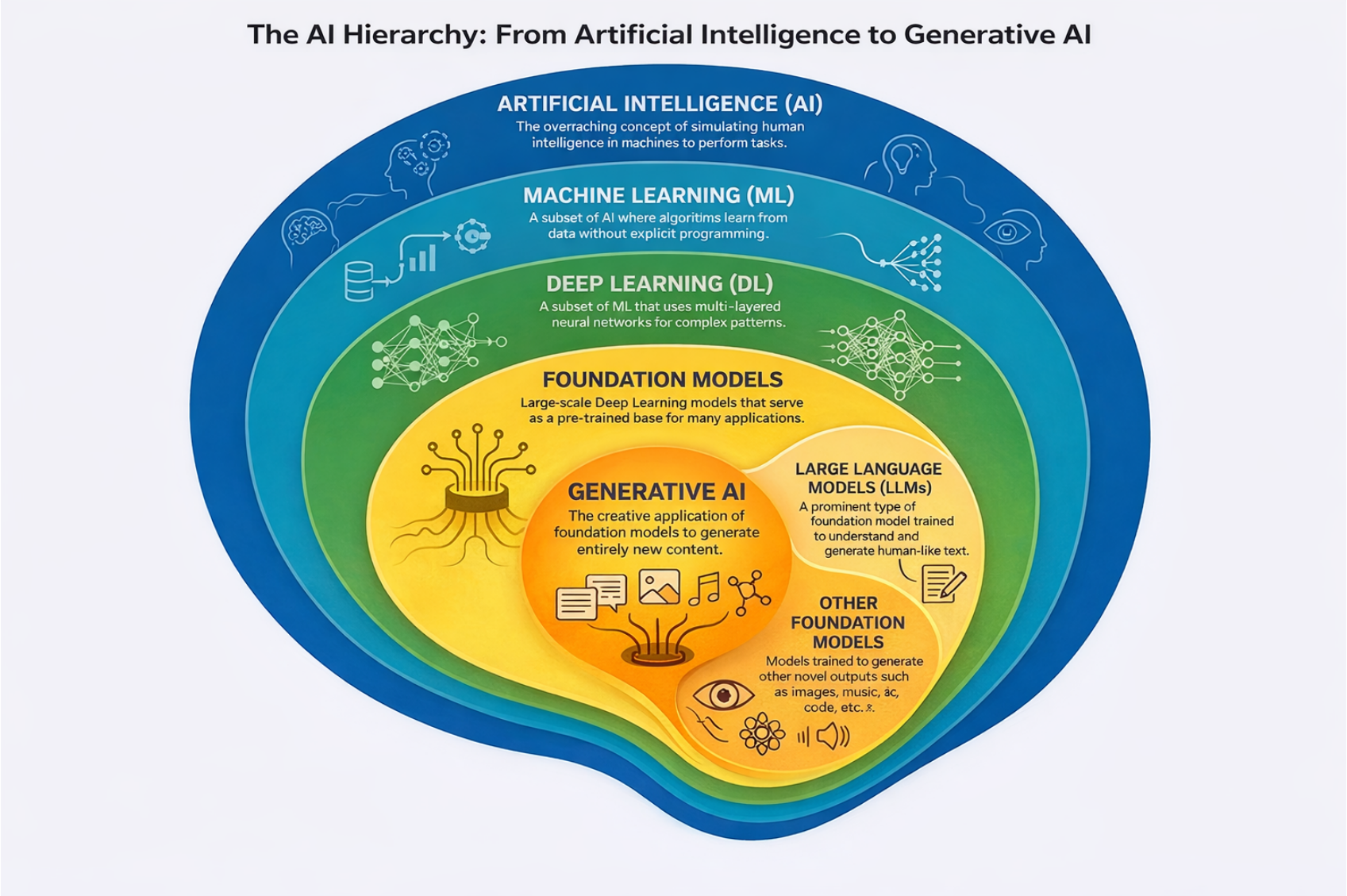
Lecture 7: Machine Learning I

K-Nearest Neighbors

Yen-Yi Ho

Department of Statistics

The World of Machine Learning



Major Types of Machine Learning Tasks

- Classification: Predicts a category or label.
- Regression: Predicts a continuous number.
- Clustering: Finds structure or groups in data without labels.
- Dimension Reduction: Compresses data while keeping important information.
- Anomaly / Outlier Detection: Finds unusual or rare cases.
- Reinforcement Learning: Learn by trial and error with rewards.

Major Types of Machine Learning Tasks

- **Classification: Predicts a category or label.**
- Regression: Predicts a continuous number.
- Clustering: Finds structure or groups in data without labels.
- Dimension Reduction: Compresses data while keeping important information.
- Anomaly / Outlier Detection: Finds unusual or rare cases.
- Reinforcement Learning: Learn by trial and error with rewards.

Gene Expression Profiles Predict Clinical Outcomes

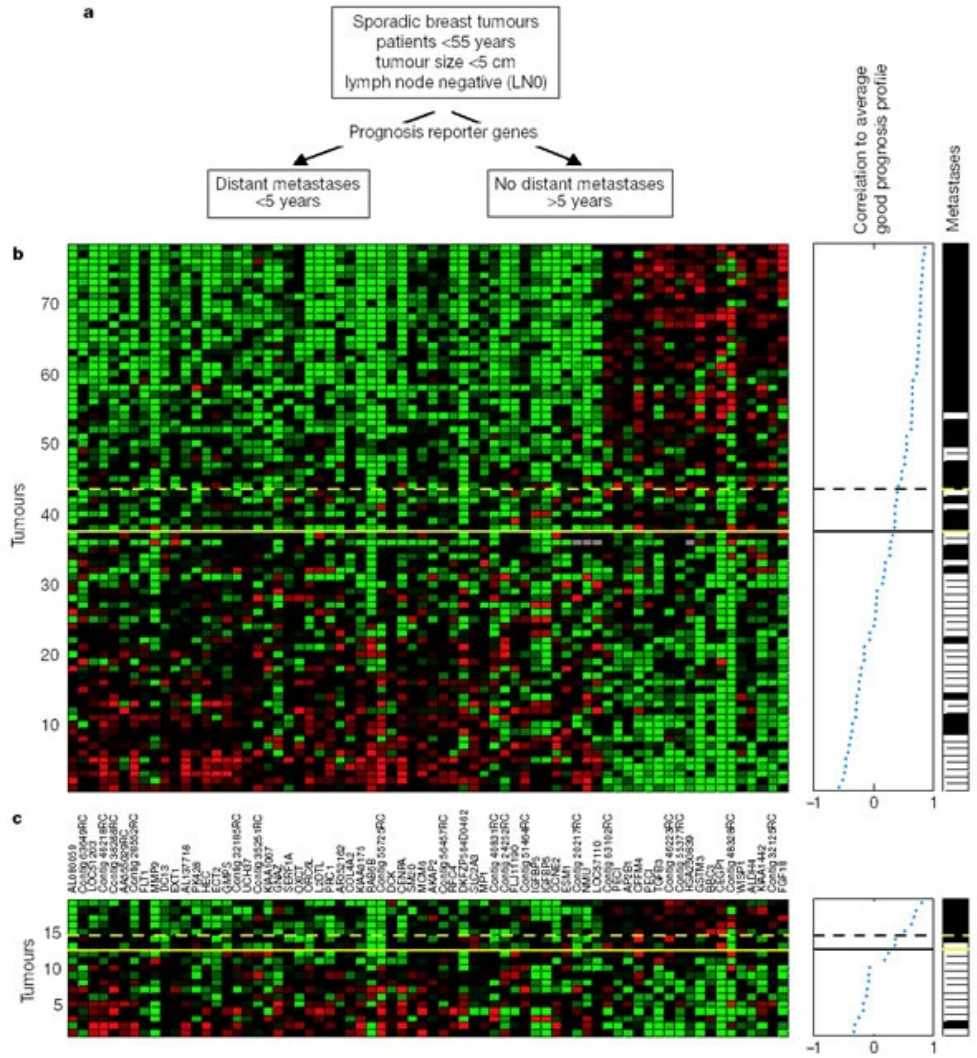


Figure adapted from van 't Veer, L., Dai, H., van de Vijver, M. et al. Gene expression profiling predicts clinical outcome of breast cancer. Nature 415, 530–536 (2002).

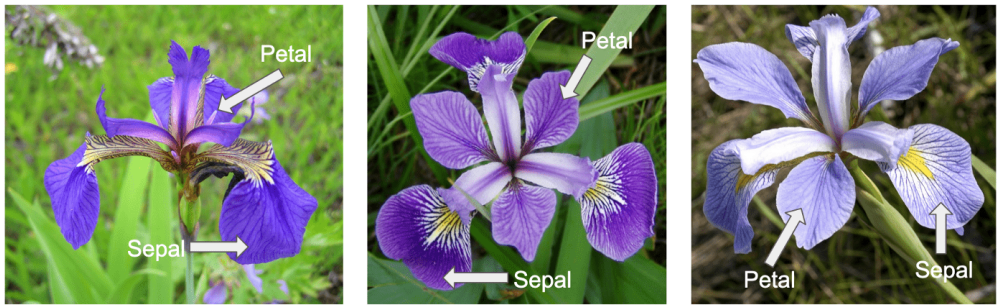
Classification

Two basic data-driven approaches for classification

- K-nearest neighbor
- Linear classifier

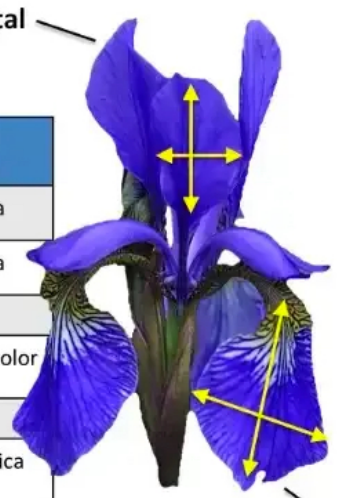
Classification Based on Features: The Iris Dataset

Iris setosa *Iris versicolor* *Iris virginica*



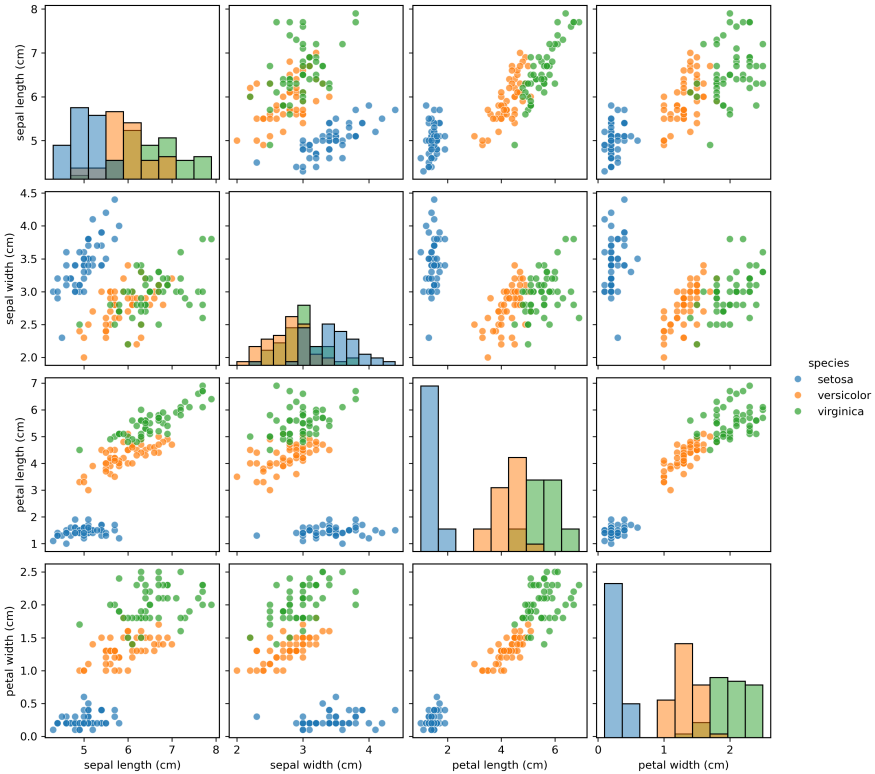
Samples
(instances, observations)

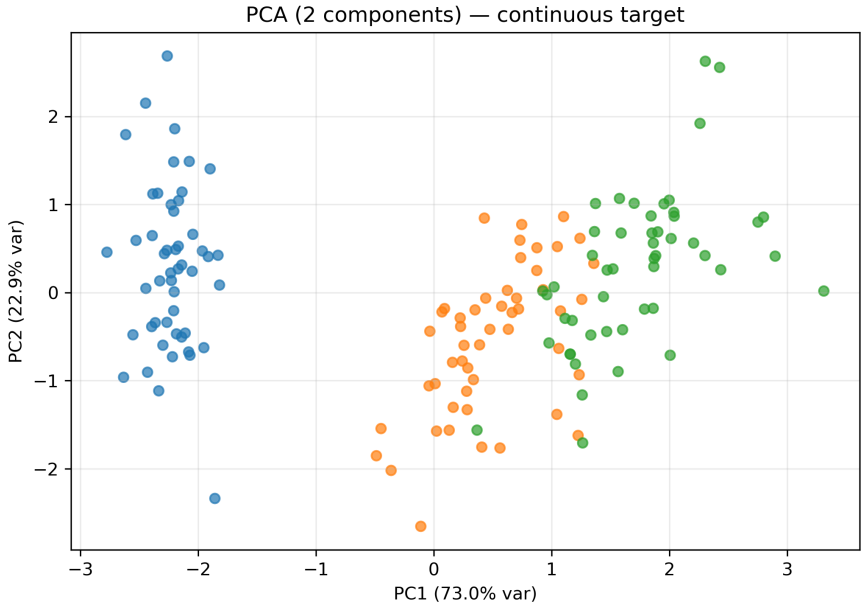
	Sepal length	Sepal width	Petal length	Petal width	Class label
1	5.1	3.5	1.4	0.2	Setosa
2	4.9	3.0	1.4	0.2	Setosa
...					
50	6.4	3.5	4.5	1.2	Versicolor
...					
150	5.9	3.0	5.0	1.8	Virginica



Features
(attributes, measurements, dimensions)

Class labels
(targets)


sepal length (cm)
sepal width (cm)
petal length (cm)
petal width (cm)
species
setosa
versicolor
virginica


PCA (2 components) — continuous target
PC1 (73.0% var)
PC2 (22.9% var)

Machine Learning: Data-Driven Approach

- Collect a genomic dataset and outcome of interest
- Use machine learning algorithms to train a classifier
- Evaluate the classifier on new data points

```
def train(data, labels)
        # Machine learning!
        return model
```

```
def predict(model, test_data)
        # Use model to predict labels
        return test_labels
```

Nearest Neighbor Classifier

```
def train(data, labels)
        """memorize all data and labels"""
        # Machine learning!
        return model
```

```
def predict(model, test_data)
        """
        Predict the test_label using the most similar
        training data point
        """
        # Use model to predict labels
        return test_labels
```

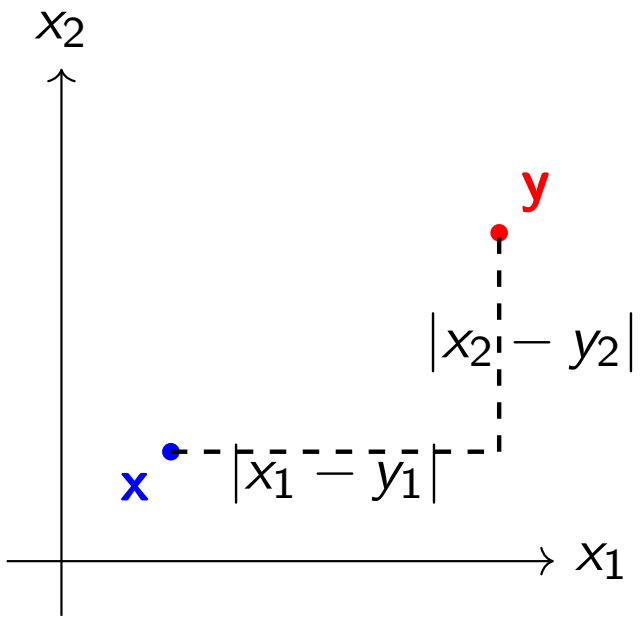
Distance Metric to Compare Data Points

Graphically: **horizontal** + **vertical** path. For example,

$$\mathbf{x} = (x_1, x_2), \qquad \mathbf{y} = (y_1, y_2).$$

The L1 (Manhattan) distance is

$$\|\mathbf{x} - \mathbf{y}\|_1 = |x_1 - y_1| + |x_2 - y_2|.$$



```python
import numpy as np

class NearestNeighbor:
  def _init_(self):
    pass

  def train(self, X, y):
    """ X is N x D where each row is a data point. Y is 1-dimension of size N """
    # the nearest neighbor classifier simply remembers all the training data
    self.Xtr = X
    self.ytr = y

  def predict(self, X):
    """ X is N x D where each row is a data point we wish to predict label for """
    num_test = X.shape[0]
    # lets make sure that the output type matches the input type
    Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

    # loop over all test rows
    for i in range(num_test):
      # find the nearest training image to the i'th test image
      # using the L1 distance (sum of absolute value differences)
      distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
      min_index = np.argmin(distances) # get the index with smallest distance
      Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

    return Ypred
```

```python
import numpy as np

class NearestNeighbor:
  def _init_(self):
    pass

  def train(self, X, y):
    """ X is N x D where each row is a data point. Y is 1-dimension of size N """
    # the nearest neighbor classifier simply remembers all the training data
    self.Xtr = X
    self.ytr = y

  def predict(self, X):
    """ X is N x D where each row is a data point we wish to predict label for """
    num_test = X.shape[0]
    # lets make sure that the output type matches the input type
    Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

    # loop over all test rows
    for i in range(num_test):
      # find the nearest training image to the i'th test image
      # using the L1 distance (sum of absolute value differences)
      distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
      min_index = np.argmin(distances) # get the index with smallest distance
      Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

    return Ypred
```

Q: With N samples, how fast are training and prediction?

```python
import numpy as np

class NearestNeighbor:
  def _init_(self):
    pass

  def train(self, X, y):
    """ X is N x D where each row is a data point. Y is 1-dimension of size N """
    # the nearest neighbor classifier simply remembers all the training data
    self.Xtr = X
    self.ytr = y

  def predict(self, X):
    """ X is N x D where each row is a data point we wish to predict label for """
    num_test = X.shape[0]
    # lets make sure that the output type matches the input type
    Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

    # loop over all test rows
    for i in range(num_test):
      # find the nearest training image to the i'th test image
      # using the L1 distance (sum of absolute value differences)
      distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
      min_index = np.argmin(distances) # get the index with smallest distance
      Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

    return Ypred
```

Q: With N samples, how fast are training and prediction?

Ans: Train O(1),

```python
import numpy as np

class NearestNeighbor:
  def _init_(self):
    pass

  def train(self, X, y):
    """ X is N x D where each row is a data point. Y is 1-dimension of size N """
    # the nearest neighbor classifier simply remembers all the training data
    self.Xtr = X
    self.ytr = y

  def predict(self, X):
    """ X is N x D where each row is a data point we wish to predict label for """
    num_test = X.shape[0]
    # lets make sure that the output type matches the input type
    Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

    # loop over all test rows
    for i in range(num_test):
      # find the nearest training image to the i'th test image
      # using the L1 distance (sum of absolute value differences)
      distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
      min_index = np.argmin(distances) # get the index with smallest distance
      Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

    return Ypred
```

Q: With N samples, how fast are training and prediction?
Ans: Train O(1), predict O(N).

```python
import numpy as np

class NearestNeighbor:
  def _init_(self):
    pass

  def train(self, X, y):
    """ X is N x D where each row is a data point. Y is 1-dimension of size N """
    # the nearest neighbor classifier simply remembers all the training data
    self.Xtr = X
    self.ytr = y

  def predict(self, X):
    """ X is N x D where each row is a data point we wish to predict label for """
    num_test = X.shape[0]
    # lets make sure that the output type matches the input type
    Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

    # loop over all test rows
    for i in range(num_test):
      # find the nearest training image to the i'th test image
      # using the L1 distance (sum of absolute value differences)
      distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
      min_index = np.argmin(distances) # get the index with smallest distance
      Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

    return Ypred
```

Q: With N samples, how fast are training and prediction?

Ans: Train O(1), predict O(N).

This is bad:

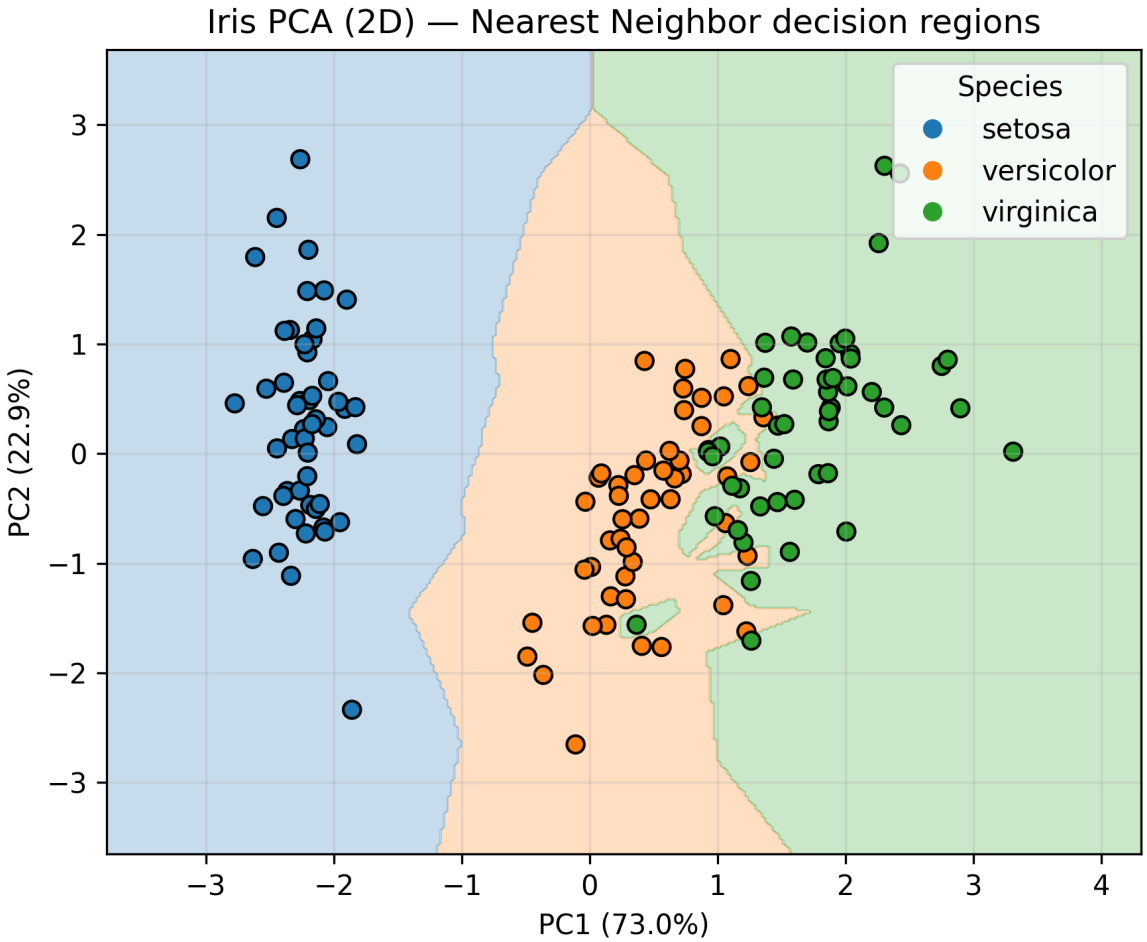
```python
import numpy as np

class NearestNeighbor:
  def _init_(self):
    pass

  def train(self, X, y):
    """ X is N x D where each row is a data point. Y is 1-dimension of size N """
    # the nearest neighbor classifier simply remembers all the training data
    self.Xtr = X
    self.ytr = y

  def predict(self, X):
    """ X is N x D where each row is a data point we wish to predict label for """
    num_test = X.shape[0]
    # lets make sure that the output type matches the input type
    Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

    # loop over all test rows
    for i in range(num_test):
      # find the nearest training image to the i'th test image
      # using the L1 distance (sum of absolute value differences)
      distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
      min_index = np.argmin(distances) # get the index with smallest distance
      Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

    return Ypred
```

Q: With N samples, how fast are training and prediction?

Ans: Train O(1), predict O(N).

This is bad: We want classifiers that are fast at prediction; slow for training is ok.

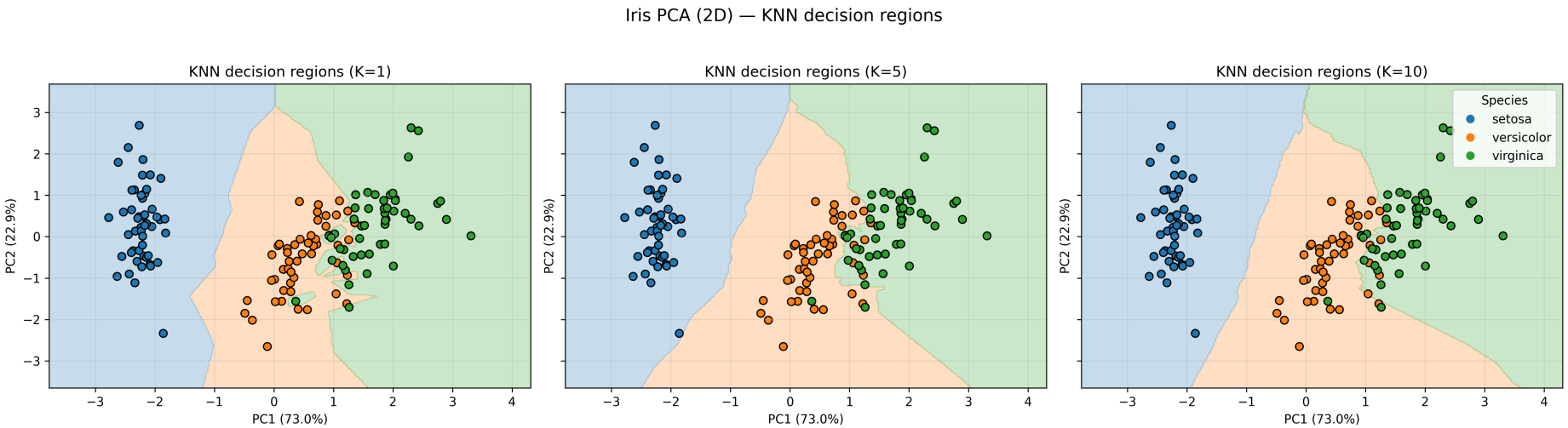
What Does Nearest Neighbor Look like



1-nearest neighbor: notice the outlier data points (island)

K-Nearest Neighbors

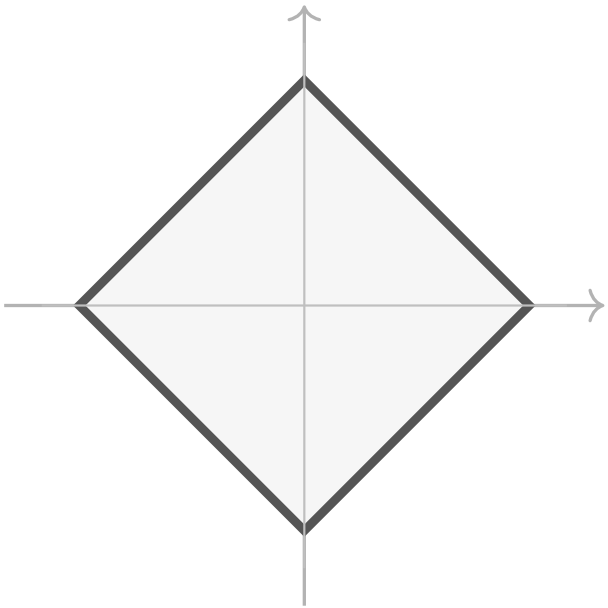
Instead of copying label from nearest neighbor, take majority vote from K closet data points



K-Nearest Neighbors: Distance Metric

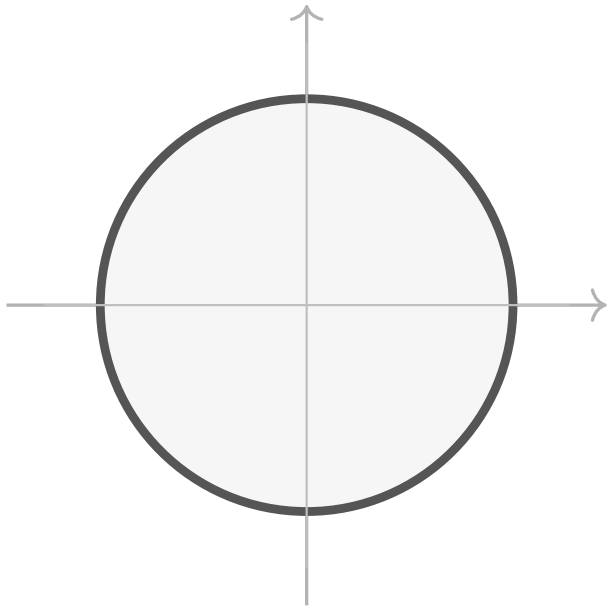
L1 (Manhattan) distance

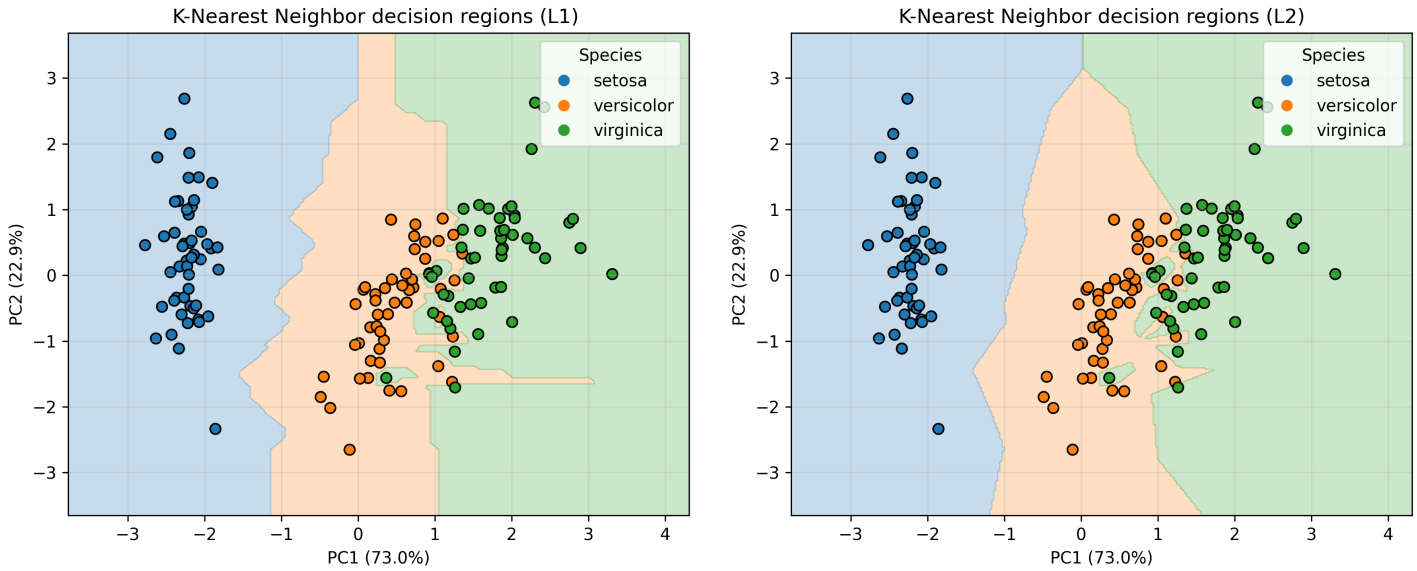
$$d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$$



L2 (Euclidean) distance

$$d_2(I_1, I_2) = \sqrt{\sum_p (I_1^p - I_2^p)^2}$$





- L1: Decision boundaries tend to be piecewise linear
- L1: Prefers solutions aligned with coordinate axes
- L2: Decision boundaries tend to be smooth and curved
- L2: Sensitive to outliers (large deviations in any direction)

Sparse versus Distributed Difference

Sparse difference: (10, 0), (0, 0)
Distributed difference: (7, 7), (0, 0)

Vector	**L1 distance**	**L2 distance**
(10, 0)	10	10
(7, 7)	14	≈ 9.9

L2 fails when

- Data is high-dimensional
- Signal is sparse & noise is spread everywhere

When small noise appears in many coordinates $\rightarrow$ L2 adds it up $\rightarrow$ everything looks similar $\rightarrow$ Nearest neighbors collapse

L1 versus L2

- L1 is good when
  - Few features matter
  - Sparse signal stands out
  - Distributed noise becomes expensive
  - Example: genomic data (few gene matter), ...
- L2 is good when
  - Small changes in many coordinates are meaningful
  - Large changes in one coordinate are often noise or artifacts
  - L2 suppresses spikes and favors smooth variation.
  - Example: Images, audio signals,...

High Dimensions: Why L1 and L2 Behave Differently

Consider two difference vectors in $\mathbb{R}^d$:

- **Sparse signal (1 feature):**

$$\mathbf{s} = (10, 0, 0, \ldots, 0)$$

- **Distributed noise (d features):**

$$\mathbf{n} = \left(\frac{10}{\sqrt{d}}, \frac{10}{\sqrt{d}}, \ldots, \frac{10}{\sqrt{d}}\right).$$

L2 distance (Euclidean)

$$\|\mathbf{s}\|_2 = \sqrt{10^2} = 10, \qquad \|\mathbf{n}\|_2 = \sqrt{d \cdot \left(\frac{10}{\sqrt{d}}\right)^2} = 10$$

Sparse signal and distributed noise collapse to the same distance.

L1 distance (Manhattan).

$$\|\mathbf{s}\|_1 = 10, \qquad \|\mathbf{n}\|_1 = d \cdot \frac{10}{\sqrt{d}} = 10\sqrt{d}$$

Distributed noise grows with dimension and becomes distinguishable.

Hyperparameters

- What is the best value of K to use?
- What is the best distance to use?

Hyperparameters are sometimes dataset/problem-dependent.

Setting Hyperparameters

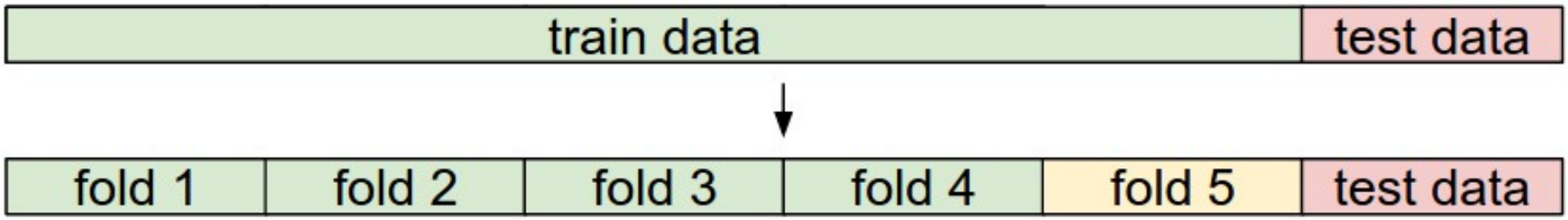
Idea 1:

Idea 2:

Idea 3:

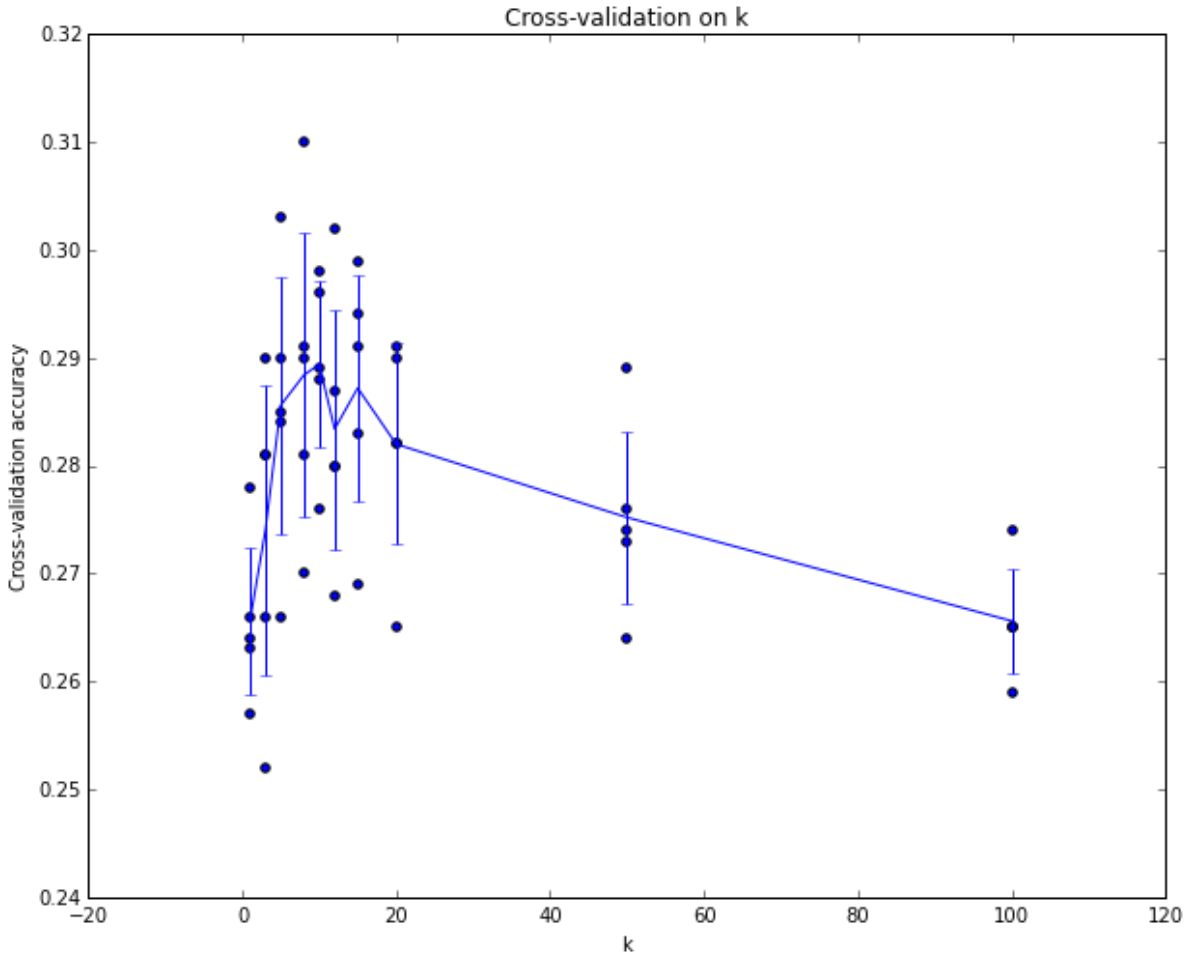
Setting Hyperparameters: Cross-Validation

Idea 4: Split your training set into folds of training sets and a validation set. Use validation set to tune all hyperparameters. At the end run a single time on the test set and report performance.



This is often advised. Useful for small datasets. However, in large-scale deep learning, it is less-practiced.

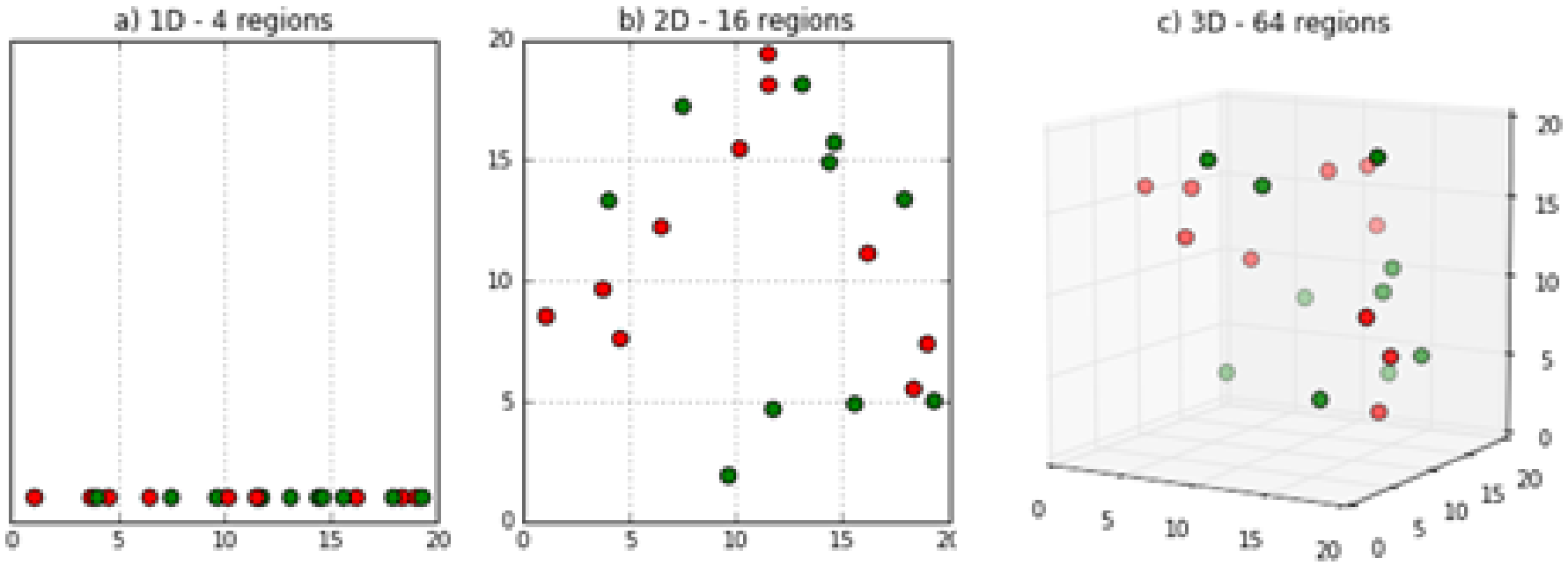
Example of 5-fold cross-validation for K



K-Nearest Neighbors: Summary

- The K-Nearest Neighbors classifier predicts labels based on the **K nearest training data points**
- Distance metric and K are hyperparameters
- Choose hyperparameters using the validation set
- Only run on the test set once at the very end!

The Curse of Dimensionality



The Curse of Dimensionality

Next: Linear Classifier