K-Nearest Neighbors (and Evaluating ML Methods)

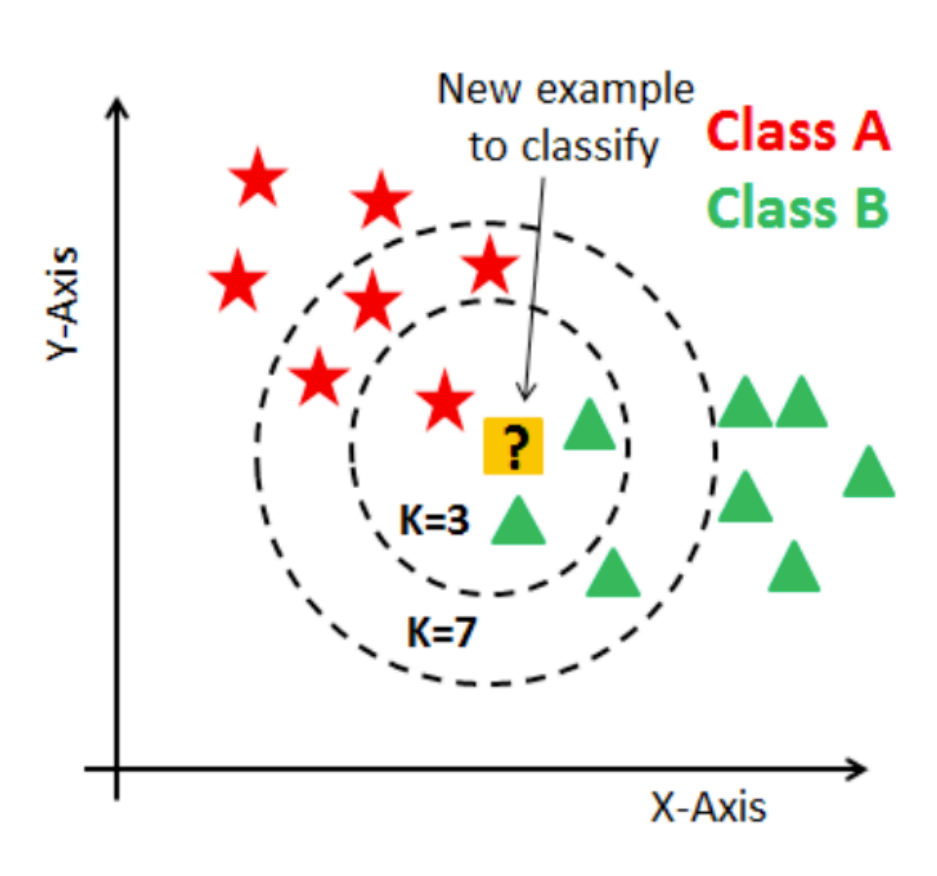
David I. Inouye

Outline

- K-Nearest Neighbors (KNN) simple algorithm
- Evaluating methods (i.e., generalization error)
  - Train vs test data
- Cross validation
- Hyperparameter tuning (choosing k)
- Curse of dimensionality

K-Nearest Neighbors (KNN) is a Very Simple and Intuitive Supervised Learning Algorithm

1. Find the k nearest neighbors
   - Equivalently, expand circle until it includes k points
2. Select most common class



https://www.datacamp.com/community/tutorials/k-nearest-neighbor-classification-scikit-learn

Naïve KNN Requires Distance to **All Training Points**

- **Input:** Test point x_0, training data $\{x_i, y_i\}_{i=1}^{n}$
- **Output:** Predicted class y_0

1. Compute distance to all training points:

$$d_i = d(x_0, x_i), \forall i$$

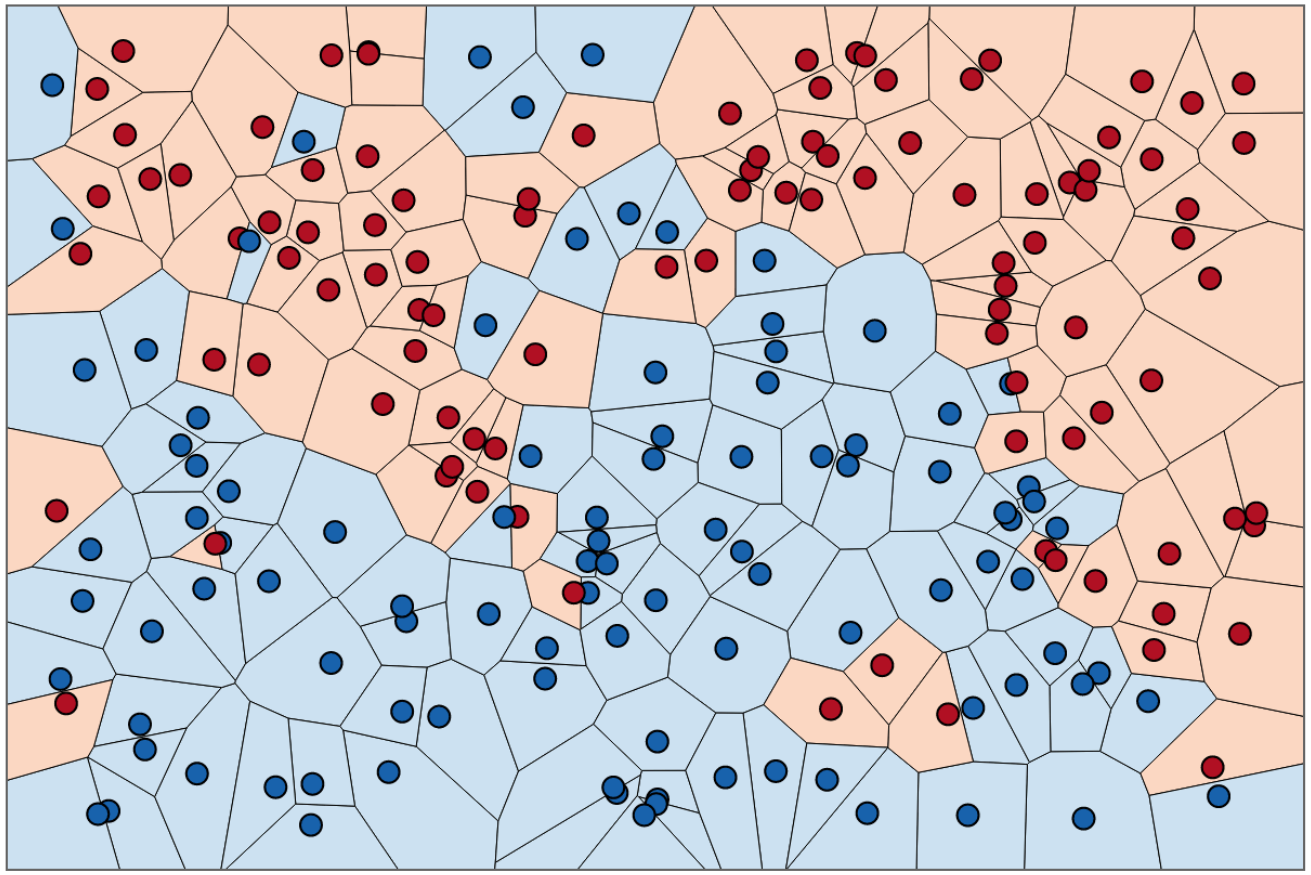
2. Sort distances where π is a permutation (e.g., $\pi(1)$ is the index of the closest point):

$$d_{\pi(1)} \leq d_{\pi(2)} \leq \cdots \leq d_{\pi(n)}$$

3. Return the most common class of the top k:

$$y_0 = \text{mode}\{y_{\pi(j)}\}_{j=1}^{k}$$

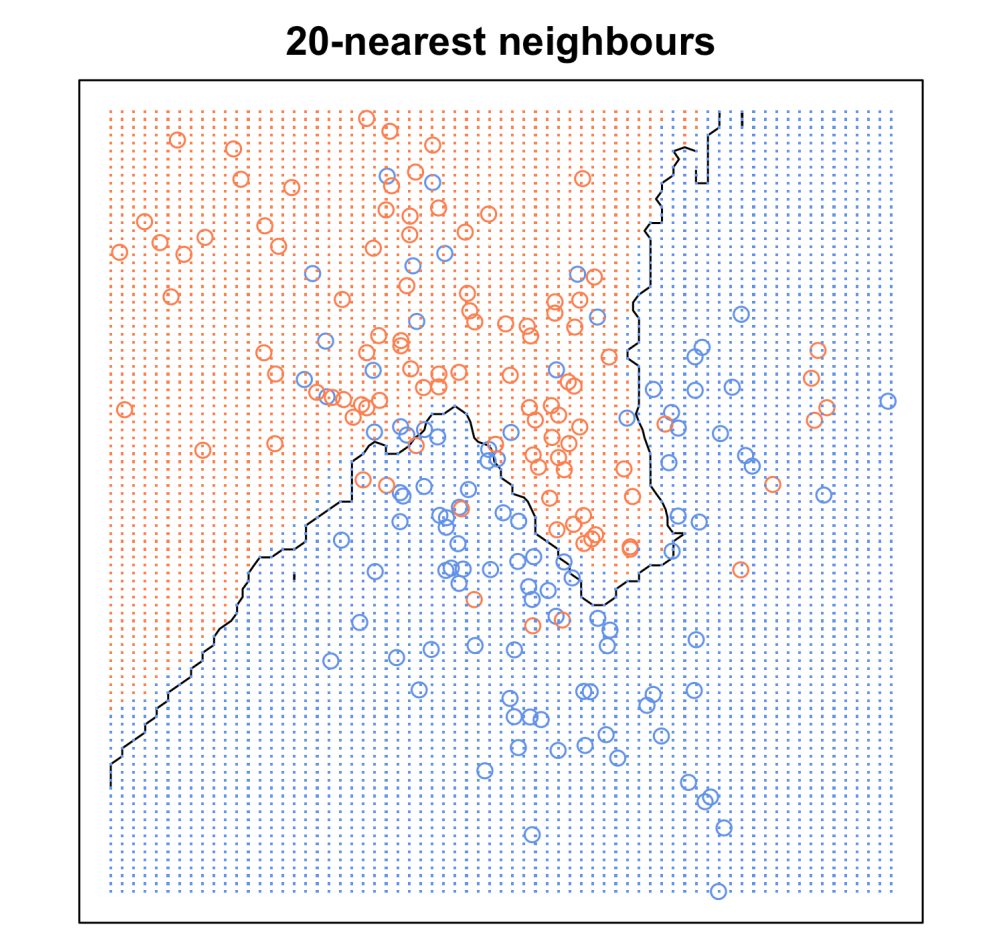
1-NN Partitions the Space Into Voronoi Cells Based on the Training Data



http://scott.fortmann-roe.com/docs/BiasVariance.html

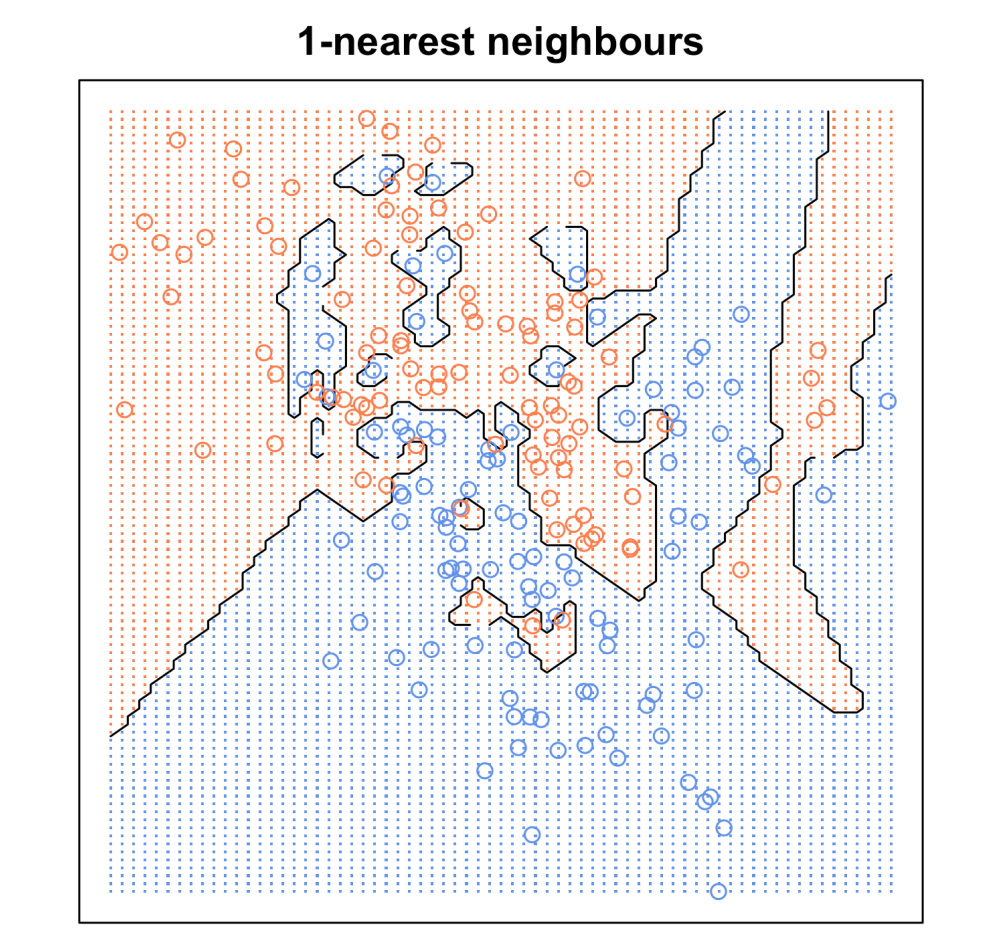
The KNN Boundary Gets "Smoother" as k Increases

1-nearest neighbours



https://kevinzakka.github.io/2016/07/13/k-nearest-neighbor/

20-nearest neighbours



How Should Method Performance Be Estimated?

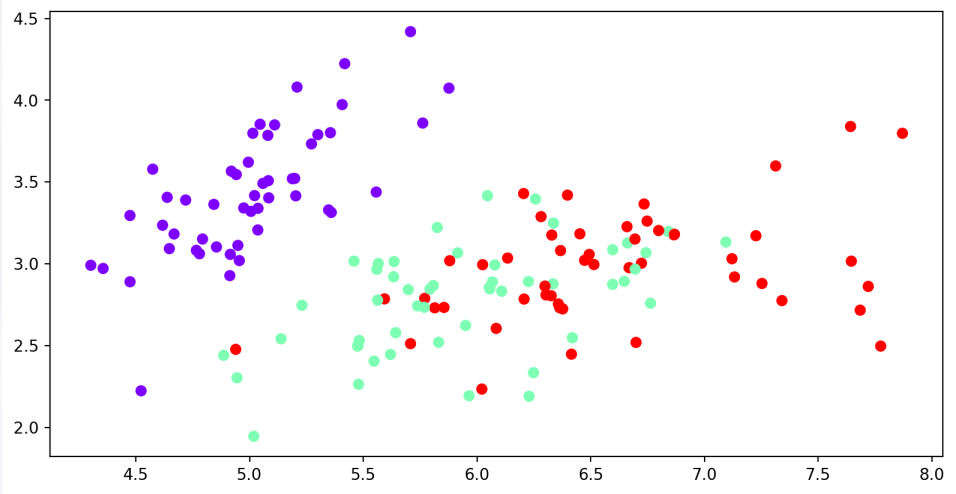
- We are going to look at a KNN demo to investigate this question.

KNN Demo: Load some toy data

```
import numpy as np
import scipy.stats
import matplotlib.pyplot as plt
from sklearn import neighbors, datasets
from sklearn.metrics import pairwise_distances
from sklearn.utils import shuffle

# import some data to play with
iris = datasets.load_iris()
# we only take the first two features
X = iris.data[:, :2]
# Add random noise since iris has exact values
X = X + 0.05 * np.random.RandomState(0).randn(*X.shape)
y = iris.target
X, y = shuffle(X, y, random_state=0)
plt.scatter(X[:, 0], X[:, 1], c=y, cmap='rainbow')
print(X.shape)
```

```
(150, 2)
```



KNN Demo: Define the KNN Classifier

```python
class SimpleKNNClassifier():
    def __init__(self, X_train, y_train, k=1):
        self.X_train = X_train
        self.y_train = y_train
        self.k = k

    def predict(self, X):
        # 1. Compute distances
        D = np.nan * np.ones((X.shape[0], self.X_train.shape[
        for i, x in enumerate(X):
            for j, xt in enumerate(self.X_train):
                D[i, j] = np.linalg.norm(x-xt)

        # 2. Get the indices of the top k smallest distances
        sorted_idx = np.argsort(D, axis=1)

        # For each data point get mode
        y = np.array([
            scipy.stats.mode(self.y_train[sidx[:self.k]], kee
            for sidx in sorted_idx
        ])
        return y
```

KNN Demo: Define the Vectorized KNN Classifier

```python
class SimpleKNNClassifier():
    def __init__(self, X_train, y_train, k=1):
        self.X_train, self.y_train, self.k = X_train, y_train, k

    def predict(self, X):
        # 1. Compute distances
        D = pairwise_distances(X, self.X_train, metric='euclidean')

        # 2. Get the indices of the top k smallest distances
        sorted_idx = np.argsort(D, axis=1)

        # Faster vectorized version
        y_ind = self.y_train[sorted_idx[:,:self.k]]
        y = scipy.stats.mode(y_ind, axis=1, keepdims=False)[0]
        return y

for k in [1, 3, 10]:
    knn = SimpleKNNClassifier(X, y, k=k)
    y_pred = knn.predict(X)
    accuracy = np.mean(y == y_pred)
    print(f'The accuracy on the training data for k={k} is: {accuracy*100:.1f}%')
```

```
The accuracy on the training data for k=1 is: 100.0%
The accuracy on the training data for k=3 is: 85.3%
The accuracy on the training data for k=10 is: 85.3%
```

- Given this, what do you think is the best k value? — Discuss with your partner.

KNN Demo: Suppose we only had 100 points for training and then received 50 new flower measurments

```python
# Use first 100 points
X_train = X[:100,:]
y_train = y[:100]

# Setup model
for k in [1, 3, 5, 7, 9, 12]:
    knn = SimpleKNNClassifier(X_train, y_train, k=k)

    # Predict on training data
    y_pred = knn.predict(X_train)
    accuracy = np.mean(y_train == y_pred)
    print(f'The accuracy on the training data for k={k} is: {
    # Now let's test our method on the new flowers
    X_new = X[100:,:]
    y_new = y[100:]
    y_pred = knn.predict(X_new)
    accuracy = np.mean(y_new == y_pred)
    print(f'The accuracy on the new data for k={k} is: {accur
```

```
The accuracy on the training data for k=1 is: 100.0%
The accuracy on the new data for k=1 is: 72.0%

The accuracy on the training data for k=3 is: 83.0%
The accuracy on the new data for k=3 is: 74.0%

The accuracy on the training data for k=5 is: 83.0%
The accuracy on the new data for k=5 is: 78.0%

The accuracy on the training data for k=7 is: 82.0%
The accuracy on the new data for k=7 is: 80.0%

The accuracy on the training data for k=9 is: 82.0%
The accuracy on the new data for k=9 is: 82.0%

The accuracy on the training data for k=12 is: 83.0%
The accuracy on the new data for k=12 is: 82.0%
```

- What do you think is the best value of k now?
- The above generalization accuracy estimation algorithm is known as using a train/test split

How Should Method Performance Be Estimated? It Should Be Evaluated on **Unseen Test Data**

- If we train and evaluate on the *same* data, **the model may not generalize well**.
- Analogy to class
  - **Training data** is like homeworks, sample problems, and sample exams.
  - **Testing data** is like the real exam.

We Actually Want the Performance on **New Unseen Data**

Medical Domain

Data we have

What we want

Photos Domain

Data we have

What we want

Business Domain

Data we have

What we want

Estimating **Generalization** on Unseen Data is Important for Model Evaluation and Model Selection

1. **Model evaluation**
   - Is the model accurate enough to deploy?
   - Example: The business department may decide that the ML predictions will be worthwhile if the accuracy in the real world is above 90% on average.
2. **Model selection**
   - Which of many possible models should be used?
   - Example: Which value of k is best for KNN?

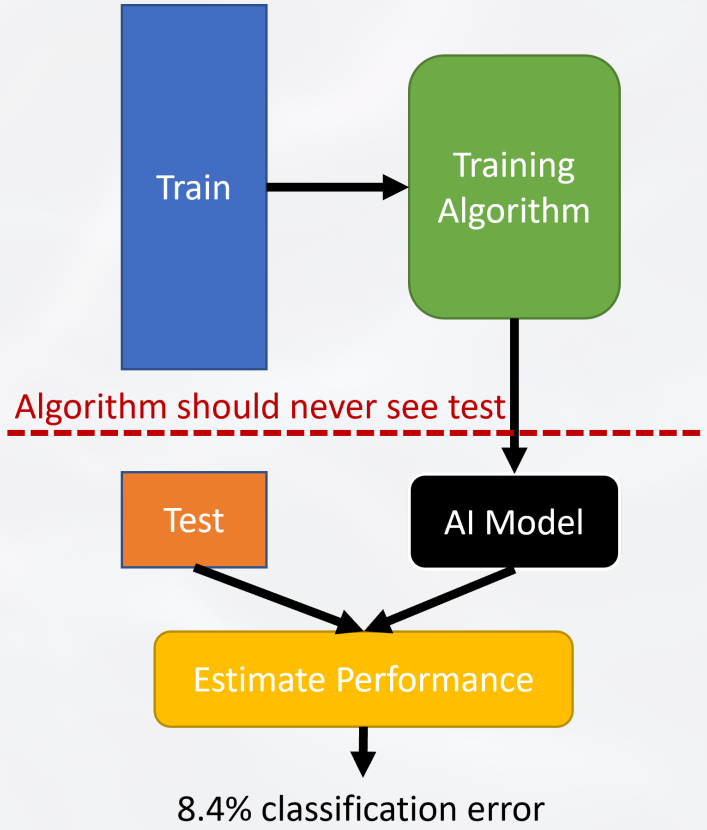
Generalization Error Measures How Much Error the Model Makes on Unseen Data

- How do we measure generalization error since (by definition) we don’t have new unseen data?

Act like we do!

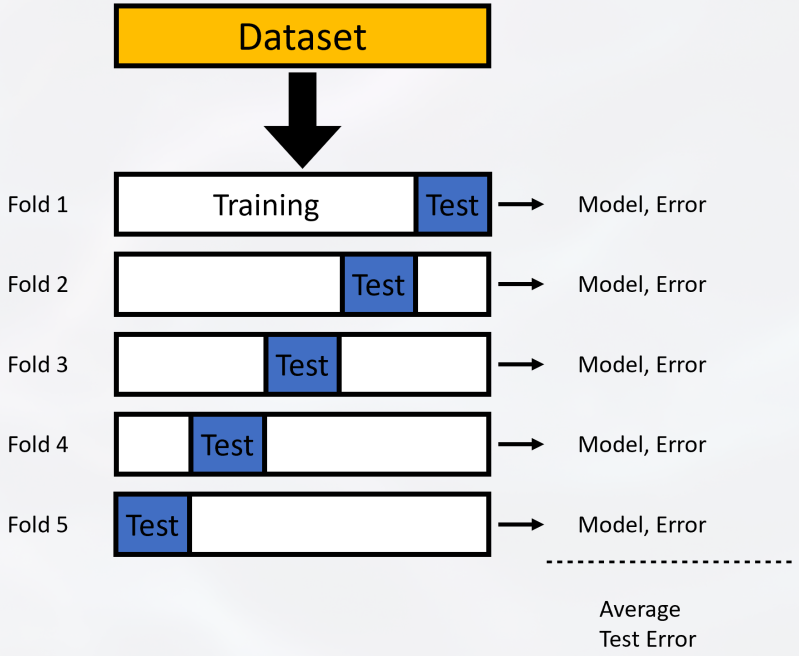
Generalization Error Can Be Estimated by Splitting the Known Dataset

- Split the dataset
  - The *training dataset* is used to estimate the model.
  - The *test dataset* (or *held-out dataset*) is used to estimate generalization error.



Cross-Validation (CV) Generalizes the Simple Train/Test Split to M Disjoint Splits

- Repeat the split process M times
- Fit new model on train
- Evaluate model on test
- Note: M models are fitted throughout process
- Final error estimate is average over all folds



$M = 3, M = 5, M = 10$ are common

KNN Demo: Cross validation is a better estimate of generalization accuracy

```
def cv_estimate(X, k, n_splits=3):
    # Setup split indices and loop over splits
    split_ind = np.floor(np.linspace(0, X.shape[0], num=n_spl
    accuracy_list = []
    for split_start, split_end in zip(split_ind[:-1], split_i
        # Setup boolean array to select test set
        test = np.zeros(X.shape[0], dtype=bool)  # Initialize
        test[int(split_start):int(split_end)] = True  # Set t

        # Create train and test sets (~ is used to flip boole
        X_train, y_train = X[~test, :], y[~test]
        X_test, y_test = X[test, :], y[test]

        # Train model for this split using X_train and y_trai
        knn = SimpleKNNClassifier(X_train, y_train, k=k)

        # Compute accuracy on test split
        y_pred = knn.predict(X_test)
        accuracy = np.mean(y_test == y_pred)
        accuracy_list.append(accuracy)
    return np.mean(accuracy_list) # Take mean of accuracy
```

KNN Demo: Cross validation is a better estimate of generalization accuracy

```python
for k in [1, 3, 5, 7, 9, 12]:
    cv_acc = cv_estimate(X, k, n_splits=3)
    print(f'CV accuracy estimate for k={k} is {100*cv_acc:.1f
```

```
CV accuracy estimate for k=1 is 74.7%
CV accuracy estimate for k=3 is 76.0%
CV accuracy estimate for k=5 is 76.7%
CV accuracy estimate for k=7 is 78.0%
CV accuracy estimate for k=9 is 77.3%
CV accuracy estimate for k=12 is 76.0%
```

- What do you think is the best value of k using CV and why? Is it the same as a single train/test split?

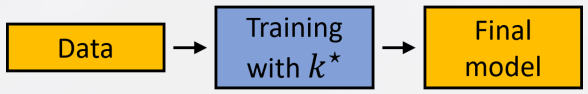
CV Generalization Error Can Aid in **Model Selection**

1. Run CV (to estimate generalization) for multiple k.

2. Choose k^* whose CV performance is the best.

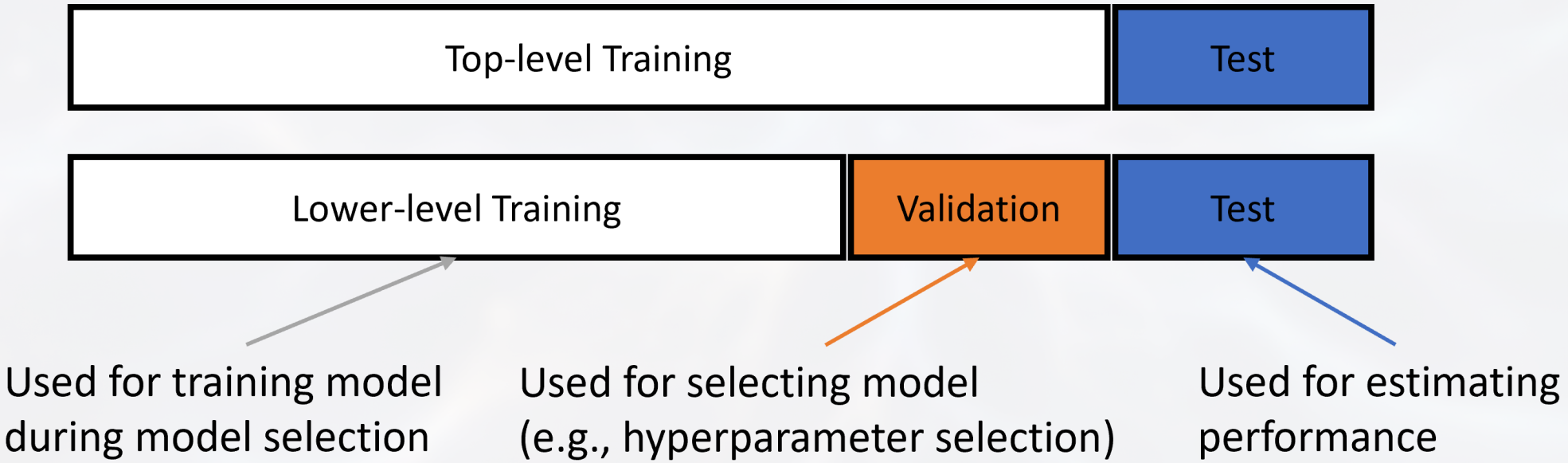
$$k^* = \arg\min_k \text{CVGenError}(k; X)$$

3. For final model, train model with all data using k^*



But What if We Want to Select a Model AND Estimate the Model's Performance?

- **Inception!**
- **Nested train/test split (most common)**



- **Nested CV (better but expensive)**

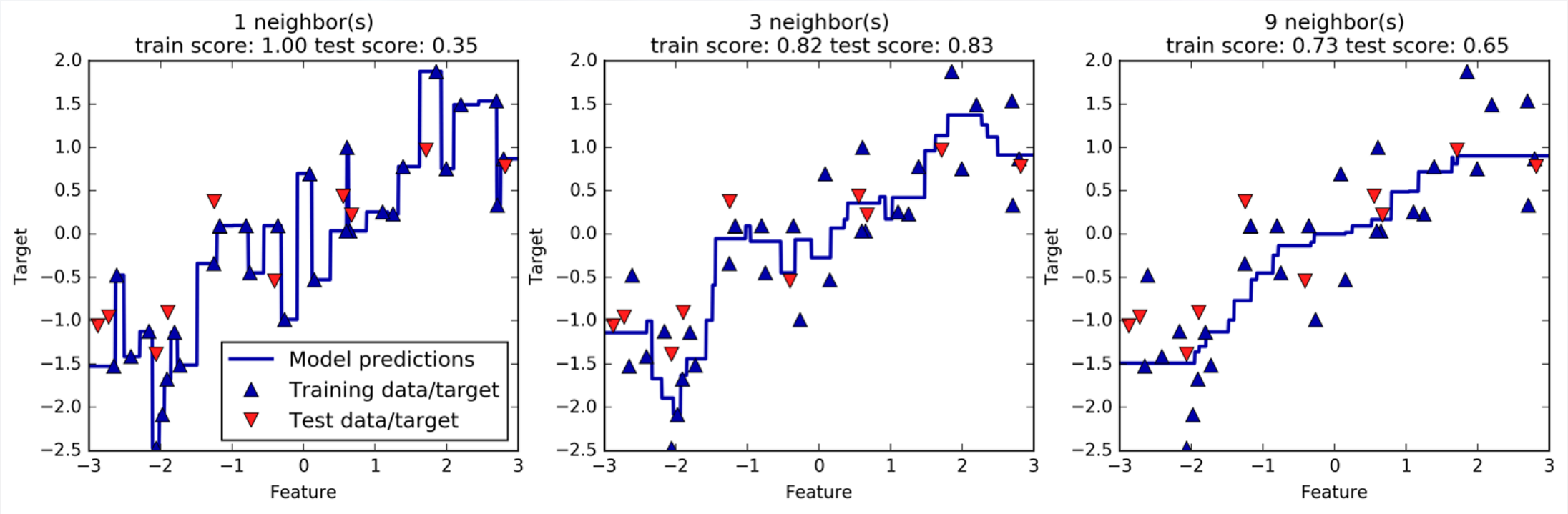
Real-World Caveat: Even CV Performance Estimates Are Only Good if **Real-World Distribution** is Like the Training Data

- Training images in the daytime, but real-world images may be at night.
  - (*Domain generalization* tackles this problem)
- Training based on historical court cases that are biased against minorities, but real-world court cases should be unbiased.
  - (Fairness in AI/ML is a recent popular topic)
- Training based on historical stock market data, but real-world stock market has changed.

Okay, Back to KNN...

KNN Regression Can Be Used to Predict Continuous Values

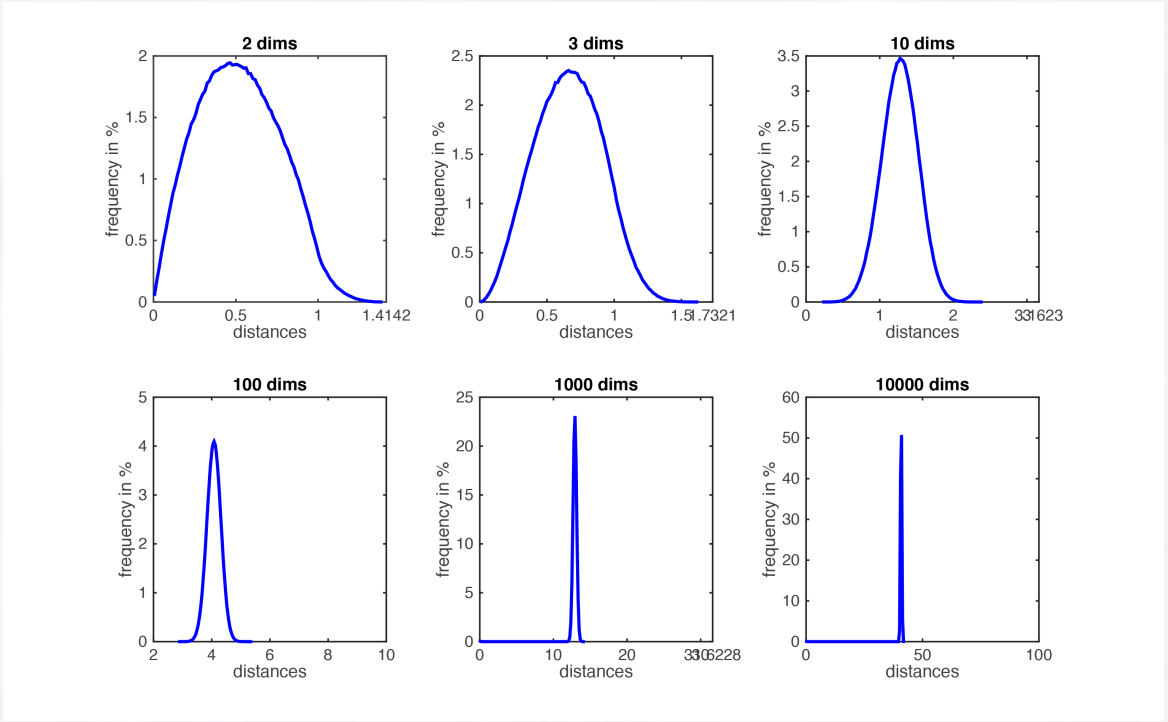
1. Find k nearest neighbors
2. Predict average of k nearest neighbors (possibly weighted by distance)



https://medium.com/analytics-vidhya/k-neighbors-regression-analysis-in-python-61532d56d8e4

The Performance and Intuitions of KNN Degrade Significantly in High Dimensions (*The Curse of Dimensionality*)

- The distances between *any two points* in high dimensions is nearly the same.



Distance between two **random points** concentrate around a single value.

https://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/lecturenote02_kNN.html

Illustration of KNN Problem: The “Ideal” 1D Case Where Neighbors are Informative

▶ Code

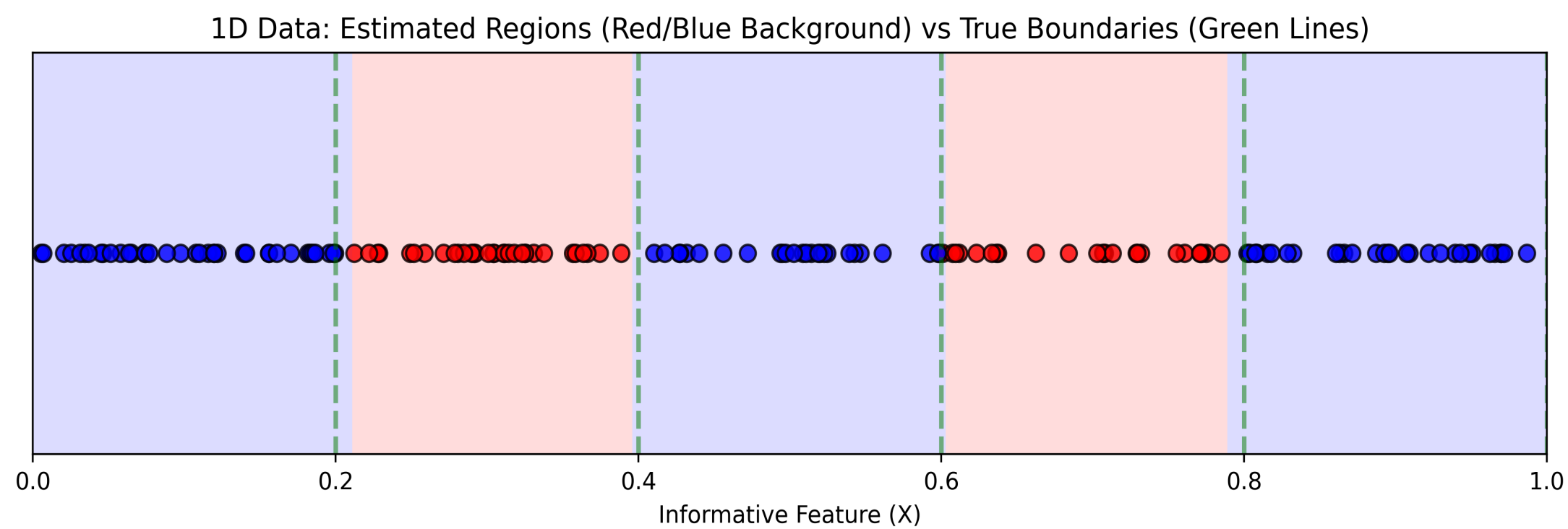


Illustration of KNN Problem: Noisy Dimension Significantly Reduces KNN Performance

▶ Code

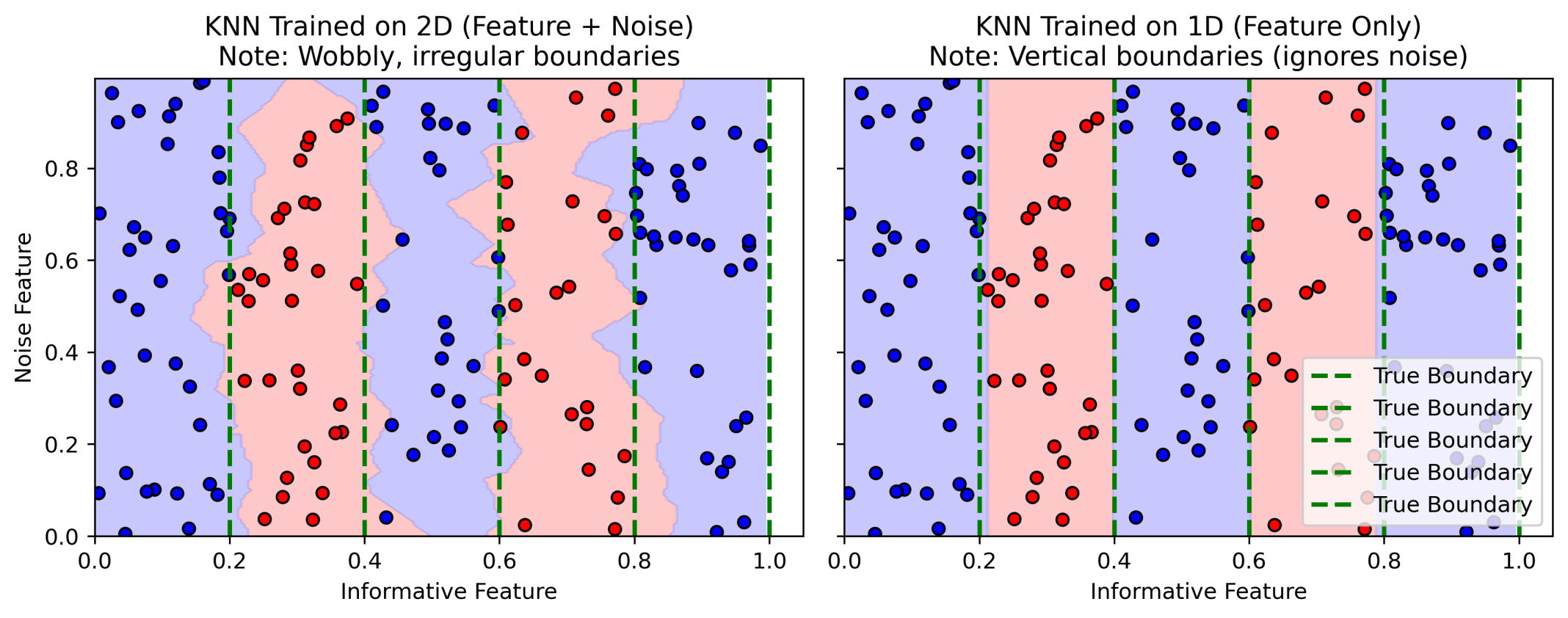
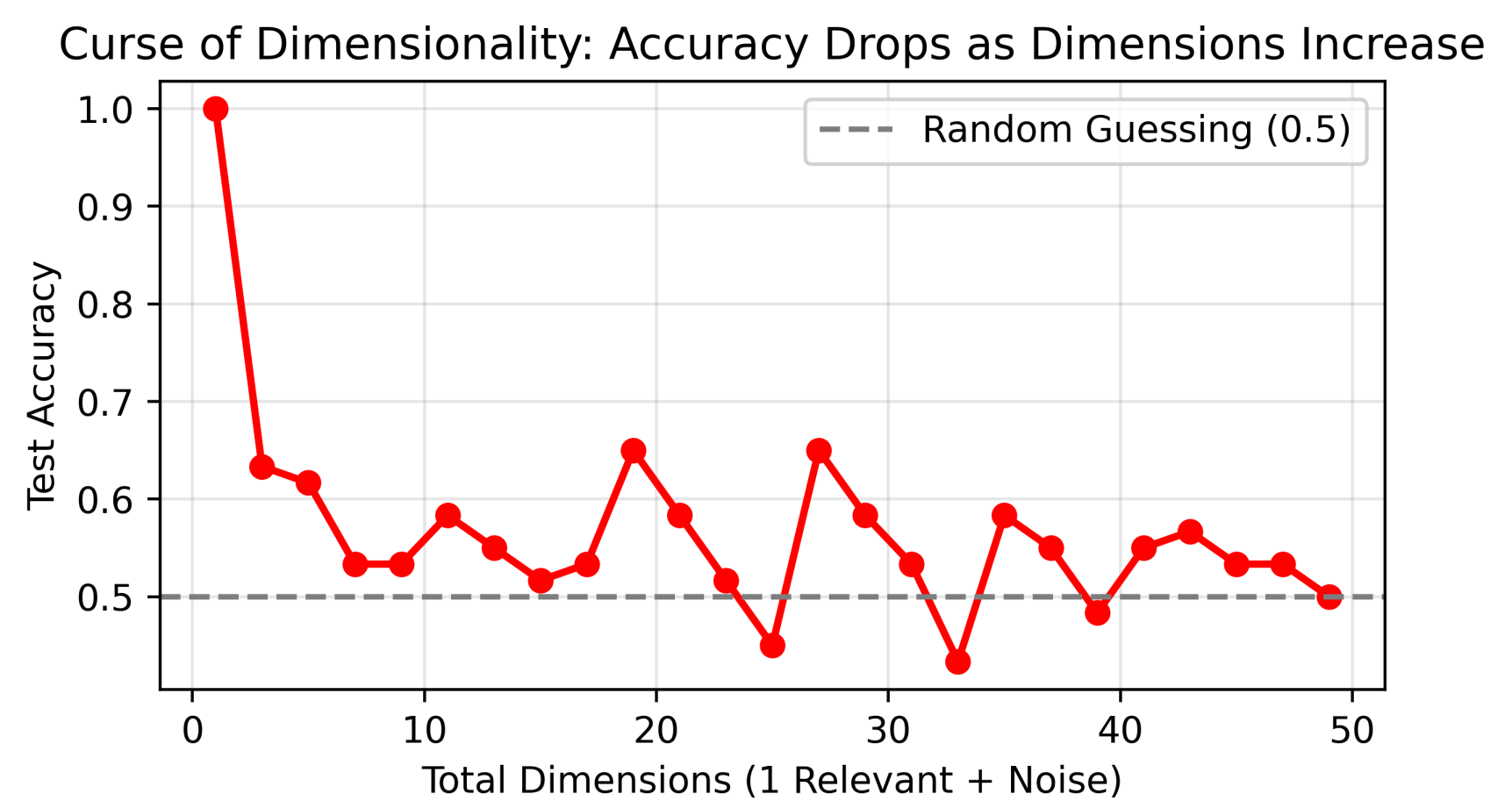


Illustration of KNN Problem: Accuracy Degrades to Random Chance in Higher Dimensions

▶ Code



Solution 1: Reduce the Dimensionality and Then Use KNN

MNIST Digits

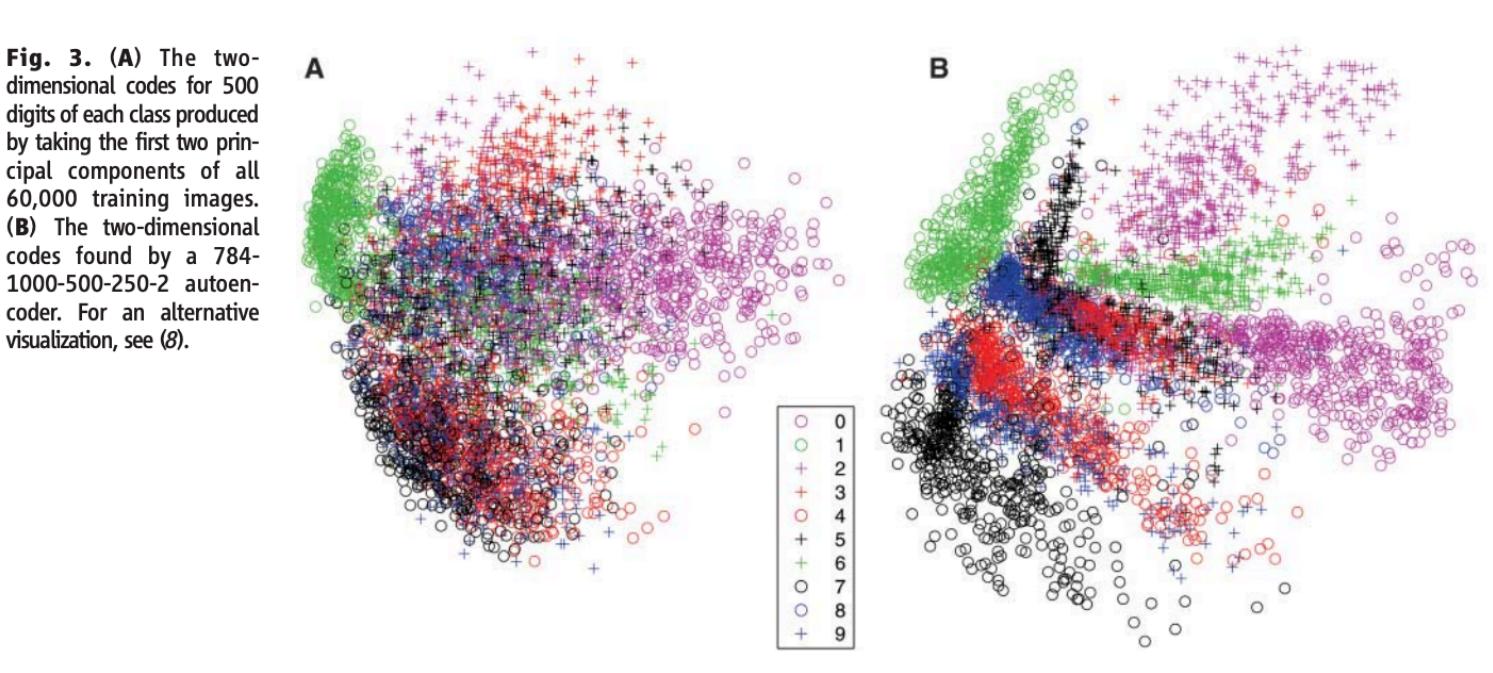
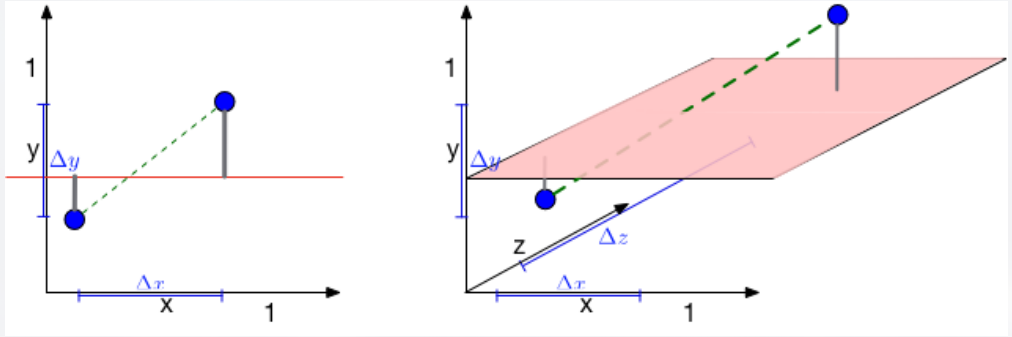
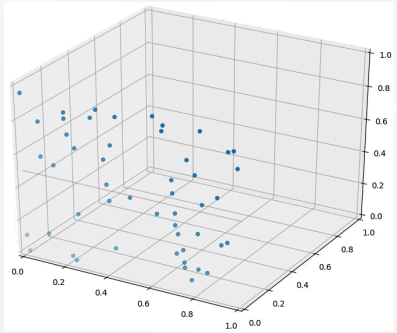


Fig. 3. (**A**) The two-dimensional codes for 500 digits of each class produced by taking the first two principal components of all 60,000 training images. (**B**) The two-dimensional codes found by a 784-1000-500-250-2 autoencoder. For an alternative visualization, see (*8*).

Reducing the Dimensionality of Data with Neural Networks, G. E. Hinton and R. R. Salakhutdinov, Science, 2006, https://www.cs.toronto.edu/~hinton/science.pdf

Solution 2 (Non-KNN): Compute Distance to Hyperplane Instead





Distance to hyperplane is **constant** but pairwise distances between points grows as dimensionality increases.

How do we compute distance to hyperplane?

- Dot product with unit normal vector plus constant!

$$x^T n + c$$

- *One view of linear classifiers*: 1D projection and then classification

Excellent illustrations from: https://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/lecturenote02_kNN.html

Related Reading and Source for KNN Curse of Dimensionality Illustrations

- https://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/lecturenote02_kNN.html