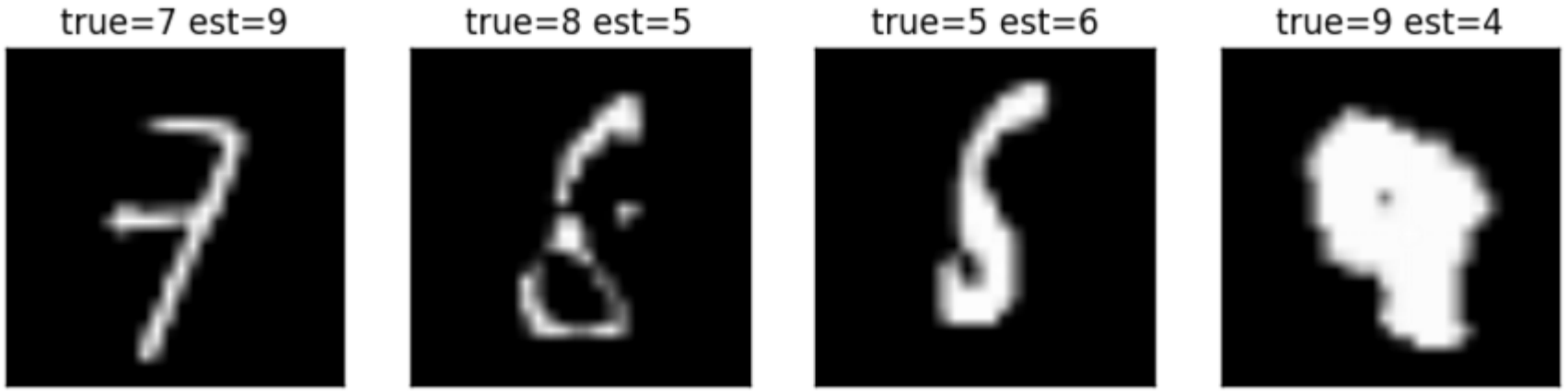
```
[LibSVM]
SVC(C=2.8, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape=None, degree=3, gamma=0.0073, kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=10)
```

```
yhat1 = svc.predict(Xts)
acc = np.mean(yhat1 == yts)
print('Accuaracy = {0:f}'.format(acc))
```

```
Accuaracy = 0.984000
```

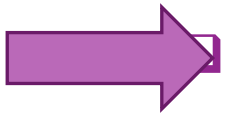
MNIST Errors

- Some of the error are hard even for a human



Outline

- Motivating example: Recognizing handwritten digits
  - Why logistic regression doesn't work well.
- Maximum margin classifiers
- Support vector machines
- Kernel trick
- Constrained optimization



Constrained Optimization

- In many problems, variables are constrained
- Constrained optimization formulation:
  - Objective: Minimize $f(\boldsymbol{w})$
  - Constraints: $g_1(\boldsymbol{w}) \leq 0, \ldots, g_M(\boldsymbol{w}) \leq 0$
- Examples:
  - Minimize the mpg of a car subject to a cost or meeting some performance
  - In ML: weight vector may have constraints from physical knowledge
- Often write constraints in vector form: Write $g(\boldsymbol{w}) \leq 0$

$$g(\boldsymbol{w}) = [g_1(\boldsymbol{w}), \ldots, g_m(\boldsymbol{w})]^T$$

Lagrangian

- Constrained optimization: Min $f(\boldsymbol{w})$ s.t. $g(\boldsymbol{w}) \leq 0$
- Consider first a single constraint: $g(\boldsymbol{w})$ is a scalar
- Define Lagrangian: $L(\boldsymbol{w}, \lambda) = f(\boldsymbol{w}) + \lambda g(\boldsymbol{w})$
  - $\boldsymbol{w}$ is called the primal variable
  - λ is called the dual variable
- Dual minimization: Given a dual parameter λ, minimize

$$\widehat{\boldsymbol{w}}(\lambda) = \arg\min_{\boldsymbol{w}} L(\boldsymbol{w}, \lambda), \qquad L^*(\lambda) = \min_{\boldsymbol{w}} L(\boldsymbol{w}, \lambda)$$

  - Minimizes a weighted combination of objective and constraint.
  - Higher $\lambda \Rightarrow$ Weight constraint more (try to make $g(\boldsymbol{w})$ smaller)
  - Lower $\lambda \Rightarrow$ Weight objective more (try to make $f(\boldsymbol{w})$ smaller)

KKT Conditions

- Given objective $f(\boldsymbol{w})$ and constraint $g(\boldsymbol{w})$
- KKT Conditions: $\hat{\boldsymbol{w}}, \hat{\lambda}$ satisfy:
  - $\hat{\boldsymbol{w}}$ minimizes the Lagrangian: $\hat{\boldsymbol{w}} = \arg\min_{\boldsymbol{w}} L(\boldsymbol{w}, \hat{\lambda})$
  - Either
    - $g(\hat{\boldsymbol{w}}) = 0$ and $\hat{\lambda} \geq 0$ [active constraint]
    - $g(\hat{\boldsymbol{w}}) < 0$ and $\hat{\lambda} = 0$ [inactive constraint]
- Theorem: Under some technical conditions,
  - if $\hat{\boldsymbol{w}}, \hat{\lambda}$ are local mimima of the constrained optimization, they must satisfy KKT conditions

General Procedure for Single Constraint

- Suppose:
  - $\boldsymbol{w} = (w_1, \ldots, w_d)^T$: d unknown primal variables
  - $g(\boldsymbol{w}) \leq 0$: scalar constraint
- Case 1: Assume constraint is active:
  - Solve $\boldsymbol{w}$ and λ: $\partial L(\boldsymbol{w}, \lambda)/\partial w_j = 0$ and $g(\boldsymbol{w}) = 0$ (resulting from setting $\partial L(\boldsymbol{w}, \lambda)/\partial \lambda = 0$)
  - $d + 1$ unknowns and $d + 1$ equations
  - Verify that $\lambda \geq 0$
- Case 2: Assume constraint is inactive
  - Solve primal objective $\partial f(\boldsymbol{w})/\partial w_j = 0$ ignoring constraint
  - d unknowns and d equations
  - Verify that constraint is satisfied: $g(\boldsymbol{w}) \leq 0$

KKT Conditions Illustrated

- Example 1: Constraint is “active”

$$\min_{w} w^2 \quad s.t.\ w + 1 \leq 0$$

- Example 2: Constraint is “inactive”

$$\min_{w} w^2 \quad s.t.\ w - 1 \leq 0$$

- Examples worked on board with illustration

Multiple Constraints

- Now consider constraint: $g(\boldsymbol{w}) = [g_1(\boldsymbol{w}), \ldots, g_M(\boldsymbol{w})]^T \leq 0.$
- Lagrangian is:

$$L(\boldsymbol{w}, \boldsymbol{\lambda}) = f(\boldsymbol{w}) + \boldsymbol{\lambda}^T g(\boldsymbol{w}) = f(\boldsymbol{w}) + \sum_{m=1}^{M} \lambda_m \, g_m(\boldsymbol{w})$$

  - Weighted sum of all M constraints
  - $\boldsymbol{\lambda}$ is called the dual vector

- KKT conditions extend to:
  - $\hat{\boldsymbol{w}}$ minimizes the Lagrangian: $\hat{\boldsymbol{w}} = \arg\min_{\boldsymbol{w}} L(\boldsymbol{w}, \hat{\lambda})$
  - For each $m = 1, \ldots, M$
    - $g_m(\hat{\boldsymbol{w}}) = 0$ and $\hat{\lambda}_m \geq 0$ [active constraint]
    - $g_m(\hat{\boldsymbol{w}}) < 0$ and $\hat{\lambda}_m = 0$ [inactive constraint]

SVM Constrained Optimization

- Recall: SVM constrained optimization

$$\min J_1(\boldsymbol{w}, b, \boldsymbol{\epsilon}), \qquad J_1(\boldsymbol{w}, b, \boldsymbol{\epsilon}) = C\sum_{i=1}^{N} \epsilon_i + \frac{1}{2}\|\boldsymbol{w}\|^2$$

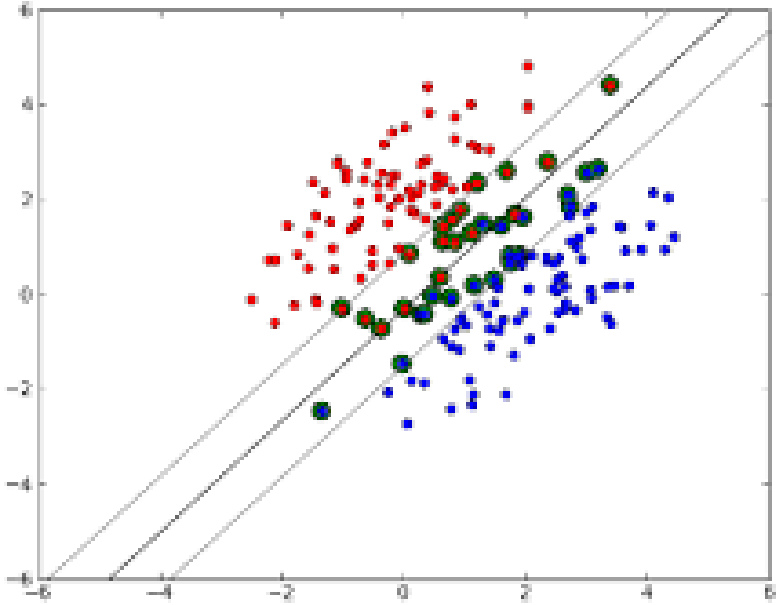
  - Constraints: $y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) \geq 1 - \epsilon_i$ and $\epsilon_i \geq 0$ for all $i = 1, \ldots, N$

- After applying KKT conditions and some algebra [beyond this class], solution is
  - Optimal weight vector: $\boldsymbol{w} = \sum_{i=1}^{N} \alpha_i y_i \boldsymbol{x}_i$ linear combination of instances
  - Dual parameters α_i minimize

$$\sum_{i=1}^{N} \alpha_i + \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \boldsymbol{x}_i^T \boldsymbol{x}_j \quad \text{s.t.} \quad 0 \leq \alpha_i \leq C$$

Support Vectors

- Classifier weight is: $\boldsymbol{w} = \sum_{i=1}^{N} \alpha_i y_i \boldsymbol{x}_i$
- Can show that $\alpha_i > 0$ only when $\boldsymbol{x}_i$ is a support vector
  - On boundary or violating constraint
  - Otherwise $\alpha_i = 0$



What you should know

- ❑Interpret weights in linear classification of images (logistic regression): Match filters
- ❑Understand the margin in linear classification and maximum margin classifier
- ❑SVM classifier: Allow violation of margin by introducing slack variables (More robust than linear classifier)
- ❑Solve constrained optimization using the Lagrangian.
  - Understand KKT conditions for a single constraint
- ❑Extend to nonlinear classifier by feature transformation: SVM with nonlinear kernels
- ❑Select SVM parameters from cross-validation

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

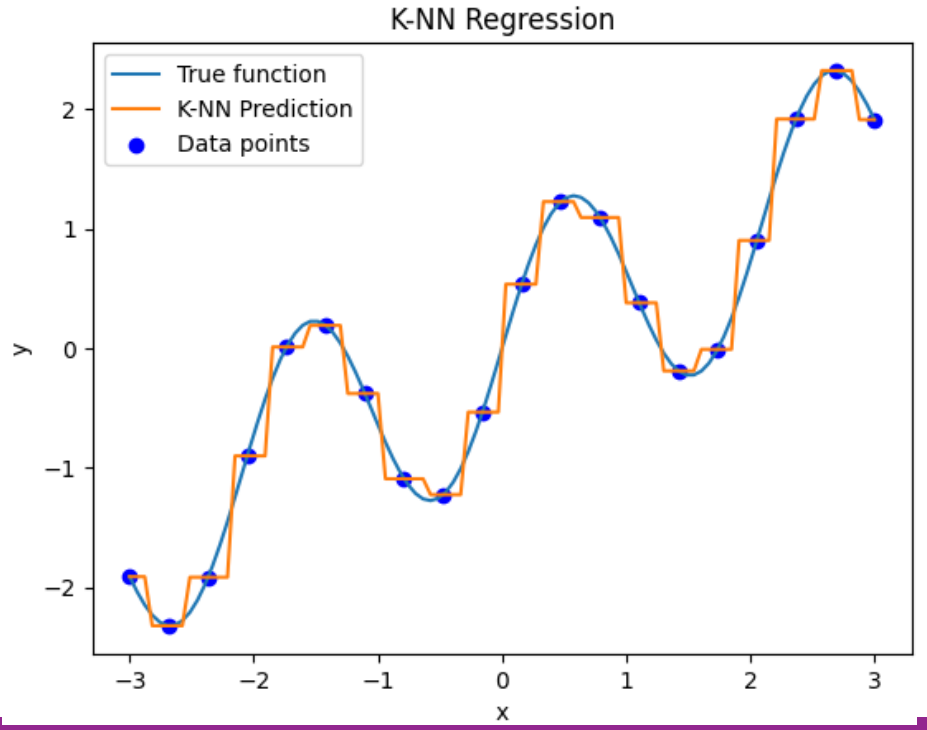
- K-Nearest Neighbors
- Kernel Smoothing
- Kernel Regression
- Example with Climate Data

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 $\hat{y}_0 = x_i$



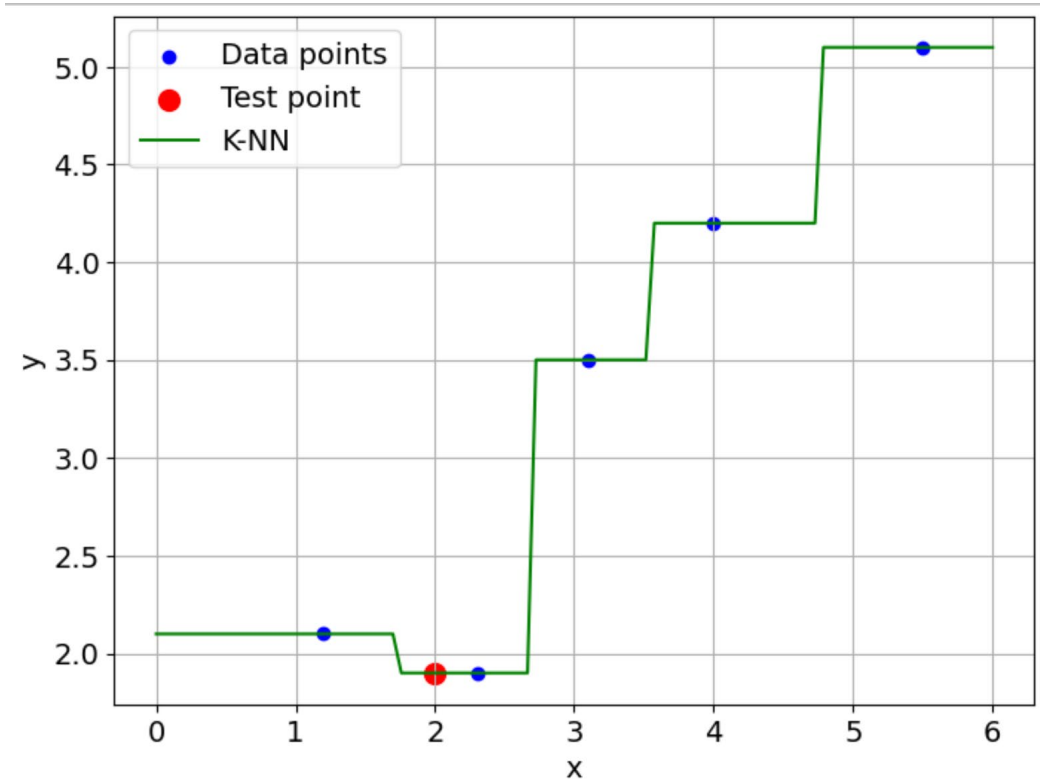
Example

Data

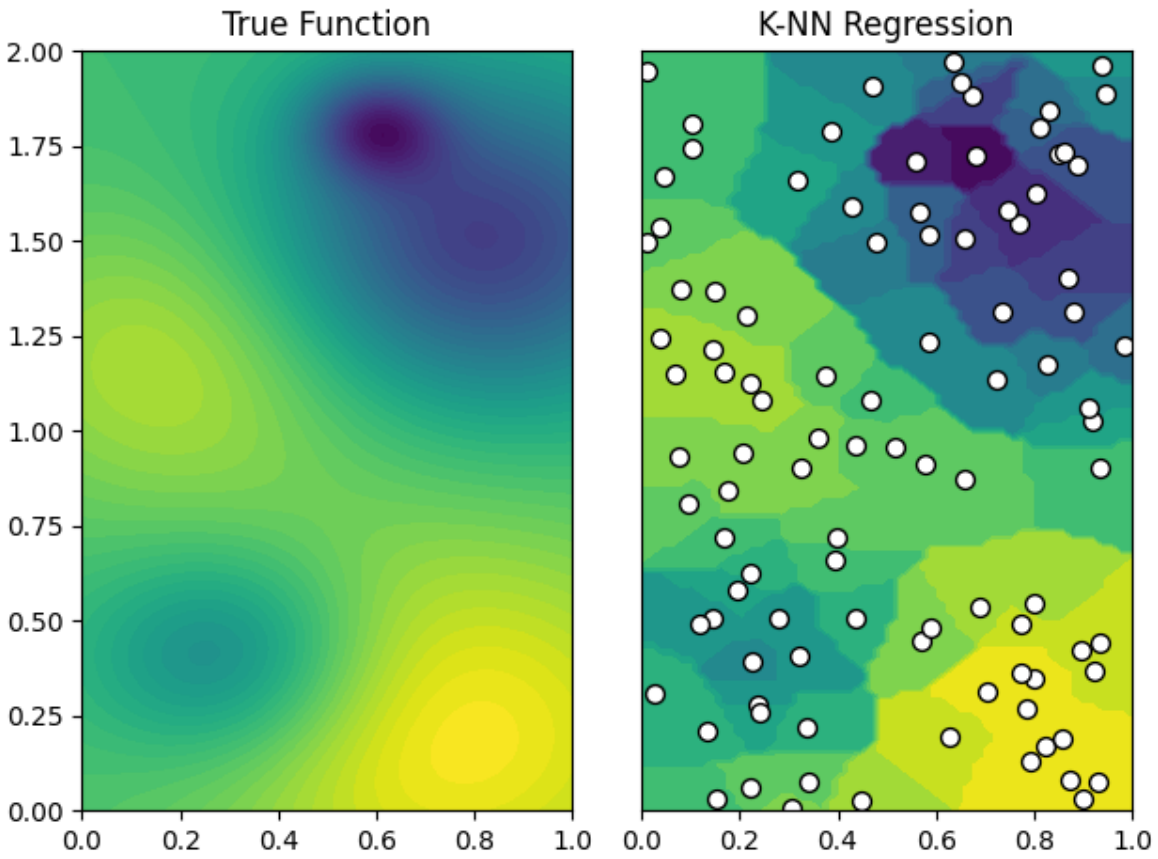
Closest to test point $x_0 = 2$

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



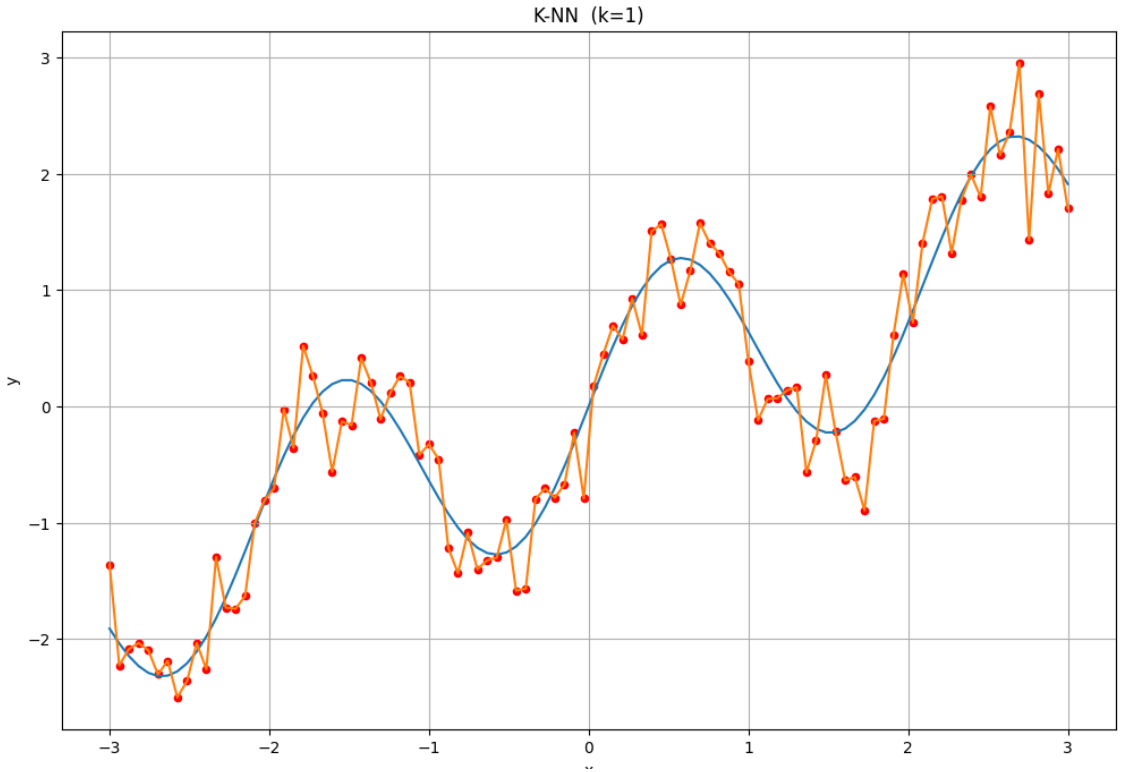
- Gives a “tessellation” of space
- Vornoi regions
  - Each region mapped to a point

K-Nearest Neighbor

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

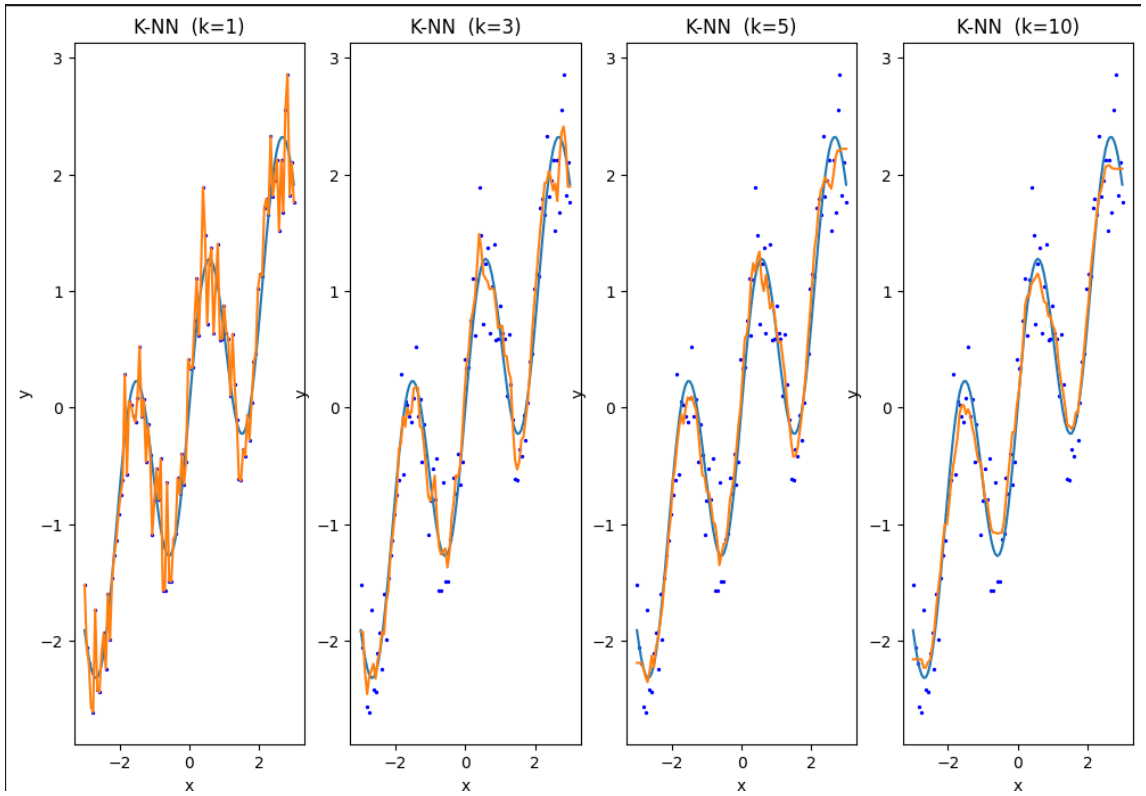
- K-Nearest neighbor:
  - Take average of K closest points

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



Example of overfitting data

1D Example



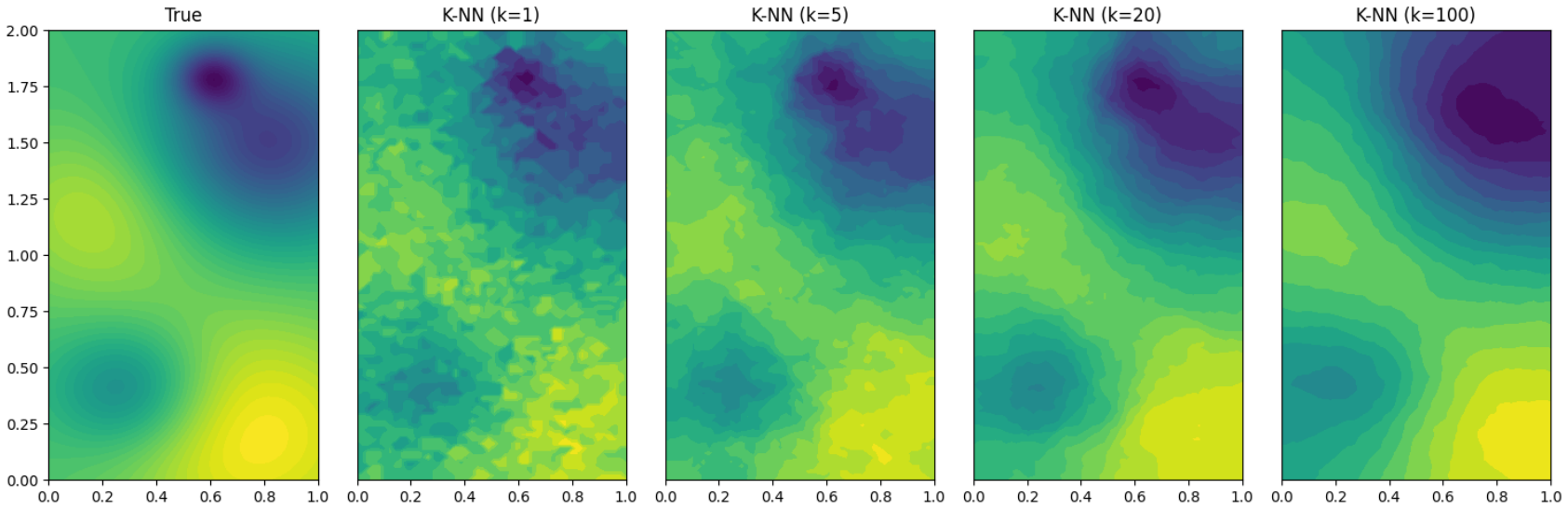
High variance error

Low bias error

Low variance error

High bias error

2D Example



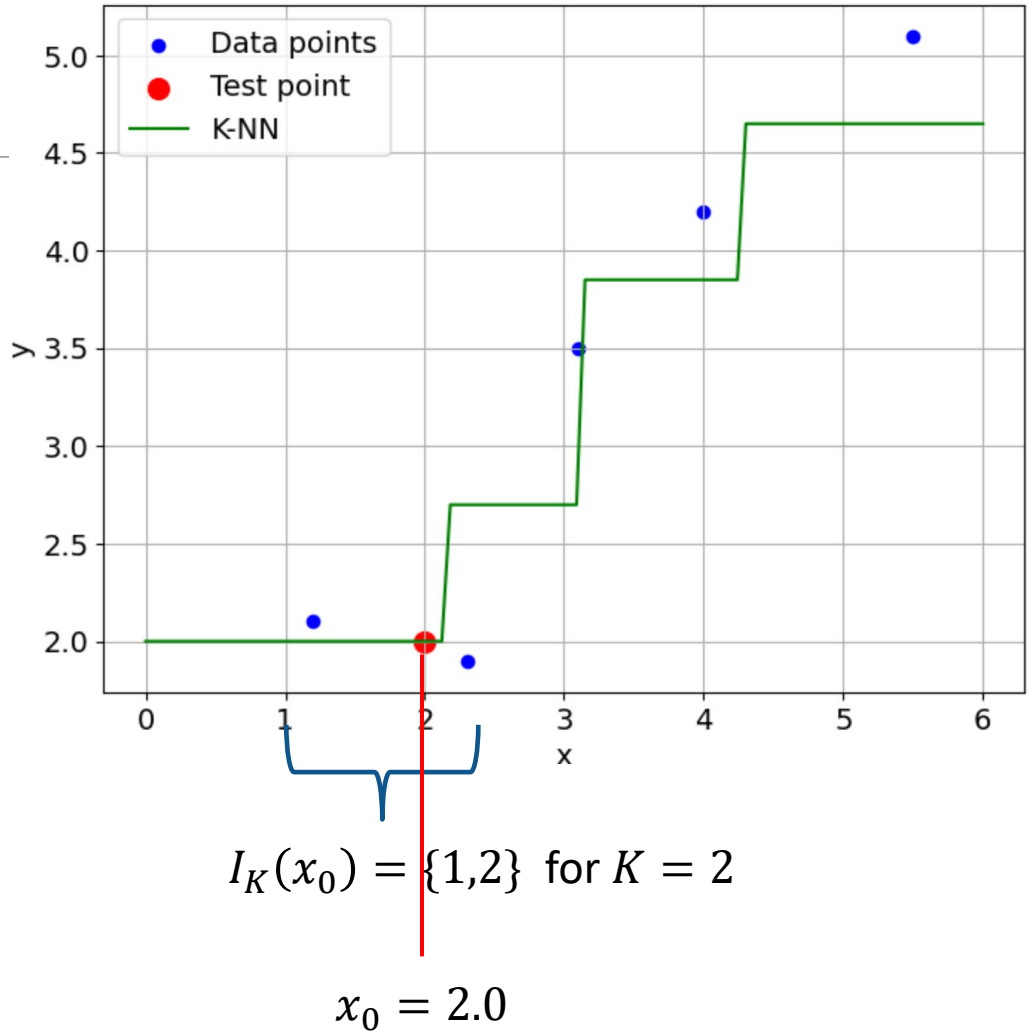
- Simulation details
  - 1000 Data points
  - $y_i = f(x_i) + w_i$
  - $E|w_i|^2 = 0.3^2$

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?

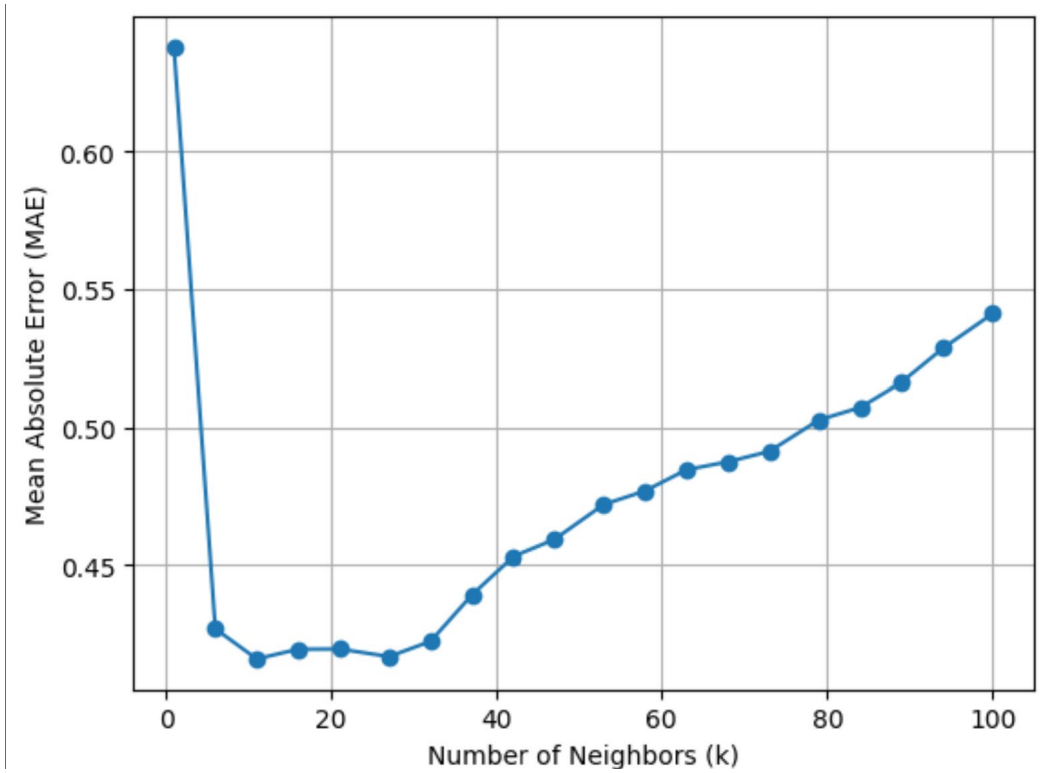


Bias and Variance

- 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

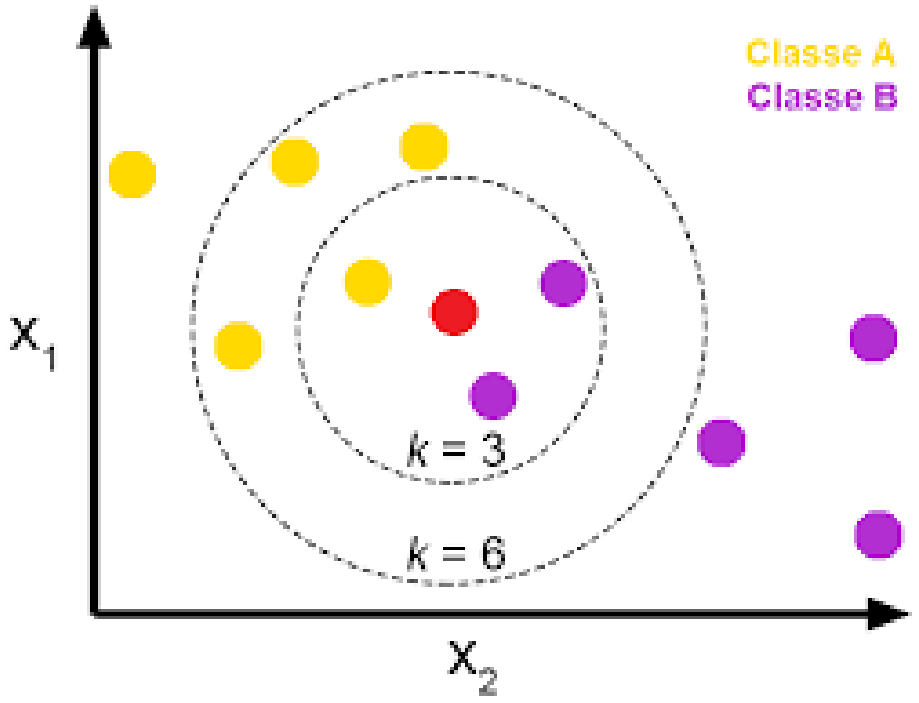
- Ex: K-NN
- No functional assumption
  - No parameters
  - But assumes smoothness
- No fitting
- Use data directly

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

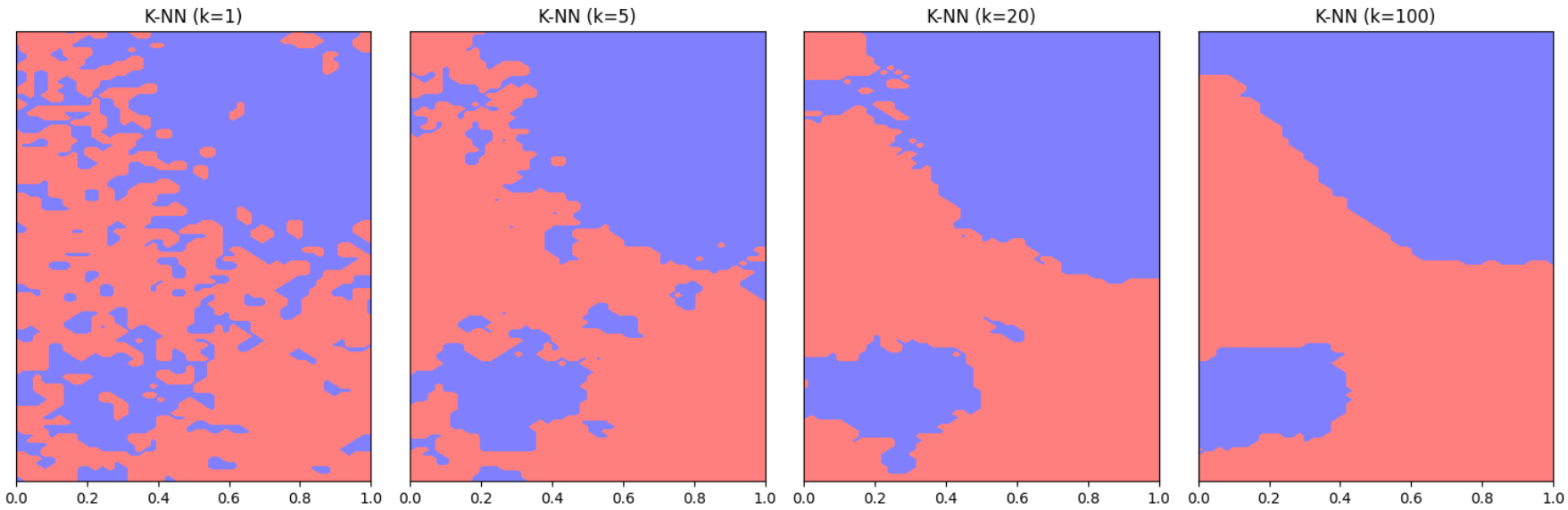
K-NN Classification

- 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

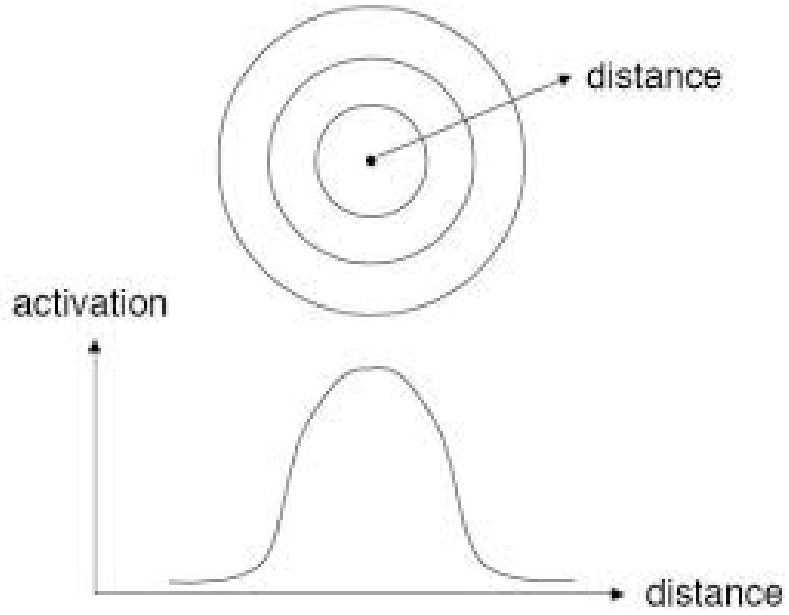
- K-Nearest Neighbors
- Kernel Smoothing
- Kernel Regression
- Example with Climate Data

The Kernel Function

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

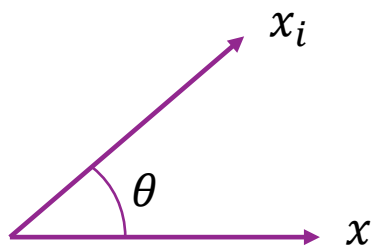
- Kernel function:
  - Function $K(\boldsymbol{x}_i, \boldsymbol{x})$
  - Measures "similarity" between new sample $\boldsymbol{x}$ and trainin

- 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

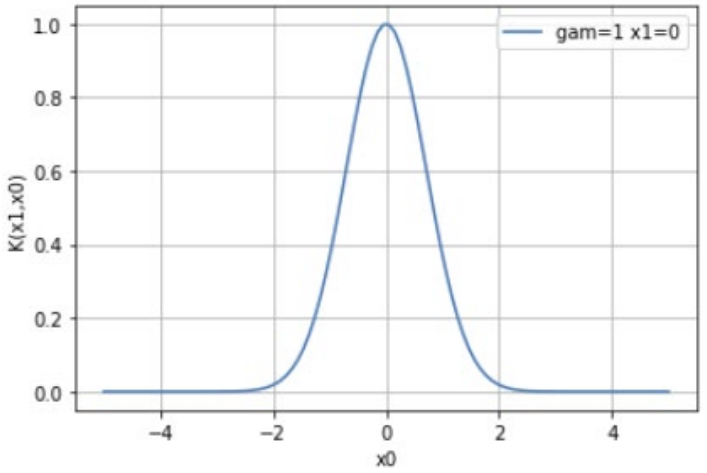
- Linear :
  - $K(x_i, x) = x_i^T x = \|x_i\|\|x\| \cos\theta$
  - Maximum when angle between vectors is small



- Radial basis function:

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

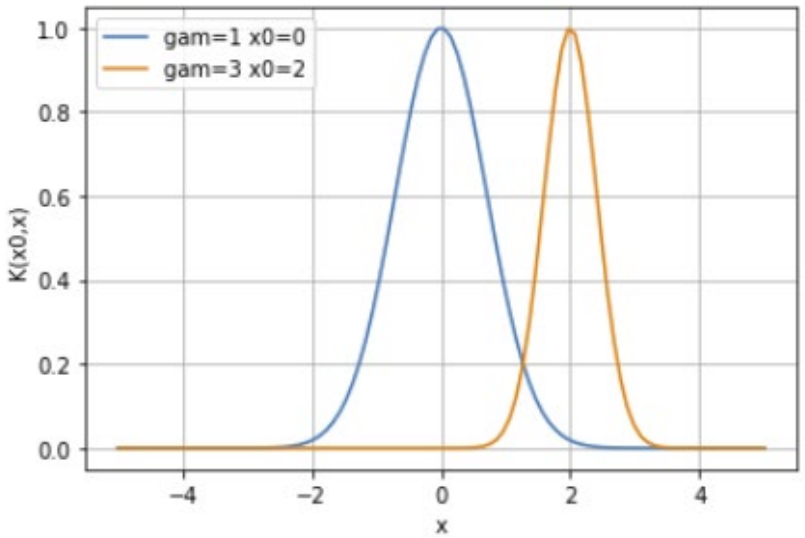
  - $1/\gamma$ indicates width of kernel



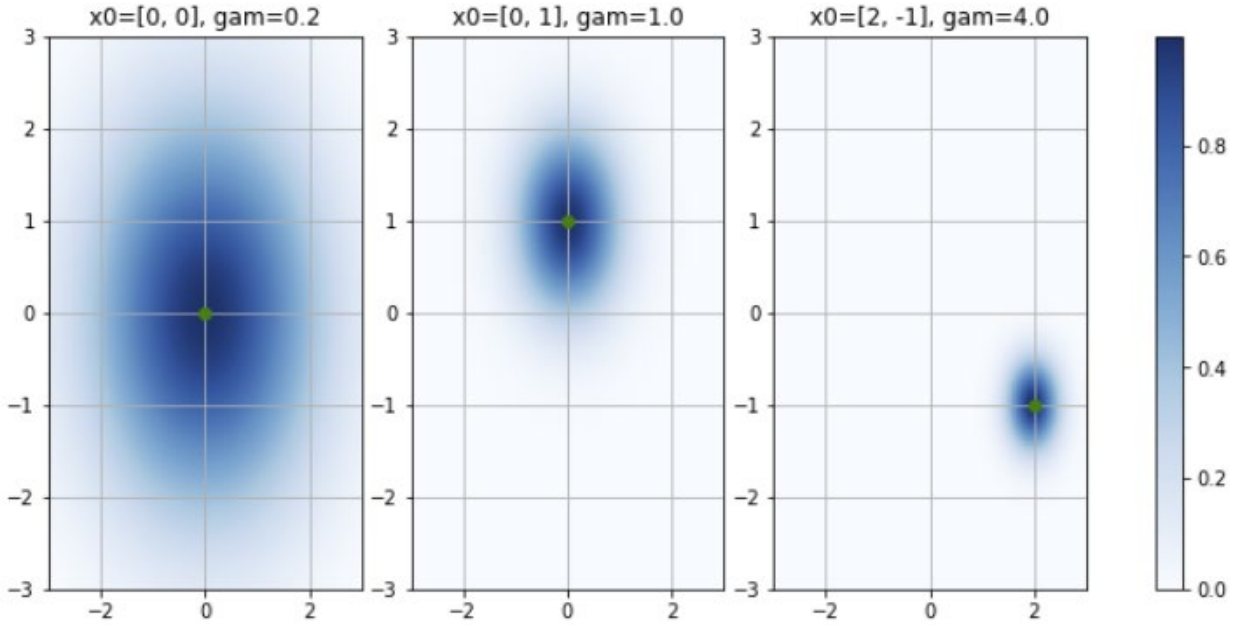
- Polynomial kernel: $K(x_i, x) = \left|x_i^T x\right|^d$
  - Typically d=2

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



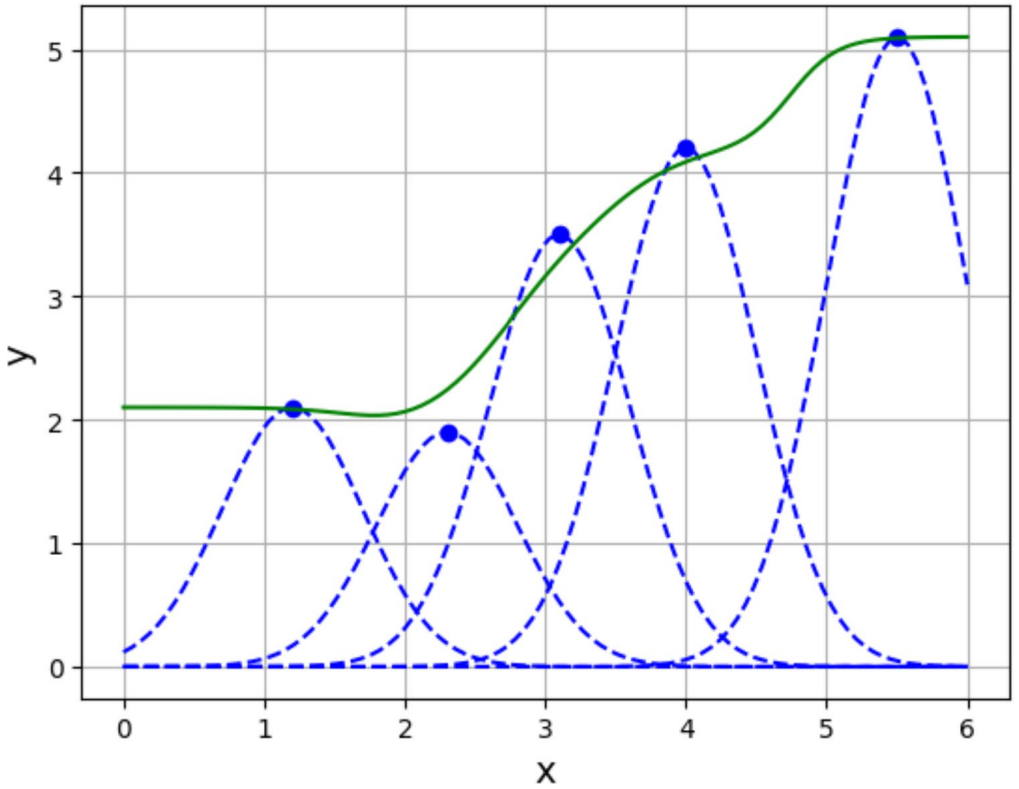
RBFs in 2D

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

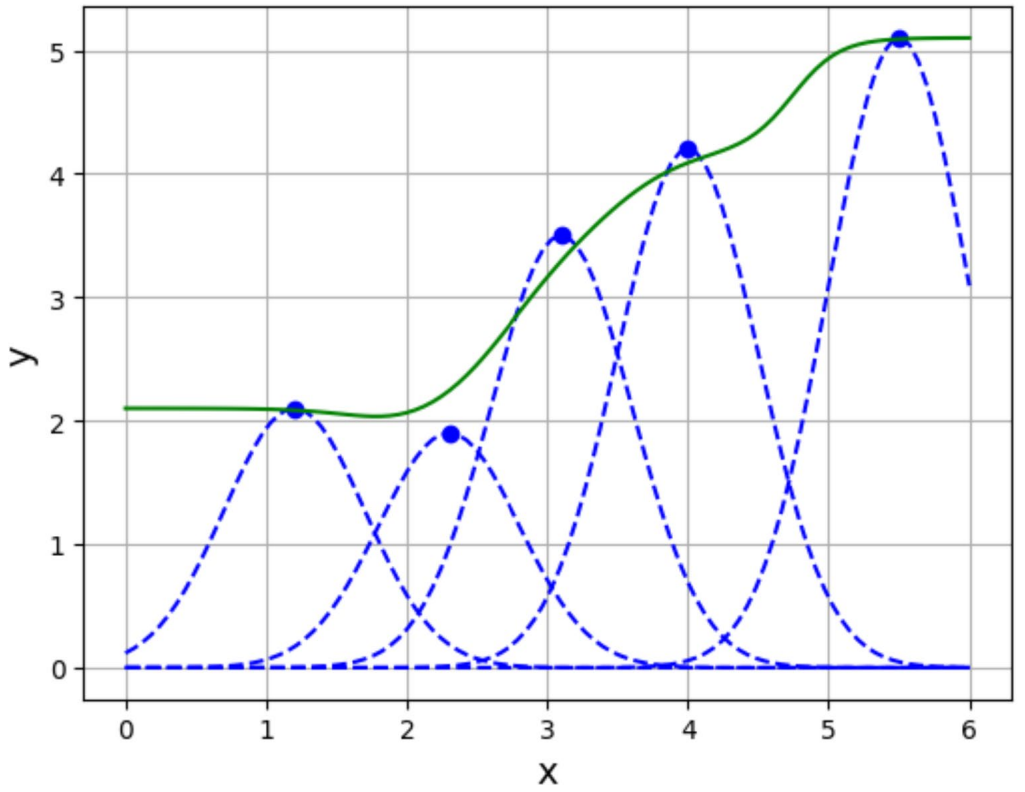


Kernel Smoothing as Weighted Sum

- 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 $K(x_0, x_i)$



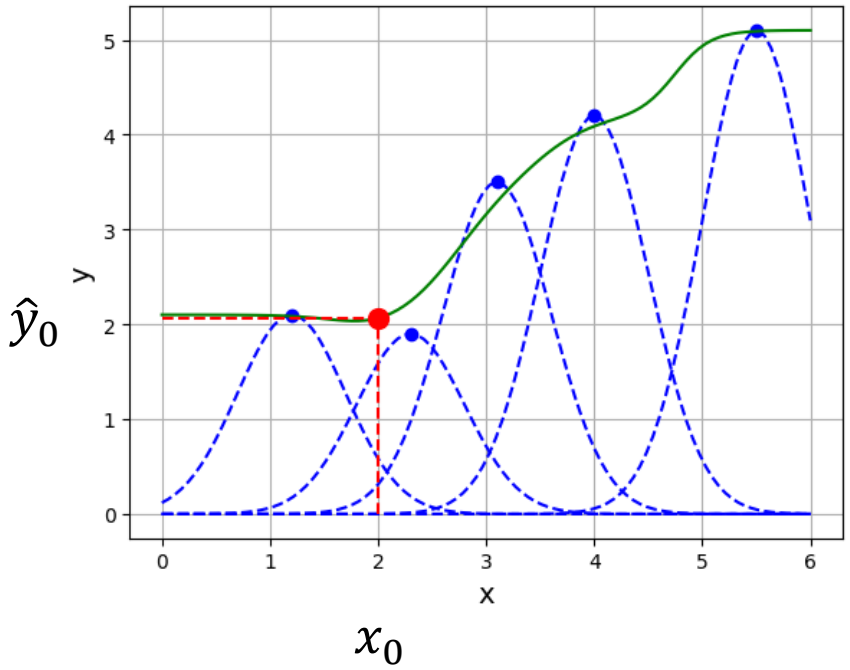
Example with RBF

Test point: $x_0 = 2$

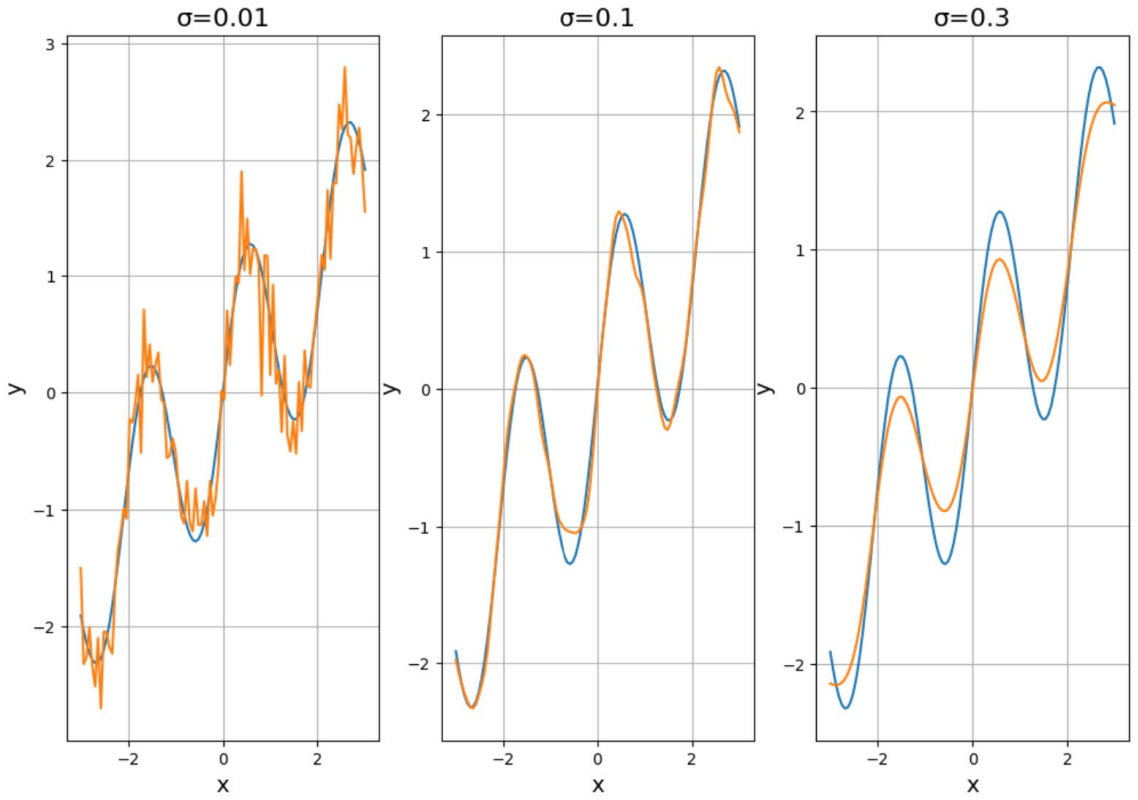
RBF kernel: $\gamma = \frac{1}{\sigma^2}, \sigma = 0.5$

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

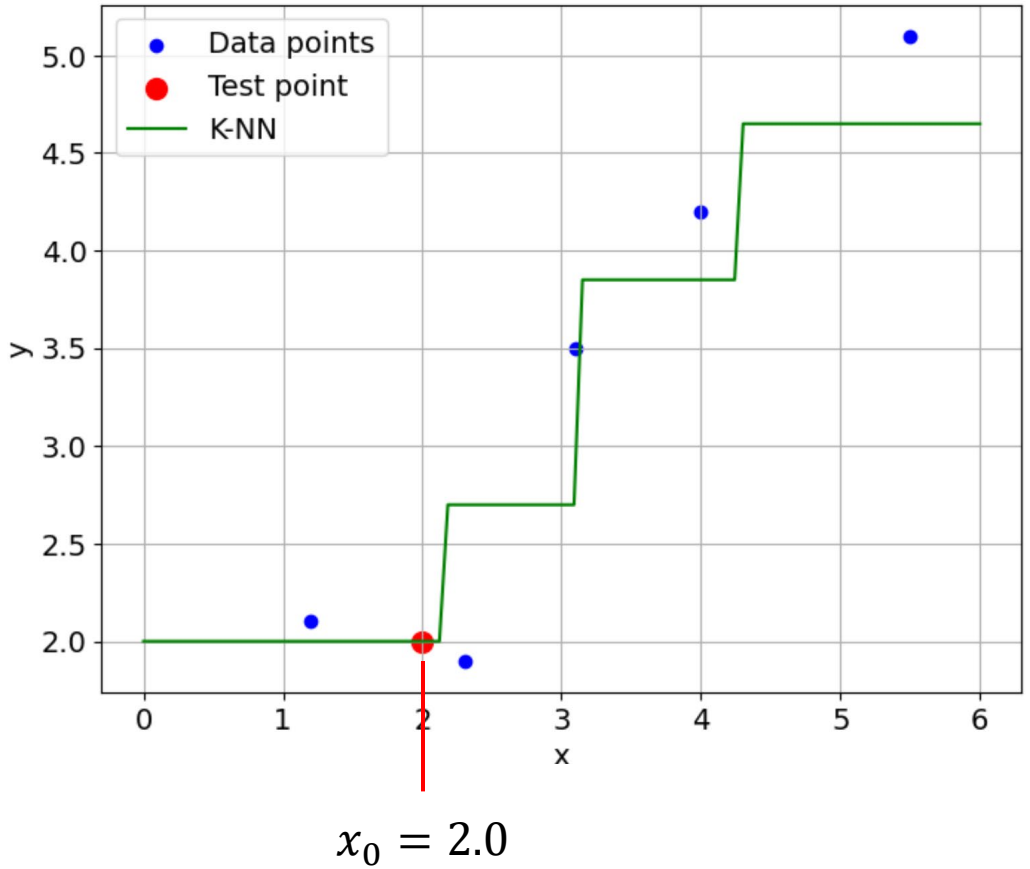
Kernel Smoother in Python

- 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

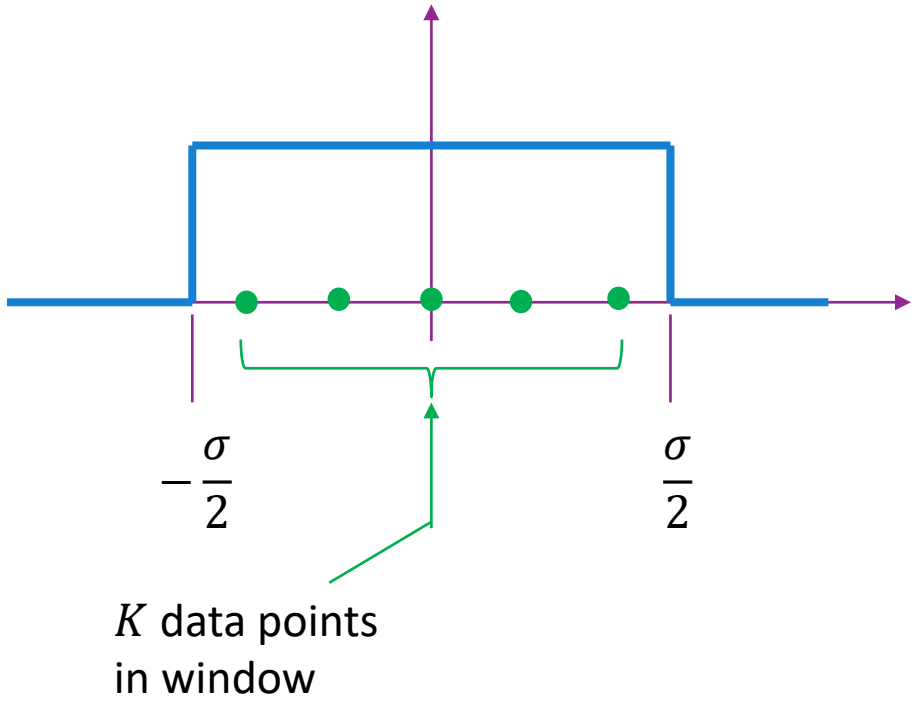
- Prediction: $\hat{y}_0 = \sum_i \alpha_i(x_0) y_i\,,\ y_i = f(x_i) + w_i$
- $E(\hat{y}_0) = \sum \alpha_i(x_0) f(x_i)$
- Bias error: $Bias(x_0) = f(x_0) - \sum \alpha_i(x_0) f(x_i)$
  - Bias will be large when $f(x)$ varies over kernel bandwidth
  - Increases with kernel radius
- Variance error: $Var(x_0)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

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



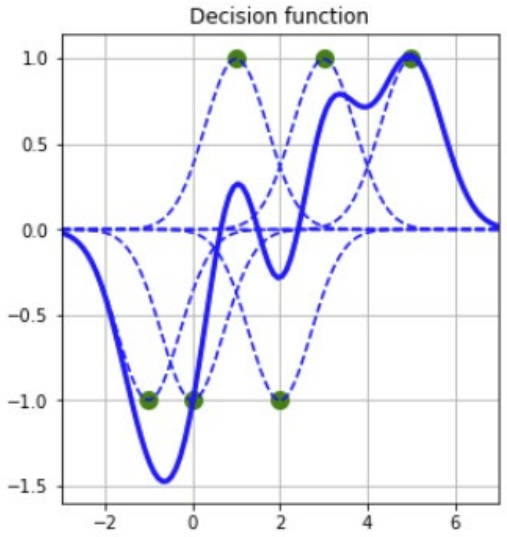
Kernel Classifier

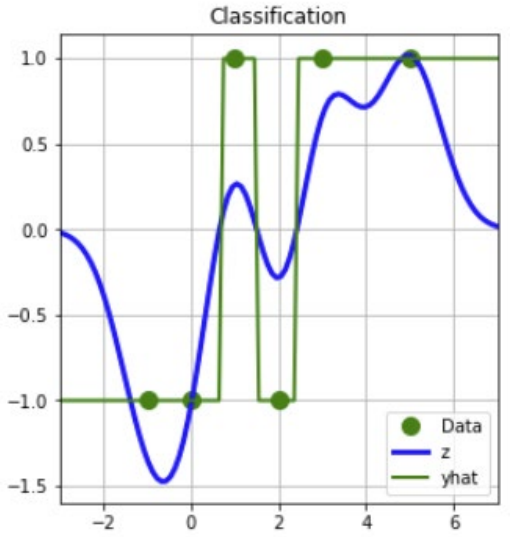
- Kernel can also be used for classification
- Given:
  - Training data (x_i, y_i) with binary labels $y_i = \pm 1$
  - Kernel $K(x_i, x)$
- To classify a new point x:
  - Decision function: $z = \sum_{i=1}^{n} y_i K(x_i, x)$
  - Classify: $\hat{y} = sign(z)$
- Idea:
  - z is large positive when x is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

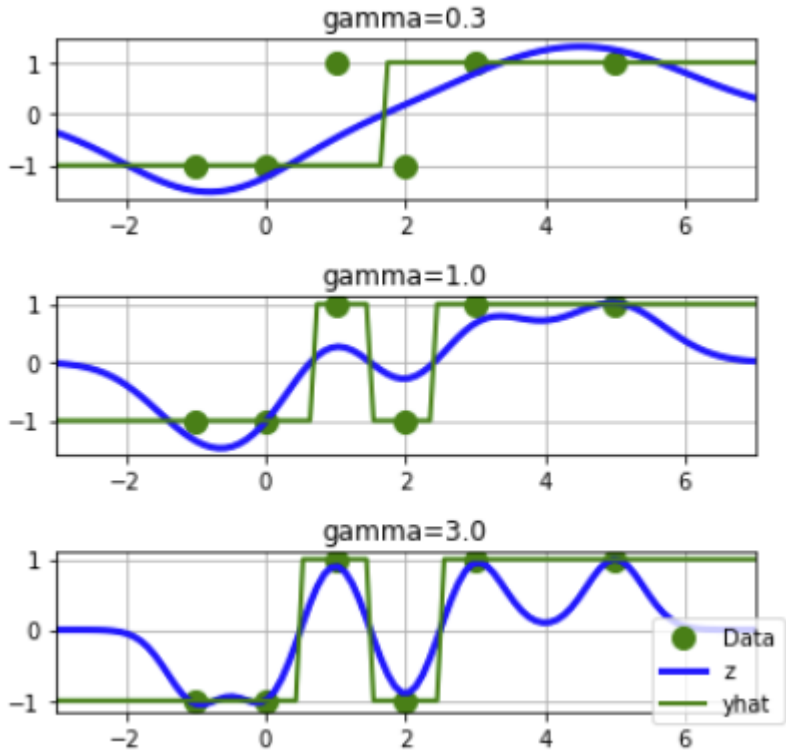
i	1	2	3	4	5	6
x_i	-1	0	1	2	3	5
y_i	-1	-1	1	-1	1	1





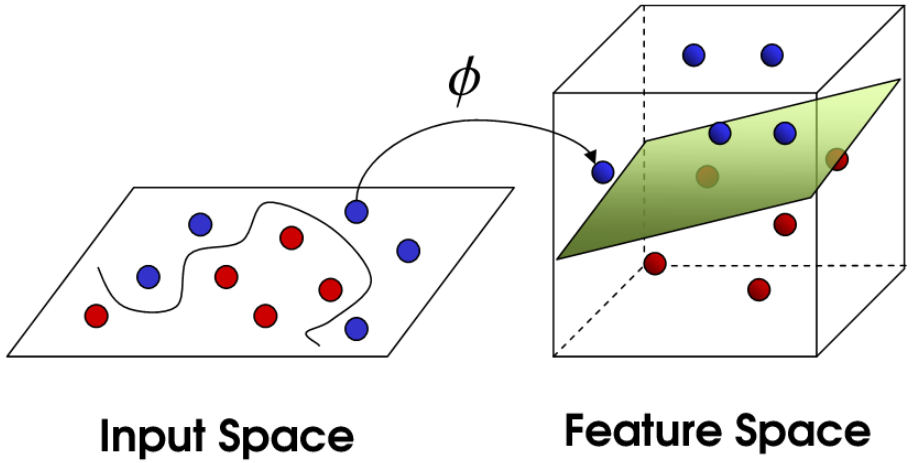
Effect of Gamma

- 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

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

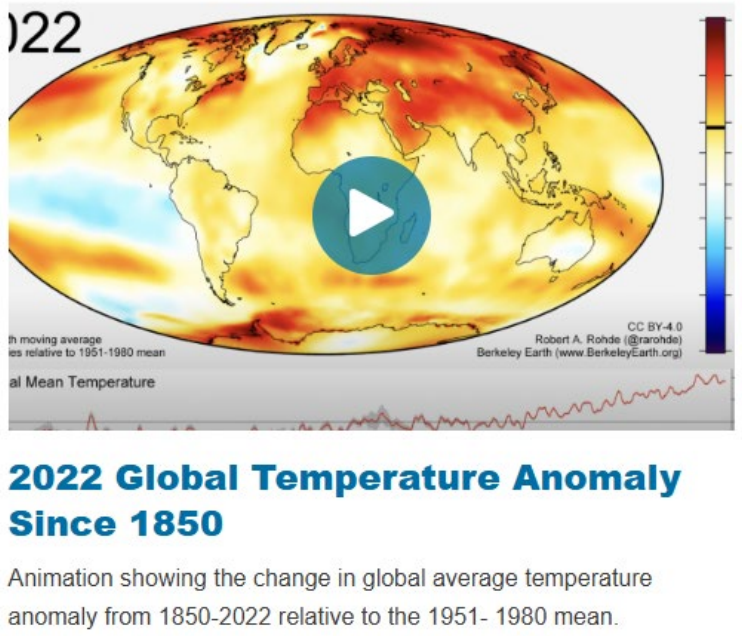
- All kernels are inner products in some transformed space
- The space may be infinite dimensional

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...
```

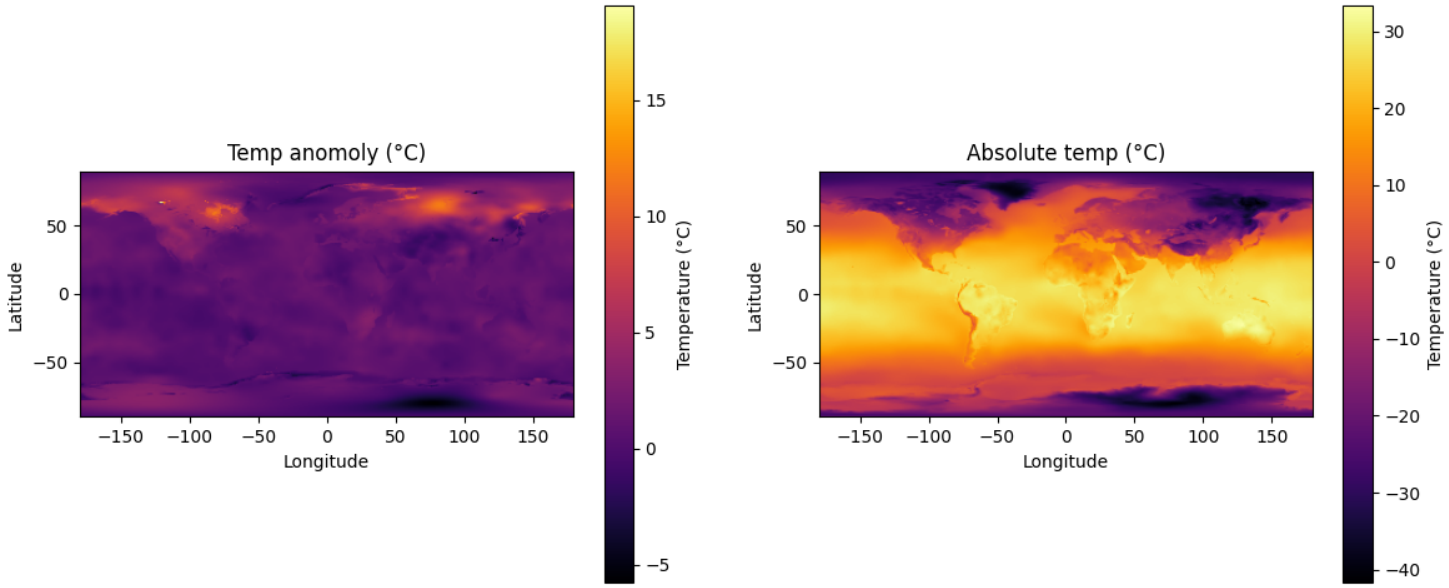
Global Gridded Data

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

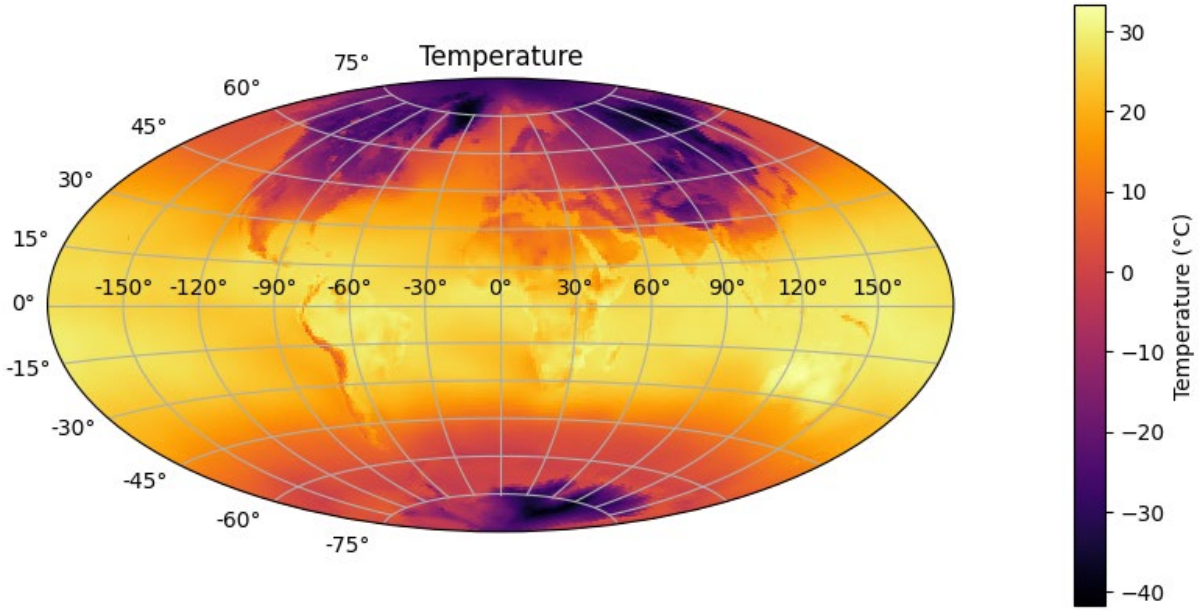
Projections

- 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

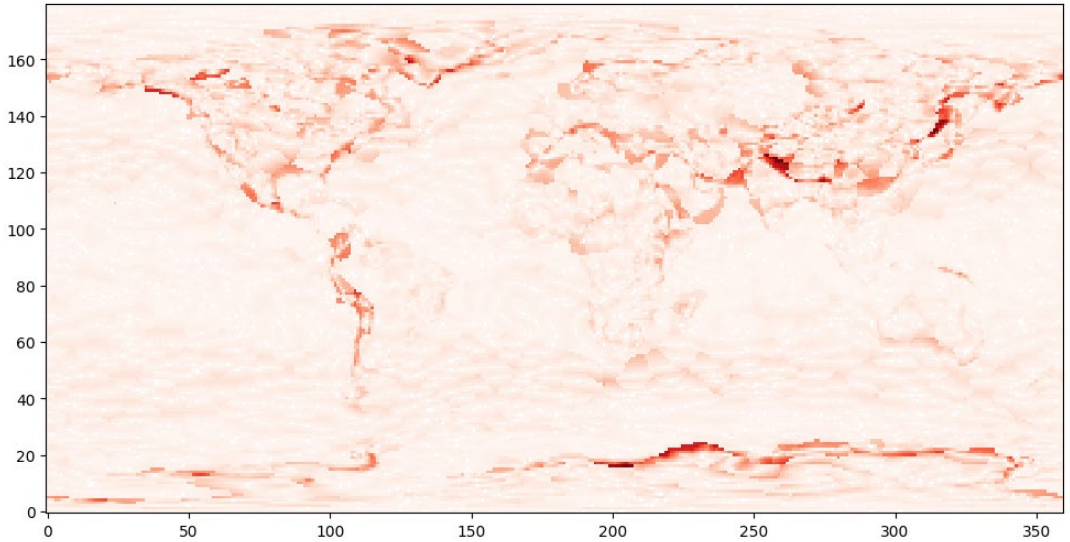
- Take 3% of the data for training
- Interpolate to remaining 90%
- Use (x, y, z) instead of lat-long for distance
  - Ensures distance is measured on sphere

```
# 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]
```

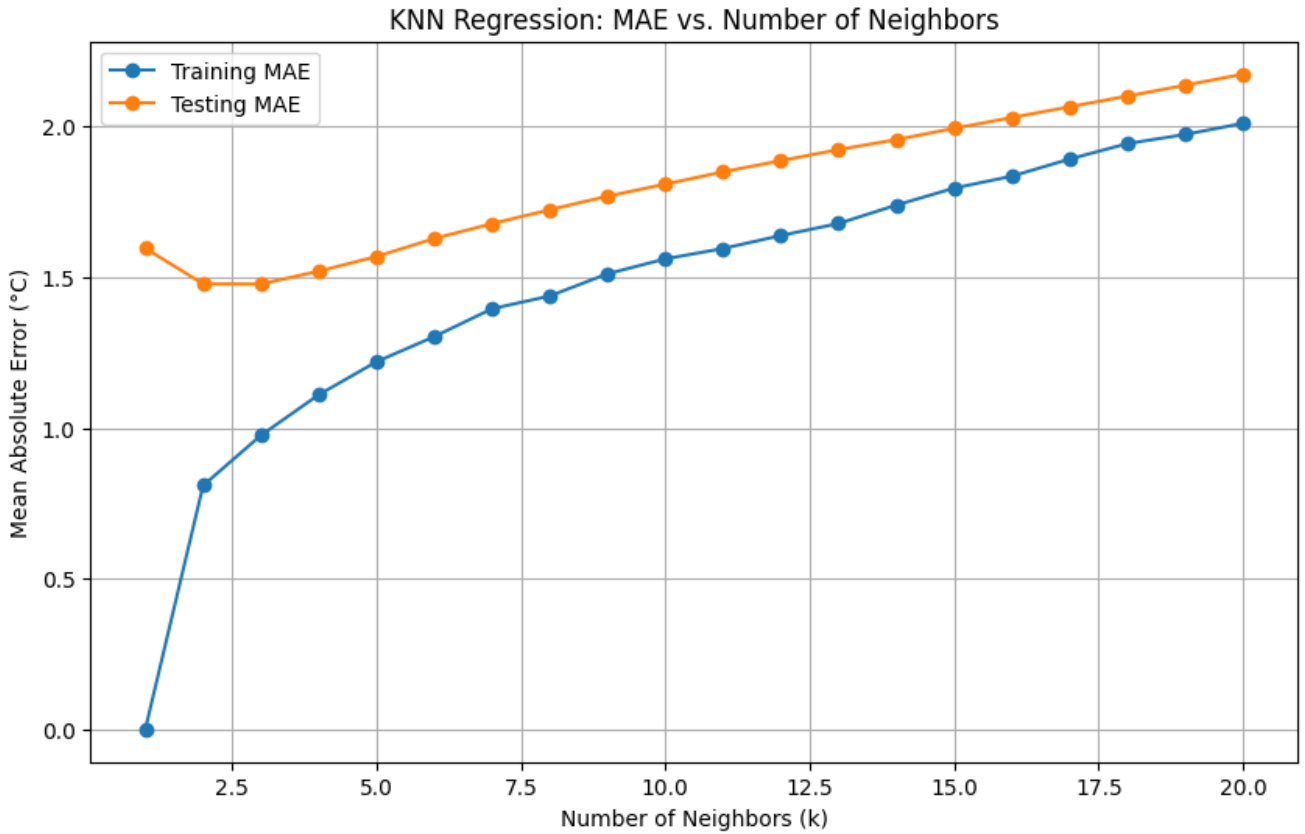
Interpolation with K-NN

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

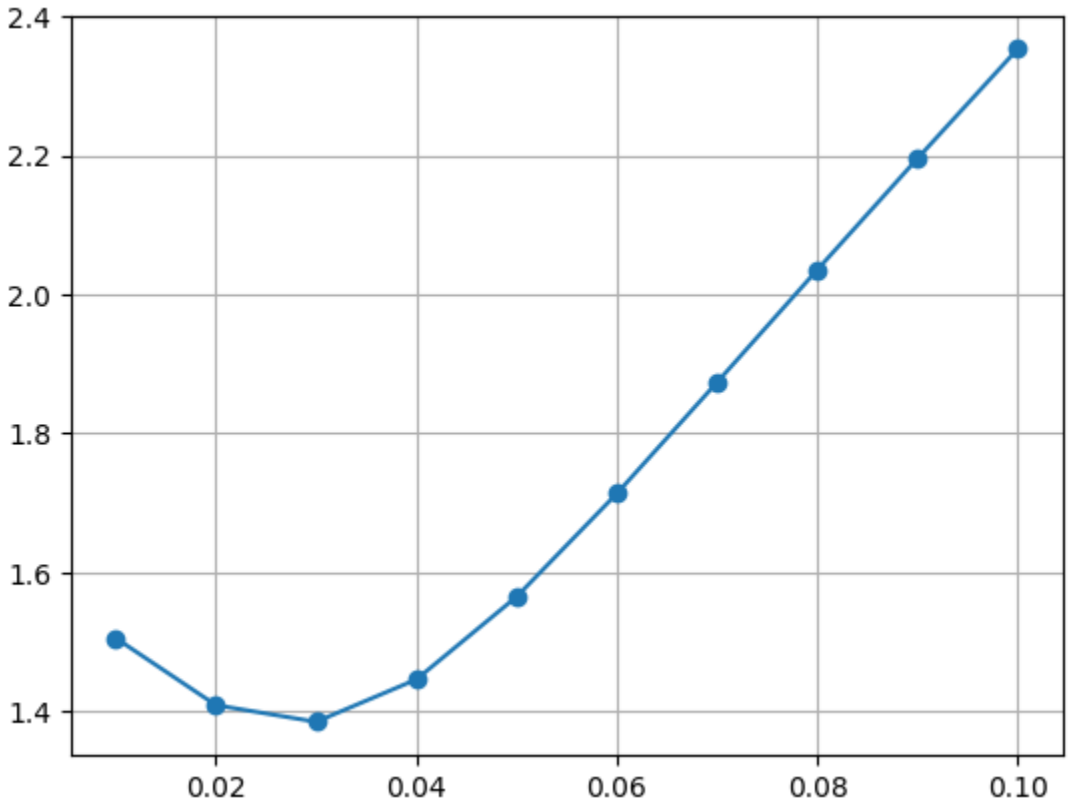


Selecting K

- $K = 3$ is optimal



Kernel Smoother



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