# K-NEAREST NEIGHBORS

# Nearest Neighbor Learning (Instance Based Learning)

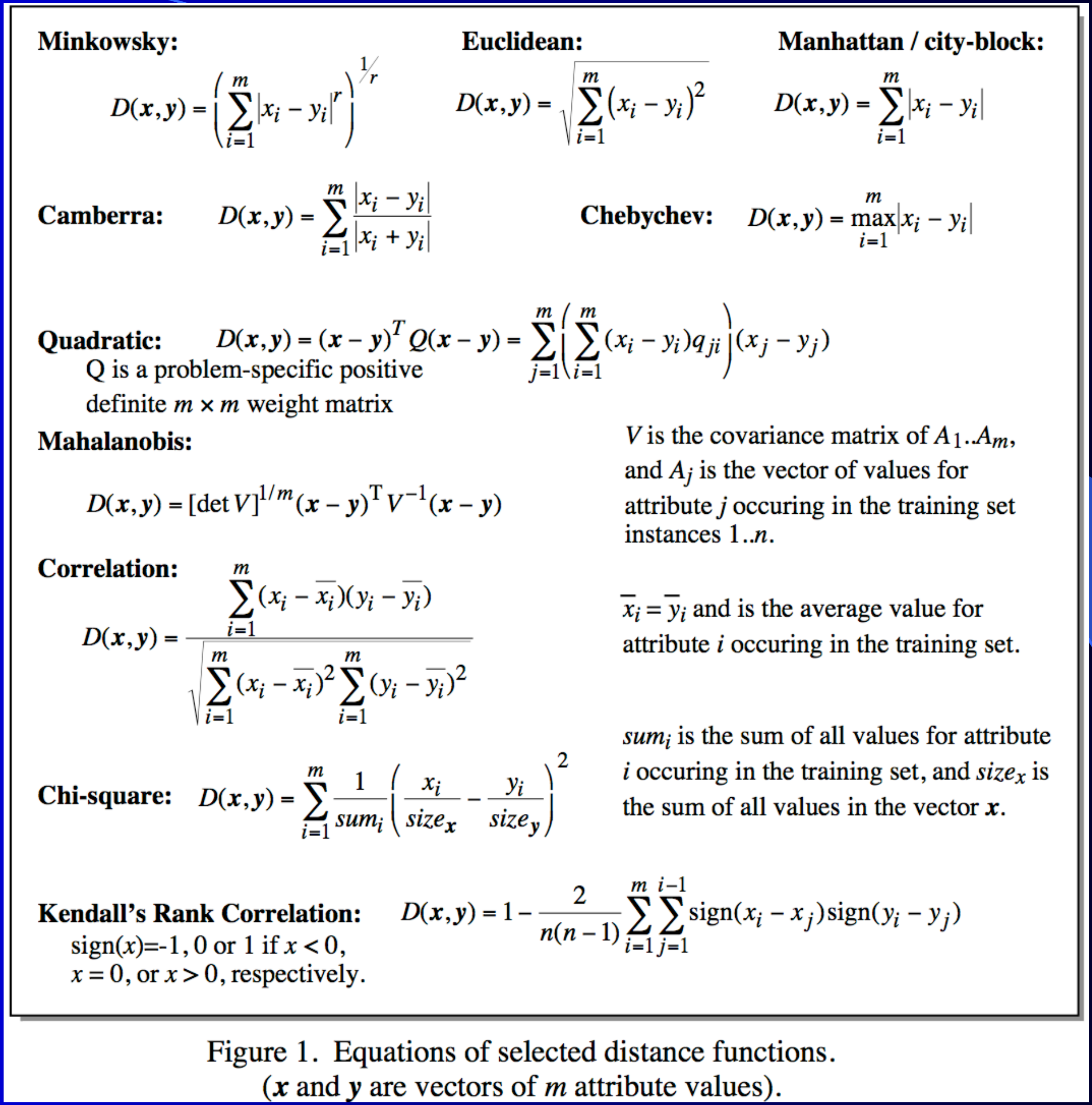
- Classify based on local similarity
- Ranges from simple *nearest neighbor* to case-based and analogical reasoning
- Use local information near the current query instance to decide the classification of that instance
- Can represent quite complex decision surfaces in a simple manner
  - Local model vs a model such as an MLP which uses a global decision surface

# *k*-Nearest Neighbor Approach

- Simply store all (or some representative subset) of the examples in the training set
- When desiring to generalize on a new instance, measure the *distance* from the new instance to all the stored instances and the nearest ones *vote* to decide the class of the new instance
- No need to pre-process a specific hypothesis (Lazy vs Eager learning)
  - Fast learning
  - Can be slow during execution and require significant storage
  - Possible to index the data or reduce the instances stored to enhance efficiency

# *k*-Nearest Neighbors

- Naturally supports real valued features
- Typically use Euclidean distance: $d(\mathbf{x}, \mathbf{y}) = \left(\sum_{i=1}^{n} (x_i - y_i)^2\right)^{1/2}$
- Nominal/unknown features can just be a 1/0 distance
- The *k* nearest neighbors each vote for their output class and we sum the votes
  - Assume three output classes A, B, C and $k = 3$ with summed votes (2, 1, 0)
  - A is the winning output class, or we could output all the votes, or normalize to a probability vector (.67, .33. 0)
- *k* greater than 1 is more noise resistant, but too large of *k* could lead to less accuracy as less relevant neighbors have more influence (common values: *k*=3, *k*=5)
  - Usually discover best *k* by trial and error (trying different values for a task)



**Minkowsky:**

$$D(\boldsymbol{x},\boldsymbol{y}) = \left( \sum_{i=1}^{m} |x_i - y_i|^r \right)^{1/r}$$

**Euclidean:**

$$D(\boldsymbol{x},\boldsymbol{y}) = \sqrt{\sum_{i=1}^{m} (x_i - y_i)^2}$$

**Manhattan / city-block:**

$$D(\boldsymbol{x},\boldsymbol{y}) = \sum_{i=1}^{m} |x_i - y_i|$$

**Camberra:**

$$D(\boldsymbol{x},\boldsymbol{y}) = \sum_{i=1}^{m} \frac{|x_i - y_i|}{|x_i + y_i|}$$

**Chebychev:**

$$D(\boldsymbol{x},\boldsymbol{y}) = \max_{i=1}^{m} |x_i - y_i|$$

**Quadratic:**

$$D(\boldsymbol{x},\boldsymbol{y}) = (\boldsymbol{x}-\boldsymbol{y})^T Q (\boldsymbol{x}-\boldsymbol{y}) = \sum_{j=1}^{m} \left( \sum_{i=1}^{m} (x_i - y_i) q_{ji} \right) (x_j - y_j)$$

Q is a problem-specific positive definite $m \times m$ weight matrix

**Mahalanobis:**

$$D(\boldsymbol{x},\boldsymbol{y}) = [\det V]^{1/m} (\boldsymbol{x}-\boldsymbol{y})^{\mathrm{T}} V^{-1} (\boldsymbol{x}-\boldsymbol{y})$$

$V$ is the covariance matrix of $A_1..A_m$, and $A_j$ is the vector of values for attribute $j$ occuring in the training set instances $1..n$.

**Correlation:**

$$D(\boldsymbol{x},\boldsymbol{y}) = \frac{\sum_{i=1}^{m} (x_i - \overline{x_i})(y_i - \overline{y_i})}{\sqrt{\sum_{i=1}^{m} (x_i - \overline{x_i})^2 \sum_{i=1}^{m} (y_i - \overline{y_i})^2}}$$

$\overline{x_i} = \overline{y_i}$ and is the average value for attribute $i$ occuring in the training set.

**Chi-square:**

$$D(\boldsymbol{x},\boldsymbol{y}) = \sum_{i=1}^{m} \frac{1}{sum_i} \left( \frac{x_i}{size_{\boldsymbol{x}}} - \frac{y_i}{size_{\boldsymbol{y}}} \right)^2$$

$sum_i$ is the sum of all values for attribute $i$ occuring in the training set, and $size_{\boldsymbol{x}}$ is the sum of all values in the vector $\boldsymbol{x}$.

**Kendall's Rank Correlation:**

$$D(\boldsymbol{x},\boldsymbol{y}) = 1 - \frac{2}{n(n-1)} \sum_{i=1}^{m} \sum_{j=1}^{i-1} \operatorname{sign}(x_i - x_j)\operatorname{sign}(y_i - y_j)$$

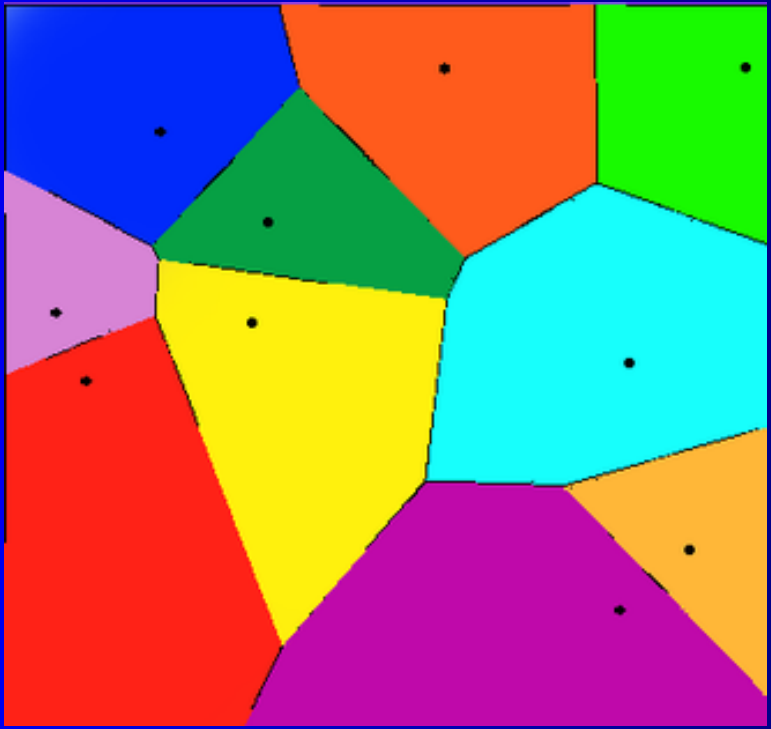
sign($x$)=-1, 0 or 1 if $x < 0$, $x = 0$, or $x > 0$, respectively.

Figure 1. Equations of selected distance functions.
($\boldsymbol{x}$ and $\boldsymbol{y}$ are vectors of $m$ attribute values).

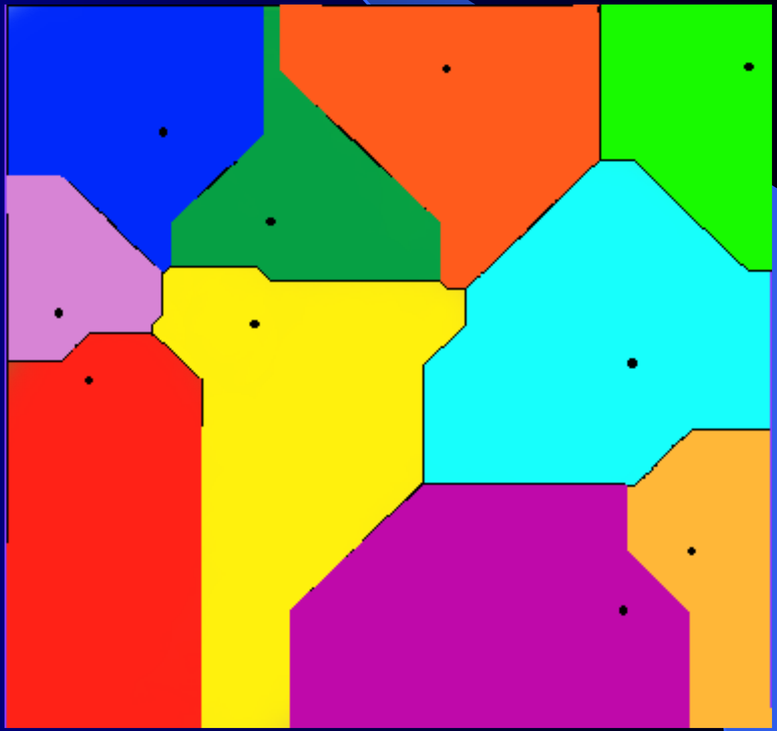
## Decision Surface

- Combining all the appropriate intersections gives a Voronoi diagram

Euclidean distance



Manhattan distance



# *k*-Nearest Neighbor (cont)

- Usually do distance weighted voting where each nearest neighbor vote is scaled inversely to its distance
- $$y_q = \operatorname{argmax}_{c \in C} \left\{ \sum_{(\mathbf{x_i}, y_i) \in NN} w_i \cdot \mathbf{1}(c = y_i) \right\}$$
- $w_i = 1$ for the non-weighted version
- Inverse of distance squared is a common weight: $w_i = 1/d(\mathbf{x_q}, \mathbf{x_i})^2$
- Gaussian is another common distance weight
- In this case the *k* value is more robust, could let *k* be even and/or be larger (even all points if desired), because the more distant points have negligible influence

## *Challenge Question* - *k*-Nearest Neighbor

- Assume the following data set
- Assume a new point (2, 6)
  - For nearest neighbor distance use Manhattan distance
  - What would the output be for 3-nn with no distance weighting? What is the total vote?
  - What would the output be for 3-nn with squared inverse distance weighting? What is the total vote?

A. A A

B. A B

C. B A

D. B B

E. None of the above

| *x* | *y* | *Label* |
|---|---|---|
| 1 | 5 | A |
| 0 | 8 | B |
| 9 | 9 | B |
| 10 | 10 | A |

## *Challenge Question* - *k*-Nearest Neighbor

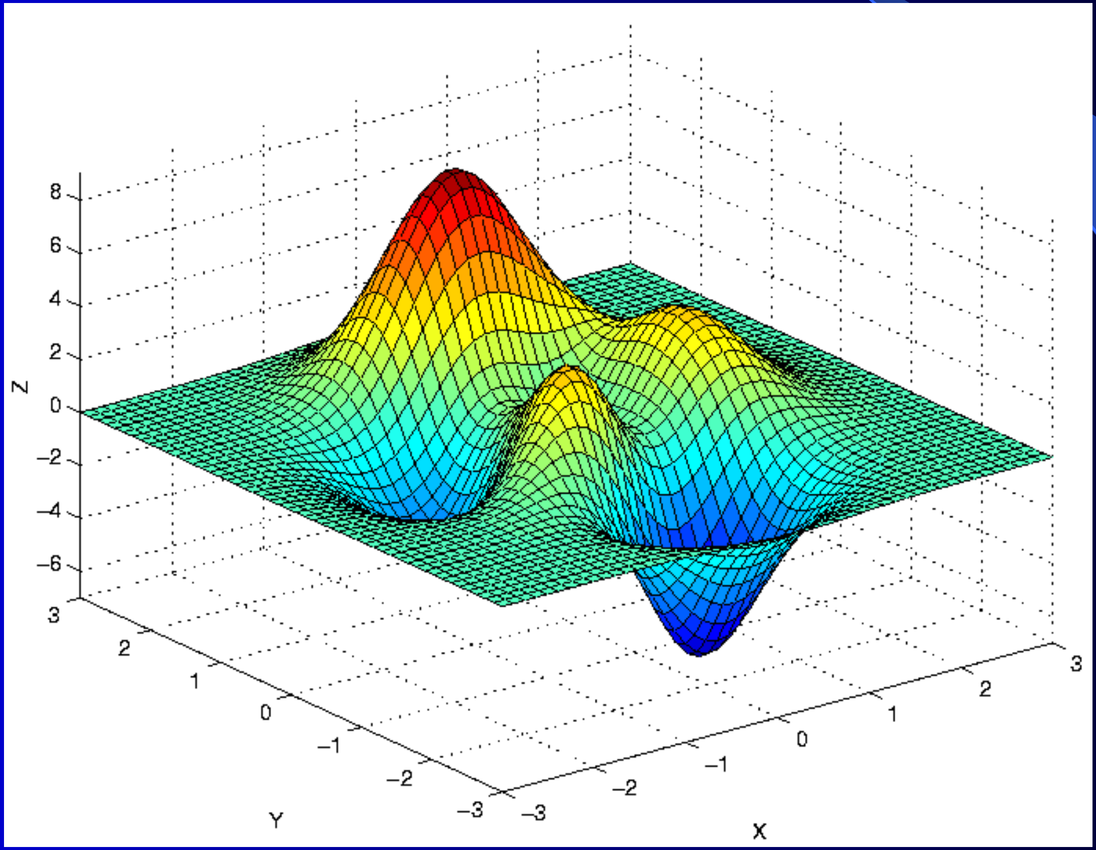
- Assume the following data set
- Assume a new point (2, 6)
  - For nearest neighbor distance use Manhattan distance
  - What would the output be for 3-nn with no distance weighting? What is the total vote? – B wins with vote 2 out of 3
  - What would the output be for 3-nn with distance weighting? What is the total vote? A wins with vote .25 vs B vote of .0625+.01=.0725

| *x* | *y* | *Label* | *Distance* | *Weighted Vote* |
|---|---|---|---|---|
| 1 | 5 | A | 1 + 1 = 2 | $1/2^2$ = .25 |
| 0 | 8 | B | 2 + 2 = 4 | $1/4^2$ = .0625 |
| 9 | 9 | B | 7 + 3 = 10 | $1/10^2$ = .01 |
| 10 | 10 | A | 8 + 4 = 12 | $1/12^2$ = .0069 |

## K-NN REGRESSION

## Regression with *k*-nn

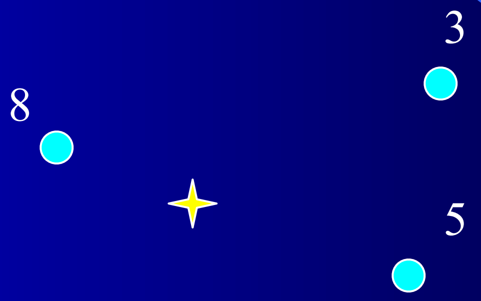
- Can do regression by letting the output be the mean of the *k* nearest neighbors



## Weighted Regression with *k*-nn

- Can do weighted regression by letting the output be the weighted mean of the *k* nearest neighbors
- For distance weighted regression
- $\hat{f}(\mathbf{x}_q) = \frac{\sum_{i=1}^{k} w_i \cdot f(\mathbf{x}_i)}{\sum_{i=1}^{k} w_i}$; where $w_i = 1/d(\mathbf{x}_q, \mathbf{x}_i)^2$
- Where *f*(*x*) is the output value for instance *x*
- $w_i = 1$ for non-weighted

## Regression Example



- What is the value of the new instance?
- Assume dist($x_q$, $x_8$) = 2, dist($x_q$, $x_5$) = 3, dist($x_q$, $x_3$) = 4
- $f(x_q) = (8/2^2 + 5/3^2 + 3/4^2) / (1/2^2 + 1/3^2 + 1/4^2) = 2.74 / .42 = 6.5$
- The denominator renormalizes the value

# *k*-Nearest Neighbor Homework

- Assume the following training set
- Assume a new point (.5, .2)
  - Use Manhattan distance and show work
  - What would the output class for 3-nn be with no distance weighting?
  - What would the output class for 3-nn be with squared inverse distance weighting?
  - What would the 3-nn regression value be for the point if we used the regression values rather than class labels? Show results for *both* no distance weighting and squared inverse distance weighting.

| *x* | *y* | *Class Label* | *Regression Label* |
|---|---|---|---|
| .3 | .8 | A | .6 |
| -.3 | 1.6 | B | -.3 |
| .9 | 0 | B | .8 |
| 1 | 1 | A | 1.2 |

## EXTRAS

## Feature Weighting

- Normalize Features!
- One of the main weaknesses of nearest neighbor is irrelevant features, since they can dominate the distance
  - Example:  assume 2 relevant and 10 irrelevant features
- Most learning algorithms weight the features (e.g. MLP and Decisions Trees do higher order weighting of features)
- Could do feature weighting - No longer lazy evaluation since you need to come up with a portion of your feature weights before generalizing
- Still an open area of research
  - Higher order weighting – 1st order helps, but not enough
  - <u>Even if all features are relevant features, all distances become similar as number of features increases, since not all features are relevant at the same time, and the currently irrelevant ones can dominate distance</u>
  - An issue with all pure distance based techniques, need higher-order weighting to ignore *currently* irrelevant features
  - Dimensionality reduction can be useful (feature pre-processing, PCA, etc.)

## Reduction Techniques

- Create a subset or other representative set of prototype nodes
  - Faster execution, and could even improve accuracy if noisy instances removed

- Approaches
  - Leave-one-out reduction - Drop instance if it would still be classified correctly
  - Growth algorithm - Only add instance if it is not already classified correctly - both order dependent, similar results
  - More global optimizing approaches
  - Just keep central points – lower accuracy (mostly linear Voronoi decision surface), best space savings
  - Just keep border points, best accuracy (pre-process noisy instances – Drop5)
  - Drop 5 (Wilson & Martinez) maintains good accuracy with approximately 15% of the original instances
    - Wilson, D. R. and Martinez, T. R., Reduction Techniques for Exemplar-Based Learning Algorithms, *Machine Learning Journal,* vol. **38**, no. 3, pp. 257-286, 2000.

# *k*-Nearest Neighbor Summary

- Very powerful yet simple scheme which does well on many tasks
- Overfitting (less of an issue) handled by using larger *k*
- Struggles with irrelevant inputs
  - Needs better incorporation of feature weighting schemes
- Issues with distance with very high dimensionality tasks
  - Too many features wash out effects of the specifically important ones (akin to the irrelevant feature problem)
  - May need distance metrics other than Euclidean distances
- Also may be helpful to reduce total # of instances
  - Efficiency
  - Sometimes accuracy