CS6375: Machine Learning

Gautam Kunapuli

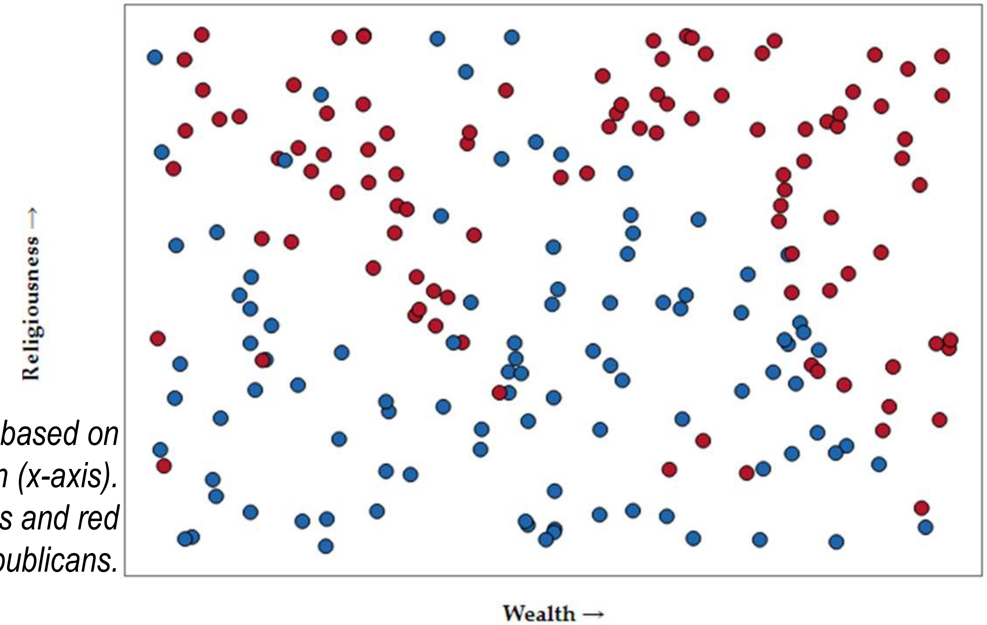
Nearest Neighbor Methods

THE UNIVERSITY OF TEXAS AT DALLAS
Erik Jonsson School of Engineering and Computer Science

Example: Predicting Political Affiliation

Example: Develop a model to **predict the political affiliation** of individuals using **demographic indicators**.

Here, the features are **wealth** (x_1) and **self-identified religiousness** (x_2)



Hypothetical party registration based on religiousness (y-axis) and wealth (x-axis). Blue represents Democrats and red represents Republicans.

This example and figures are borrowed from Scott Fortmann-Roe's blog article "[Understanding the Bias-Variance Tradeoff]", July 2012.

Solution Approach 1 (lazy solution):

*"**lazy**" means **learning does not occur till a test example** is presented; this is in contrast to "**eager**" algorithms that learn **without seeing any test examples** and **discard** training examples **after learning***

Initialize: Store **all** training examples $(\boldsymbol{x}_i, y_i)_{i=1}^{n}$

Classifying a new test point $\boldsymbol{x}_{\text{test}}$

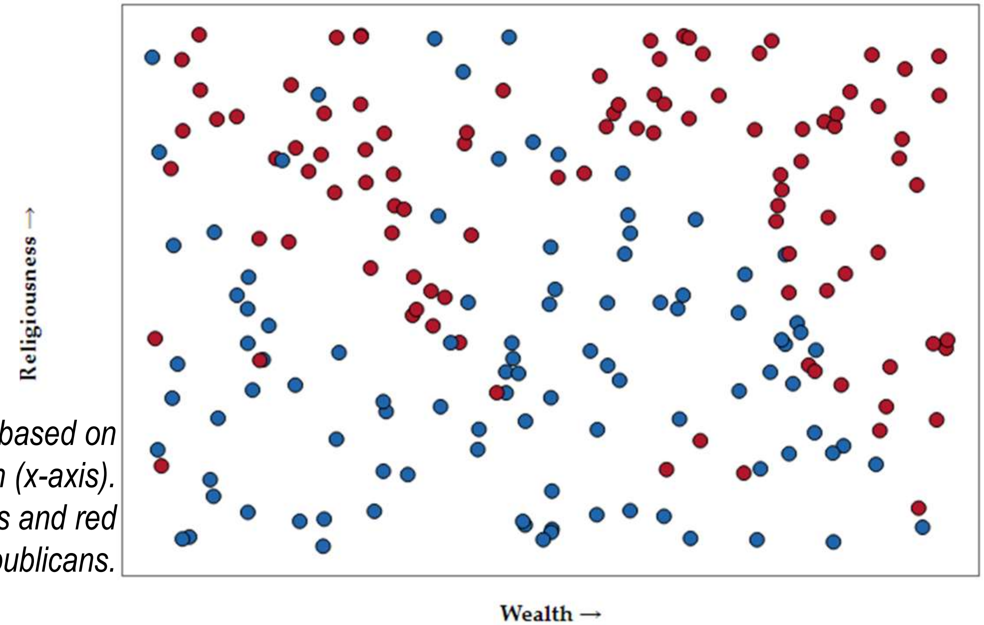
Find the training example $(\boldsymbol{x}_i, y_i)$ such that $\boldsymbol{x}_i$ is **closest** to $\boldsymbol{x}_{\text{test}}$

Classify $\boldsymbol{x}_{\text{test}}$ with the label y_i

1-Nearest Neighbor

Example: Develop a model to **predict the political affiliation** of individuals using **demographic indicators**.

Here, the features are **wealth** (x_1) and **self-identified religiousness** (x_2)



Hypothetical party registration based on religiousness (y-axis) and wealth (x-axis). Blue represents Democrats and red represents Republicans.

Solution Approach 1 (lazy solution, 1-nearest neighbor):

*"lazy" means **learning does not occur till a test example** is presented; this is in contrast to "eager" algorithms that learn **without seeing any test examples** and **discard** training examples **after learning***

Initialize: Store **all** training examples $(\boldsymbol{x}_i, y_i)_{i=1}^{n}$

Classifying a new test point $\boldsymbol{x}_{\text{test}}$

Find the training example $(\boldsymbol{x}_i, y_i)$ such that $\boldsymbol{x}_i$ is **closest** to $\boldsymbol{x}_{\text{test}}$

Classify $\boldsymbol{x}_{\text{test}}$ with the label y_i

How do we measure **closeness**? Since we view data as d-dimensional points/vectors, measure the **distance between them** in d-dimensional space.

Euclidean distance is a natural way to measure closeness between two points; it is denoted $D(\boldsymbol{x}, \boldsymbol{z}) = ||\boldsymbol{x} - \boldsymbol{z}||$, and computed as

$$D(\boldsymbol{x}, \boldsymbol{z}) = \sqrt{(x_1 - z_1)^2 + \cdots + (x_d - z_d)^2}$$

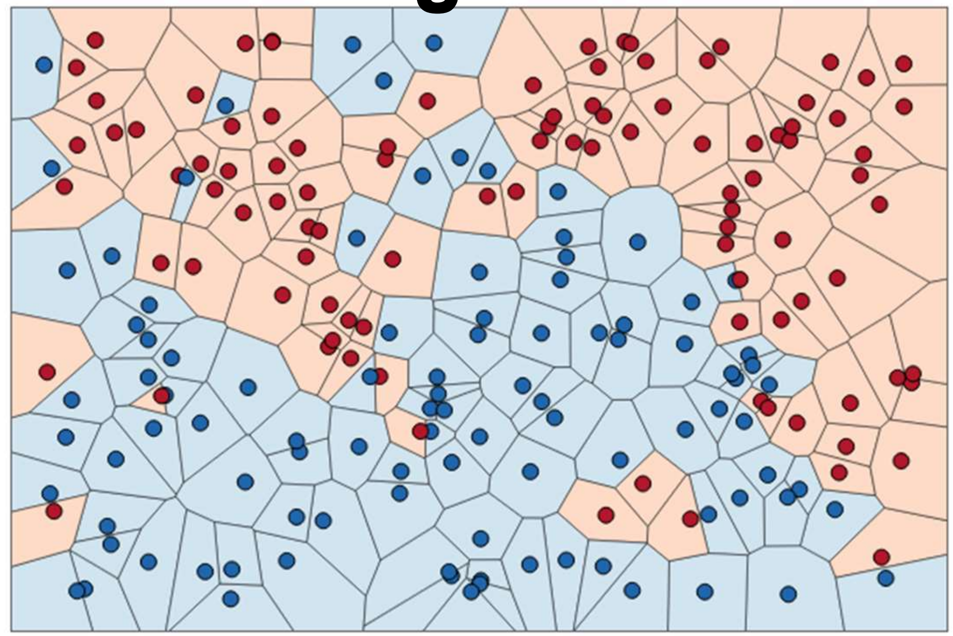
For **efficiency**, we often simply use **squared distance** to avoid the square root computation as it does not change the result (why?)

$$D(\boldsymbol{x}, \boldsymbol{z})^2 = (x_1 - z_1)^2 + \cdots + (x_d - z_d)^2$$

1-Nearest Neighbor and Voronoi Diagrams

Example: Develop a model to **predict the political affiliation** of individuals using **demographic indicators**.

Here, the features are **wealth** (x_1) and **self-identified religiousness** (x_2)



Hypothetical party registration based on religiousness (y-axis) and wealth (x-axis). Blue represents Democrats and red represents Republicans.

Solution Approach 1 (lazy solution, 1-nearest neighbor):

*"**lazy**" means **learning does not occur till a test example** is presented; this is in contrast to "**eager**" algorithms that learn **without seeing any test examples** and **discard** training examples **after learning***

Initialize: Store <u>**all**</u> training examples $(\boldsymbol{x}_i, y_i)_{i=1}^{n}$

Classifying a new test point $\boldsymbol{x}_{\text{test}}$

Find the training example $(\boldsymbol{x}_i, y_i)$ such that $\boldsymbol{x}_i$ is **closest** to $\boldsymbol{x}_{\text{test}}$

Classify $\boldsymbol{x}_{\text{test}}$ with the label y_i

- We can visualize the classifier using the **Voronoi diagram**, Given a set of points, a **Voronoi diagram** of the points describes **areas that are closest to each point** in the set
- These areas can be viewed as **zones of control**
- **Different measures** of closeness produce **different diagrams**
    - Euclidean distance is the most popular, others are possible e.g., `scikit`'s `DistanceMetric` class supports Manhattan, Chebyshev, Minkowski, Mahalanobis, Weighted and many others

1-Nearest Neighbor: Properties

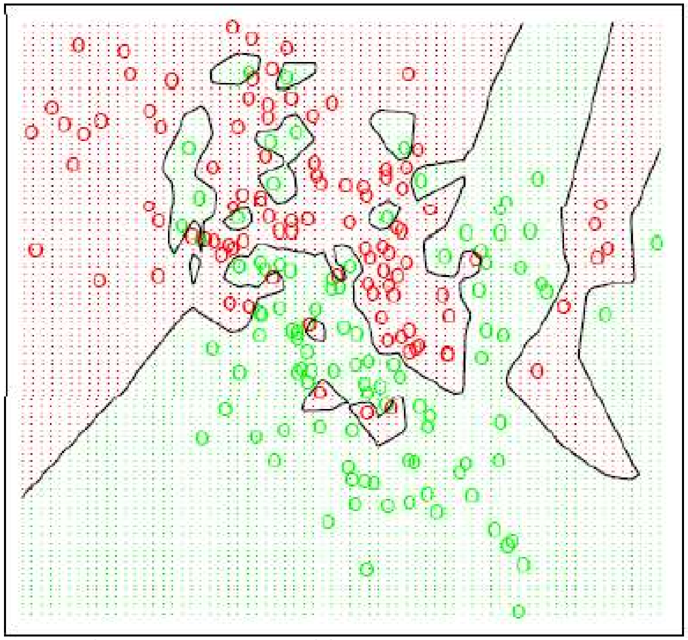
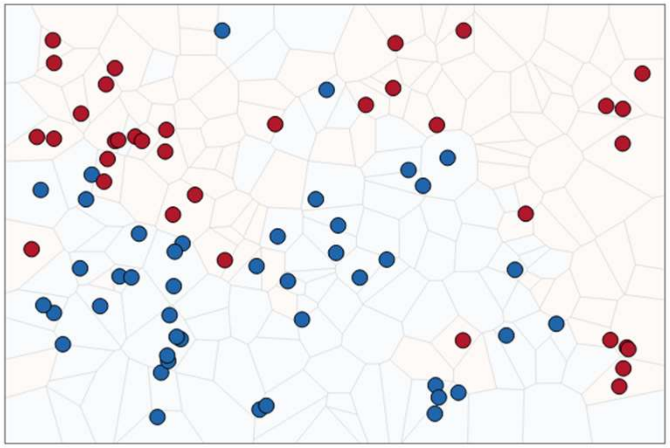


Figure from The Elements of Statistical Learning *by Hastie, Tibshirani and Friedman.*

- **Decision boundaries** are formed by the training examples
- Each line segment making up the decision boundary is **equidistant between two points** of the opposite class
- Noise and large number of examples can easily lead to **overfitting** (as we could start having islands of neighborhoods)

Prediction is equivalent to identifying which region the new test point will be in (i.e., a red or a blue region).



1-NN depends **critically** on the choice of **distance metric**.

All features must have the **same range of values**. Otherwise, features will larger range become more important. To avoid this, we can **normalize** the feature values.

$$\mu_j = \frac{1}{n}\sum_{i=1}^{n} x_i^j \quad \textbf{(feature mean)}$$

$$\sigma_j^2 = \frac{1}{n}\sum_{i=1}^{n}\left(x_i^j - \mu_j\right)^2 \quad \textbf{(feature variance)}$$

$$\bar{x}_i^j = \frac{x_i^j - \mu_j}{\sigma_j} \quad \textbf{(normalized features)}$$

1-Nearest Neighbor: Properties

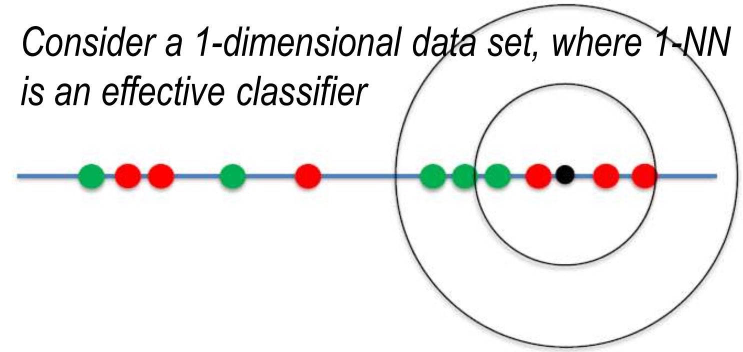
1-NN is highly **sensitive to irrelevant inputs**
Irrelevant or noisy features will add random perturbations to the distance measure and can easily hurt performance

Several approaches are possible to handle this issue:
- **Feature Weighting**: Weight each feature based on its **mutual information** to the target class (label). Then use the weighted squared-distance as the distance metric

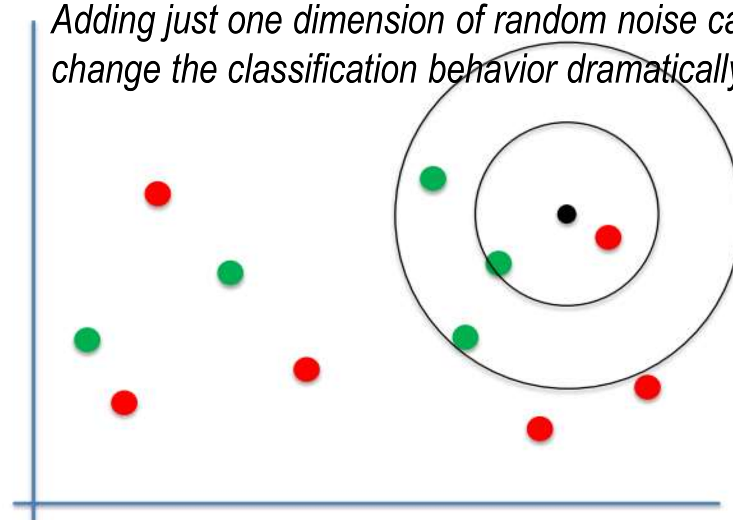
$$D(\boldsymbol{x}, \boldsymbol{z}) = \sqrt{w_1(x_1 - z_1)^2 + \cdots + w_d(x_d - z_d)^2}$$

- **Metric Learning**: Learn a metric and use the Mahalanobis distance
- **Smoothing**: Find the **k nearest neighbors** and have them **vote**; considering multiple neighbors can reduce the effects of noise

Consider a 1-dimensional data set, where 1-NN is an effective classifier



Adding just one dimension of random noise can change the classification behavior dramatically



Figures by Nicholas Ruozzi

Computational Complexity: For classification, we have to compute distances between **all points** in the training set and the **new test point**. With n data points in d-dimensional space, this takes $O(nd)$ time for Euclidean distance

k-Nearest Neighbor

k-NN looks at the **k closest points** in the training set and uses majority voting to determine the label *(choose k to be odd)*

Solution Approach 2 (k-nearest neighbors):

Initialize: Store __all__ training examples $(\boldsymbol{x}_i, y_i)_{i=1}^{n}$
Classifying a new test point $\boldsymbol{x}_{\text{test}}$
Find ***k*** training examples $(\boldsymbol{x}_i, y_i)$ such that $\boldsymbol{x}_i$ are **closest** to $\boldsymbol{x}_{\text{test}}$
Classify $\boldsymbol{x}_{\text{test}}$with the majority label from the nearest neighbor set $\{y_i\}, i \in nearestneigbors(\boldsymbol{x}_{\text{test}})$

k-NN smooths out noise by considering many neighboring training data points to make a decision. **How many neighbors** should we consider?

- Too small and estimates are noisy, too large and accuracy suffers

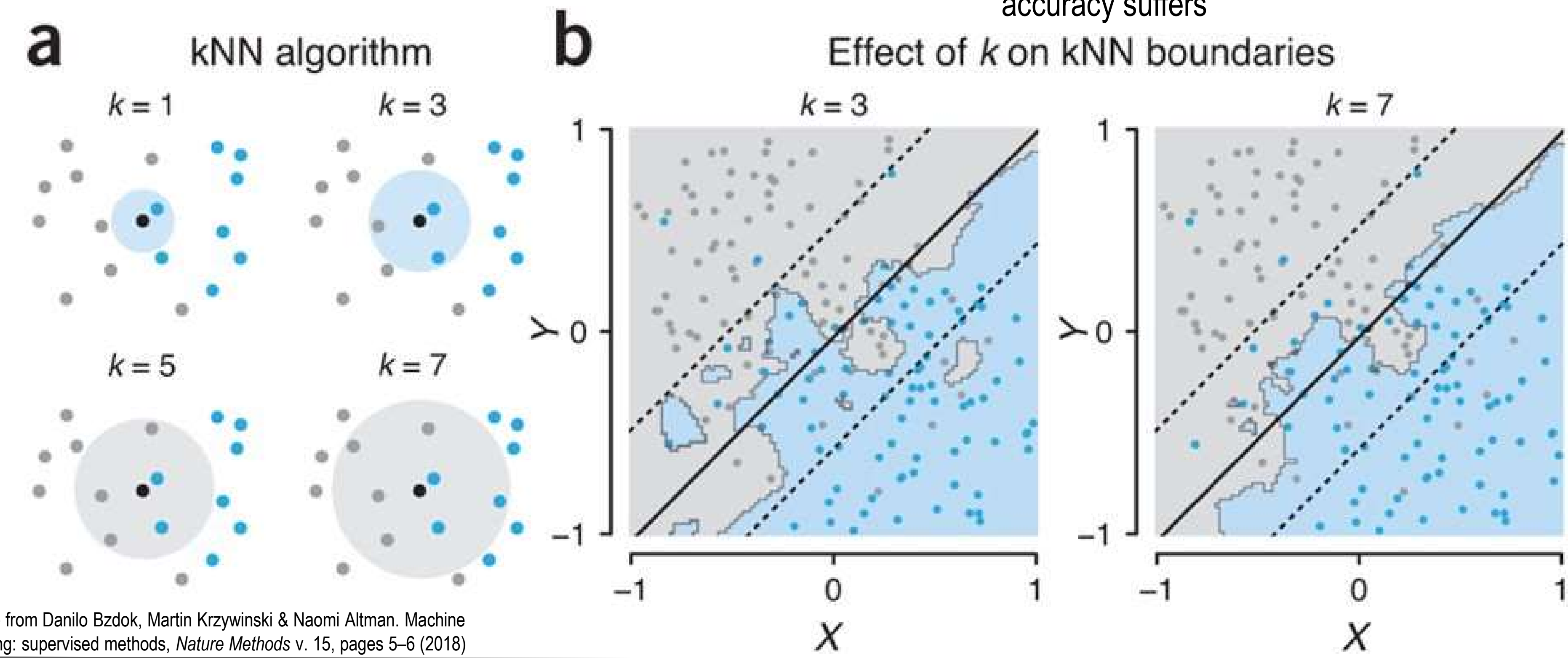
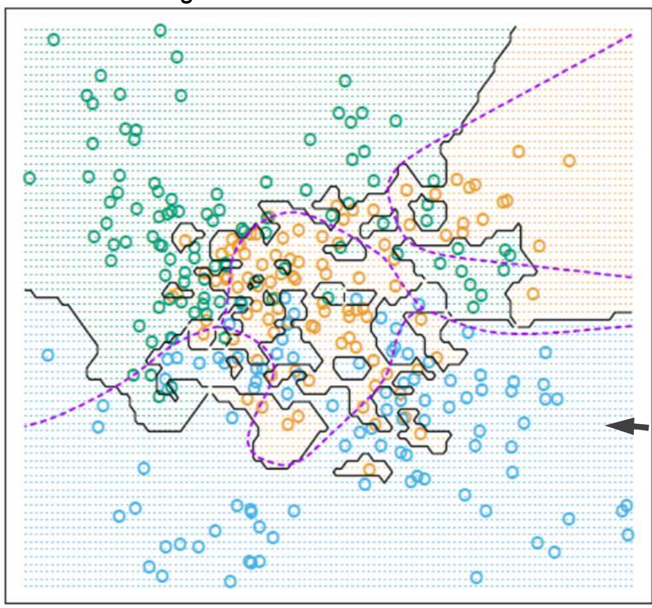


Figure from Danilo Bzdok, Martin Krzywinski & Naomi Altman. Machine learning: supervised methods, *Nature Methods* v. 15, pages 5–6 (2018)

k-Nearest Neighbor: Selecting k

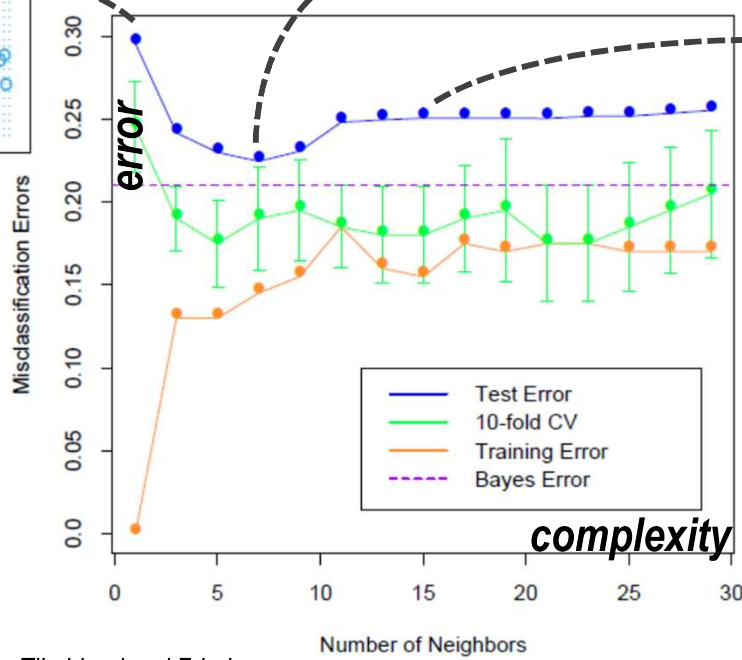
k-NN can easily handle **multi-class data** as it simply looks at the neighbors to make a decision. This makes it well suited for classification problems for
- handwriting recognition
- satellite scene image analysis
- EKG pattern classification

1 nearest neighbor

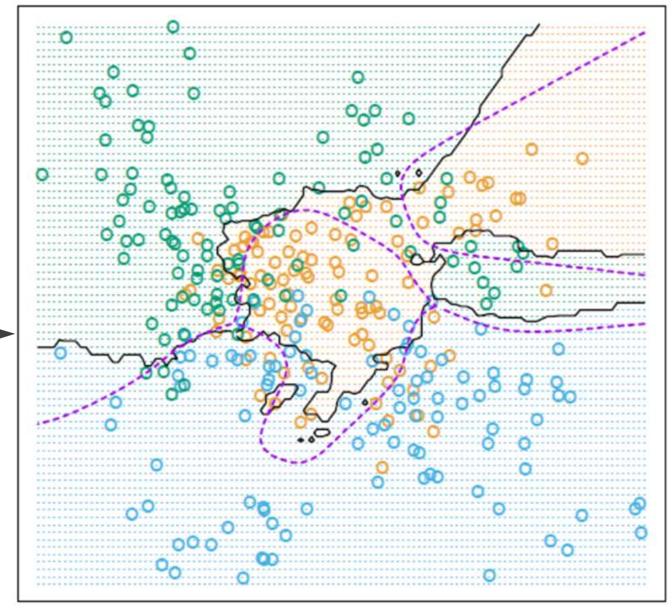


1-NN only uses a single closest point, so its bias is low and variance is high

7 nearest neighbors



15 nearest neighbors



15-NN only uses many closest points, so its bias is higher

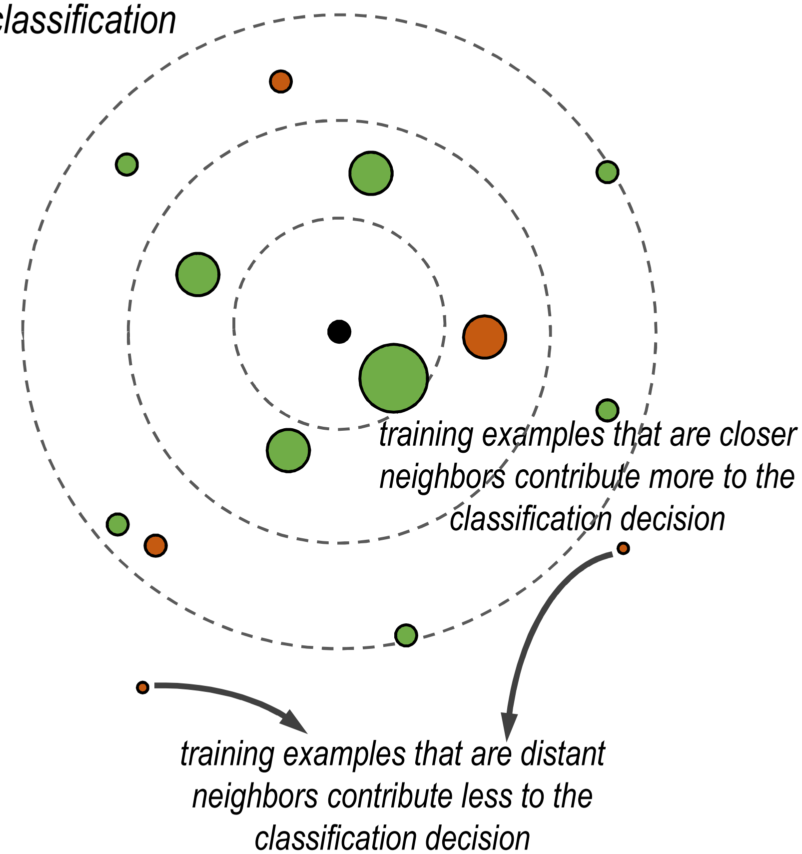
Figures from The Elements of Statistical Learning *by Hastie, Tibshirani and Friedman.*

Practical Issues

What happens when some of the nearest neighbors are very far away? Use **Distance-Weighted Nearest Neighbors** to weight **training examples**.

Weight **contribution of a neighbor** to classification decision according to its **distance from the new test example**

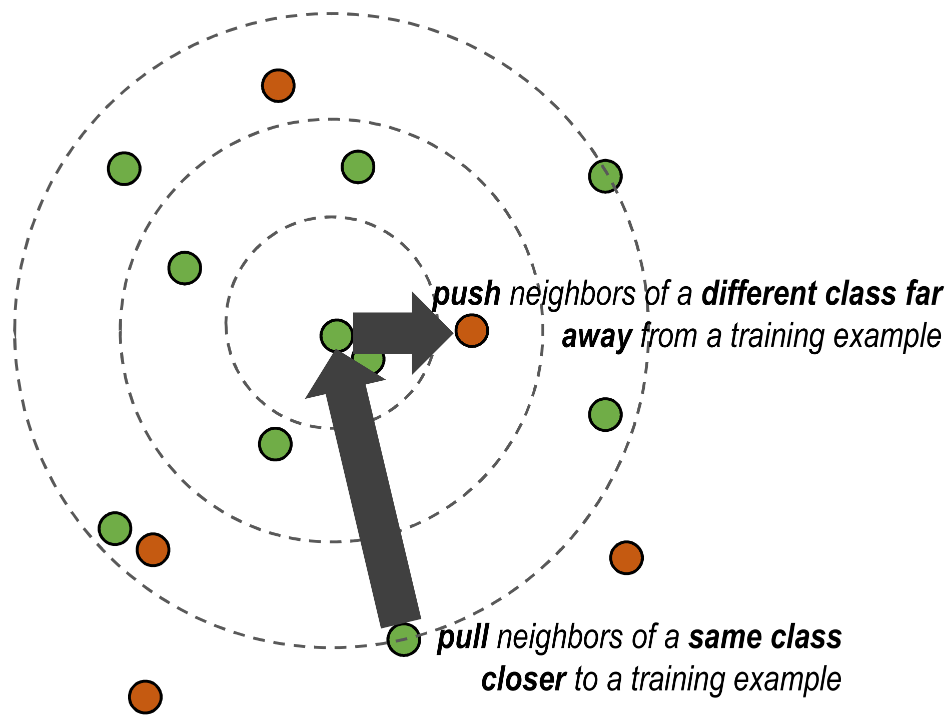
- Weight varies inversely with distance such that closer points get higher weight: $w_i = \frac{1}{D(\boldsymbol{x}_i, \boldsymbol{x}_{\text{test}})}$
- *In the extreme case, we can use the entire data set after weighting for classification*



What happens when some irrelevant attributes mislead kNN? Use **Metric Learning** to discover domain-specific means to measure distances and ignore bad features

- ***Feature weighting*** *weights each feature based on its ability to reduce classification error using criteria such as mutual information*
- Metric learning **learns** a **distance metric** from scratch using the training data (that is, learn M in the Mahalanobis distance metric)

$$D(\boldsymbol{x}, \boldsymbol{z})^2 = (\boldsymbol{x} - \boldsymbol{z})^T M (\boldsymbol{x} - \boldsymbol{z})$$

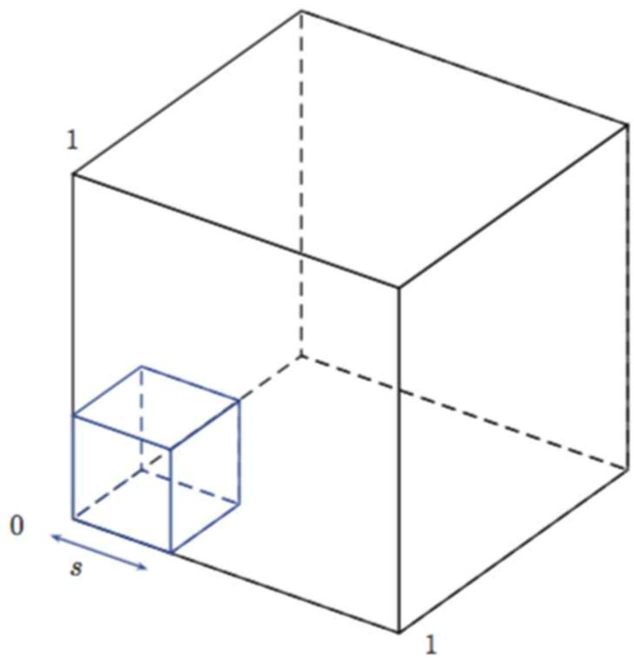


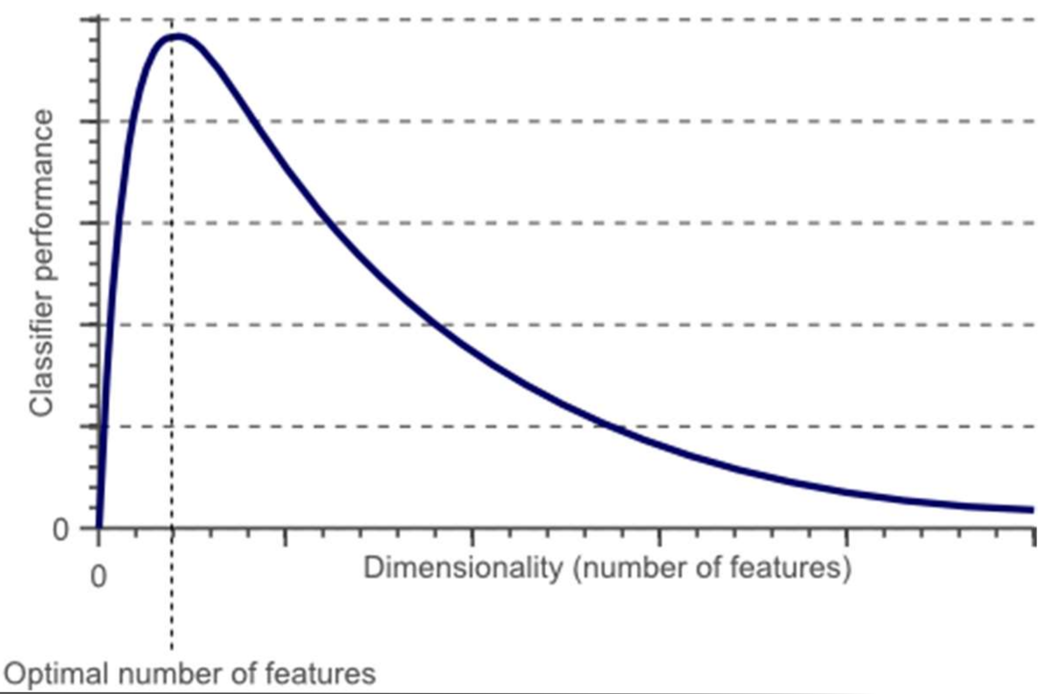
Metric learning is an active research area

Curse of Dimensionality

Nearest neighbor **breaks down in high-dimensional spaces** because the “neighborhood” becomes very large. Suppose we have 5000 points **uniformly distributed** in the unit hypercube and we want to apply the 5-nearest neighbor algorithm; suppose our test point is at the origin

- 1D: On a one-dimensional line, we must go a distance of 5/5000 = 0.001 on average to capture the 5 nearest neighbors
- 2D: In two dimensions, we must go $\sqrt{0.001}$ to get a square that contains 0.001 of the volume
- d-dimensions: In d dimensions, we must go $\sqrt[d]{0.001}$



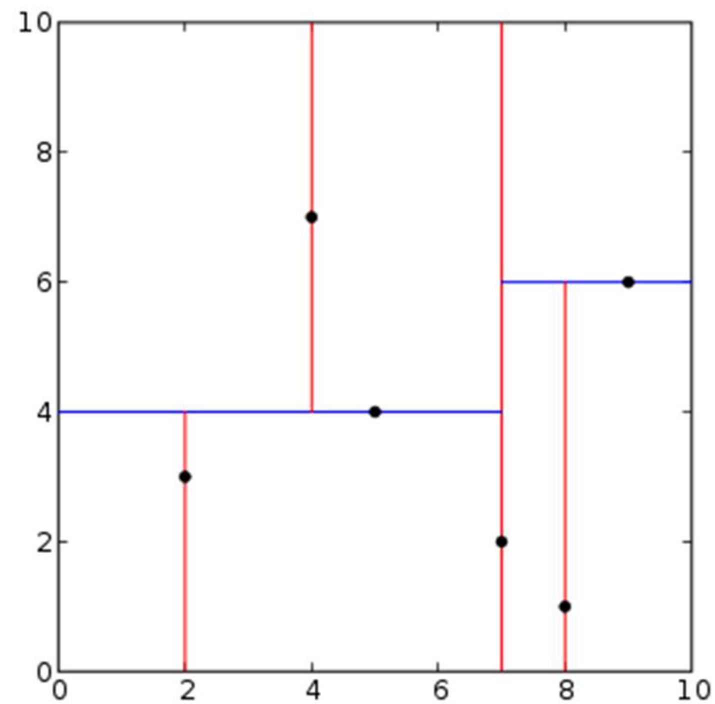


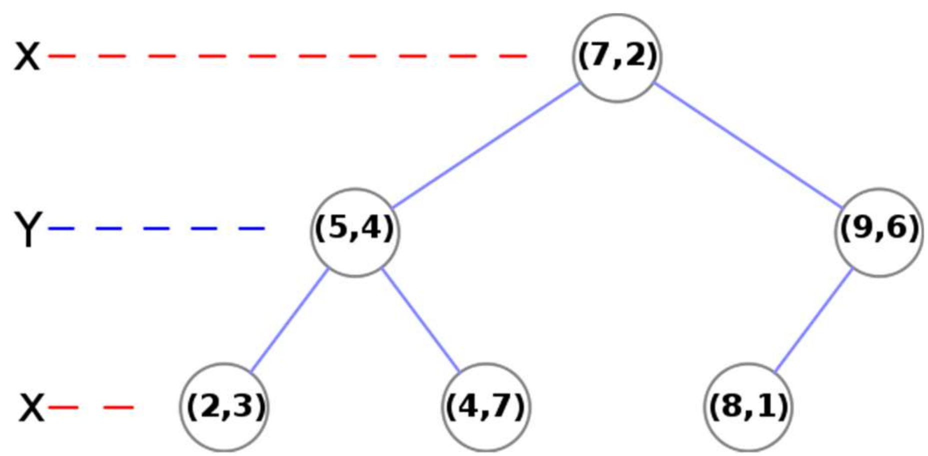
k-Dimensional Trees

k-d tree (short for k-dimensional tree) is a **space-partitioning data structure** for **organizing points** in a k-dimensional space
- binary tree in which every node is a k-dimensional point
- every non-leaf node is an axis-aligned hyperplane that splits the space into two parts

- *Starting with the entire training set, choose some dimension, i*
  - *Select an element of the training data whose i-th dimension has the median value among all elements of the training set*
  - *Divide the training set into two pieces: depending on whether their i-th attribute is smaller or larger than the median*
  - *Repeat this partitioning process on each of the two new pieces separately*

By design, the constructed k-d tree is "**bushy**"
- The idea is that if new points to classify are **evenly distributed** throughout the space, then the expected cost of classification is approximately $O(d \log n)$ operations



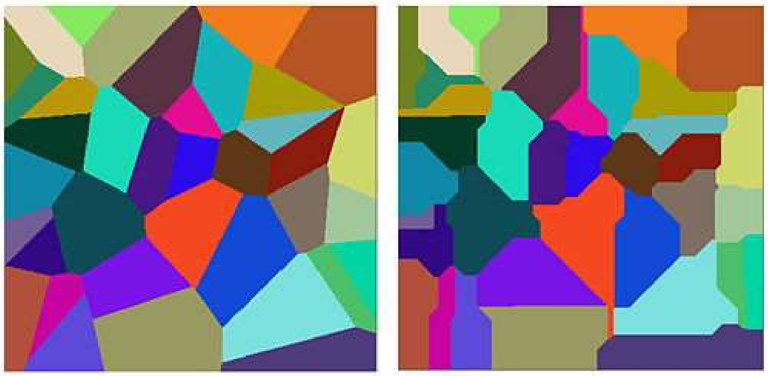


Figures from Wikipedia

Nearest Neighbor Methods

Advantages:

- Model is extremely simple and intuitive
- Very easy to implement, can be very fast in practice
- Flexible decision boundaries
- Variable-sized hypothesis space



Euclidean Manhattan

Disadvantages:

- Distance function must be carefully chosen
- Irrelevant features have a high impact and must be handled/removed
    - Use **feature weighting** or **metric learning**
- Typically cannot handle high-dimensional spaces
- Memory and classification-time costs grow with dimensionality
    - Use specialized data structures such as **k-d trees** to efficiently find nearest neighbors