TEXAS A&M UNIVERSITY

CSCE 633: Machine Learning

Lecture 3

Outline

- K-Nearest Neighbor (K-NN)
    - Basics
    - Example & Representation
- Practical use of K-NN
    - How to choose the right K and distance metric?
    - How to pre-process the data?
    - What to do in the case of a tie?
- Variations
    - Condensed Nearest Neighbor
    - Weighted Distance K-NN

Outline

- K-Nearest Neighbor (K-NN)
    - Basics
    - Example & Representation
- Practical use of K-NN
    - How to choose the right K and distance metric?
    - How to pre-process the data?
    - What to do in the case of a tie?
- Variations
    - Condensed Nearest Neighbor
    - Weighted Distance K-NN

K-Nearest Neighbor: Example

Recognizing types of Iris flowers (by R. Fisher)

setosa

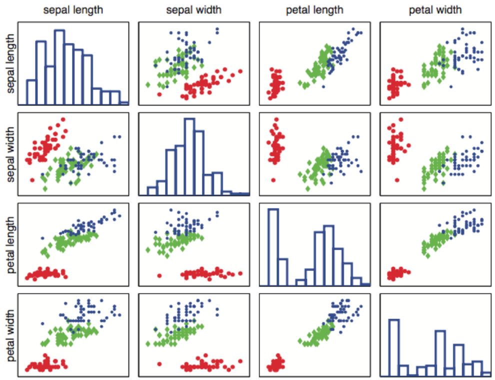
versicolor

virginica

Features: the widths and lengths of sepal and petal

K-Nearest Neighbor: Example

Visualizing features to get better intuition about our data

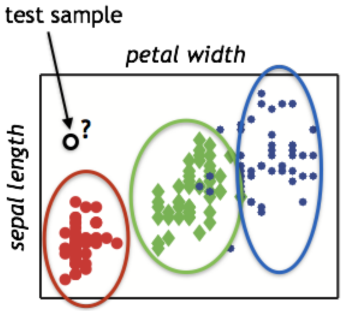


Each colored datapoint is one sample

setosa, versicolor, virginica

K-Nearest Neighbor: Example

Using two features: sepal length & petal width



setosa, versicolor, virginica

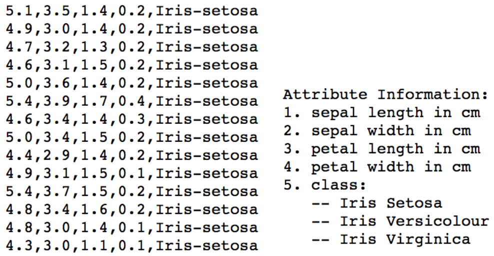
Test sample is closer to red cluster → label it as setosa

K-Nearest Neighbor: Example

Recognizing types of Iris flowers (by R. Fisher)

Often data is organized in a table

Each row is one sample with 4 features and 1 label



```
5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
4.7,3.2,1.3,0.2,Iris-setosa
4.6,3.1,1.5,0.2,Iris-setosa
5.0,3.6,1.4,0.2,Iris-setosa
5.4,3.9,1.7,0.4,Iris-setosa
4.6,3.4,1.4,0.3,Iris-setosa
5.0,3.4,1.5,0.2,Iris-setosa
4.4,2.9,1.4,0.2,Iris-setosa
4.9,3.1,1.5,0.1,Iris-setosa
5.4,3.7,1.5,0.2,Iris-setosa
4.8,3.4,1.6,0.2,Iris-setosa
4.8,3.0,1.4,0.1,Iris-setosa
4.3,3.0,1.1,0.1,Iris-setosa
```

```
Attribute Information:
1. sepal length in cm
2. sepal width in cm
3. petal length in cm
4. petal width in cm
5. class:
   -- Iris Setosa
   -- Iris Versicolour
   -- Iris Virginica
```

[Source: `https://archive.ics.uci.edu/ml/datasets/iris`]

K-Nearest Neighbor: Representation

Training Data

- N samples/datapoints/instances: $\mathcal{D}^{train} = \{(\mathbf{x_1}, y_1), \ldots, (\mathbf{x_N}, y_N)\}$
- Used for learning representation $f : \mathbf{x} \rightarrow y$

Testing Data

- M samples/datapoints/instances: $\mathcal{D}^{test} = \{(\mathbf{x_1}, y_1), \ldots, (\mathbf{x_M}, y_M)\}$
- Used to assess how well $f(\cdot)$ will do in predicting an unseen sample

Train and test data should not overlap: $\mathcal{D}^{train} \cap \mathcal{D}^{test} = \emptyset$

K-Nearest Neighbor: Representation

Classify data into one out of multiple classes

- Input: $\mathbf{x} \in \mathbb{R}^D$ (features, attributes, etc.)
- Output: $y \in \{1, 2, \ldots, C\}$ (labels)
- Model: $f : \mathbf{x} \rightarrow y$

Special case: binary classification (C=2)

- Output: $y \in \{1, 2\}$ or $\{0, 1\}$ or $\{-1, 1\}$, etc.

K-Nearest Neighbor: Representation

Nearest Neighbor (or 1-Nearest Neighbor, 1-NN)

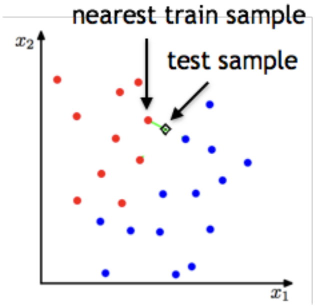
- Assigns test sample **x** to the closest training sample
- Model

$$y = f(\mathbf{x}) = y_{nn(\mathbf{x})}$$

$$nn(\mathbf{x}) = arg \min_{n=1,\ldots,N} \|\mathbf{x} - \mathbf{x_n}\|^2 = arg \min_{n=1,\ldots,N} \sum_{d=1}^{D} (x_d - x_{nd})^2$$

K-Nearest Neighbor: Representation

Nearest Neighbor (or 1-Nearest Neighbor, 1-NN): Example

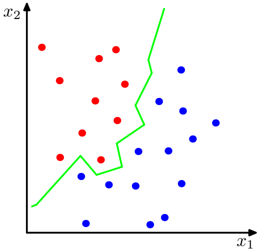


The nearest point to test sample **x** is a red training instance, therefore **x** will be labeled as red.

K-Nearest Neighbor: Representation

Nearest Neighbor (or 1-Nearest Neighbor, 1-NN): Example

Decision boundary: For every point in the space, we can determine its label using the nearest neighbor rule. This gives us a decision boundary that partitions the space into different regions.



The above decision boundary is very sensitive to noise
What would be the solution for this?

K-Nearest Neighbor: Representation

Increase number of nearest neighbors to use

- 1-nearest neighbor: $nn_1(\mathbf{x}) = arg\min_{n \in \{1,\ldots,N\}} \|\mathbf{x} - \mathbf{x_n}\|_2^2$
- 2-nearest neighbor: $nn_2(\mathbf{x}) = arg\min_{n \in \{1,\ldots,N\} \setminus nn_1(\mathbf{x})} \|\mathbf{x} - \mathbf{x_n}\|_2^2$
- 3-nearest neighbor:
  $nn_3(\mathbf{x}) = arg\min_{n \in \{1,\ldots,N\} \setminus \{nn_1(\mathbf{x}), nn_2(\mathbf{x})\}} \|\mathbf{x} - \mathbf{x_n}\|_2^2$

The set of K-nearest neighbors is

$$knn(\mathbf{x}) = \{nn_1(\mathbf{x}), \ldots, nn_K(\mathbf{x})\}$$

Neighbors $nn_1, \ldots, nn_K$ in order of increasing distance from sample $\mathbf{x}$

K-Nearest Neighbor: Representation

K-NN Model

- Each neighbor in $knn(\mathbf{x}) = \{nn_1(\mathbf{x}), \ldots, nn_K(\mathbf{x})\}$ votes one class
- Count the number of neighbors that have voted each class

$$v_c = \sum_{k \in knn(\mathbf{x})} \mathbb{I}(y_k = c), \quad c = 1, \ldots, C$$

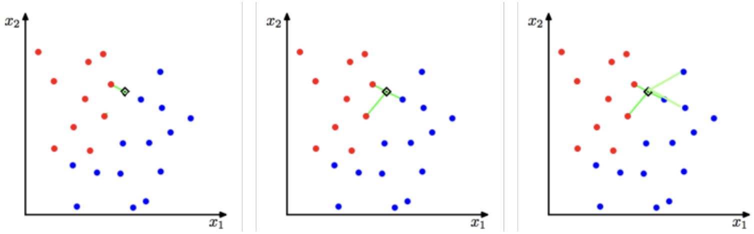
- Assign test sample $\mathbf{x}$ to to the majority class membership of the K neighbors

$$y = f(\mathbf{x}) = arg \max_{c=1,\ldots,C} v_c$$

K-Nearest Neighbor: Representation

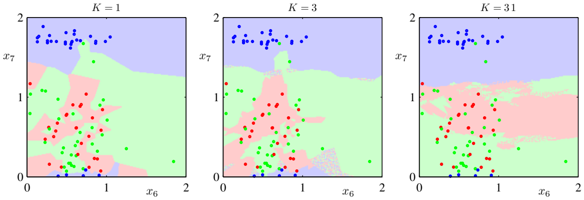
K-NN Example

K=1, label=red | K=3 label=red | K=5, label=blue



K-Nearest Neighbor: Representation

K-NN Decision Boundary



Number of neighbors K controls the degree of smoothing

$K \downarrow$: many small regions of each class

$K \uparrow$: fewer larger regions of each class

K-Nearest Neighbor: Computational Cost

Question: What is the computational cost of K-NN for labelling one test sample $\mathbf{x} \in \mathbb{R}^D$ given that we have N training data?

A) $O(ND)$
B) $O(KD)$
C) $O(ND + NK)$
D) $O(NKD)$

K-Nearest Neighbor: Computational Cost

Question: What is the computational cost of K-NN for labelling one test sample $\mathbf{x} \in \mathbb{R}^D$ given that we have N training data?

A) $O(ND)$
B) $O(KD)$
C) $O(ND + NK)$
D) $O(NKD)$

The correct answer is C.

The cost of measuring the distance between the test sample and every sample in the training data is $O(D)$
The cost of computing distances for all N train samples is $O(ND)$
The cost of finding the K closets samples is $O(NK)$ (can be optimized)
So the total cost is $O(ND + NK)$

[Nice video source: `https://www.youtube.com/watch?v=UPAnUE_g5SQ`]

Parametric v.s. non-parametric models

- Many possible ways to categorize learning models
- Non-parametric models (or instance/memory-based)
    - more flexible
    - computationally intractable for large datasets
    - e.g. K-NN
- Parametric models
    - faster to use
    - make strong assumptions about data
    - e.g. linear regression

Curse of dimensionality

- In a high-dimensional space:
    - all intuition fails in higher dimensions
    - harder to generalize
    - harder to systematically search
    - harder to accurately approximate a target function
- K-NN is prone to high dimensions

K-Nearest Neighbor: Theoretical guarantees

Modeling expected mistakes

- Assume data $(\mathbf{x}, y)$ is drawn from the joint distribution $p(\mathbf{x}, y)$
    - In practice, $p(\mathbf{x}, y)$ is unknown
- Assume we use classifier $f(\cdot)$ (2 classes for simplicity)
- Classification mistake on $\mathbf{x}$ with ground truth y, or "0/1 loss function" $L(f(\mathbf{x}), y) = \begin{cases} 0, & f(\mathbf{x}) = y \\ 1, & f(\mathbf{x}) \neq y \end{cases}$
- Expected classification mistake on $\mathbf{x}$, or "expected conditional risk"
$$R(f, \mathbf{x}) = \mathbb{E}_{y \sim p(y|\mathbf{x})} L(f(\mathbf{x}), y)$$
$$= P(y = 1|\mathbf{x})\mathbb{I}(f(\mathbf{x}) = 0) + P(y = 0|\mathbf{x})\mathbb{I}(f(\mathbf{x}) = 1)$$

K-Nearest Neighbor: Theoretical guarantees

Bayes Optimal Classifier (2 class)

- For any labeling function $f(\cdot)$, we have $R(f^*, \mathbf{x}) \leq R(f, \mathbf{x})$
- Namely, $f^*(\cdot)$ is optimal, i.e. the Bayes optimal classifier always yields the lowest expected conditional risk
- In practice
    - The Bayes optimal classifier is generally not computable as it assumes the knowledge of $p(\mathbf{x}, y)$ and $p(y|\mathbf{x})$
    - However, it is useful as a conceptual tool to formalize how well a classifier can do without knowing the joint distribution

K-Nearest Neighbor: Theoretical guarantees

How well does K-NN do wrt Bayes Optimal Classifier? (2-class)

Theorem

For the K-NN labeling function f^{K-NN} for binary classification, we have:

$R(f^*) \leq R(f^{K-NN}) \leq 2R(f^*)(1 - R(f^*)) \leq 2R(f^*)$

Given infinite data, K-NN is guaranteed to approach the Bayes error rate under ideal conditions.

In short K-NN seems to do a reasonable thing.

K-Nearest Neighbor: Advantages & Disadvantages

Advantages	Disadvantages
1. Intuitive	1. Computationally expensive for large datasets
2. Simple & easy to implement	2. We need to keep the training data in memory
3. Guarantees that it "works"	3. Choosing the right K can be tricky
	4. Sensitive to noisy features
	5. May perform badly in high dimensions

Outline

- K-Nearest Neighbor (K-NN)
    - Basics
    - Example & Representation
- Practical use of K-NN
    - How to choose the right K and distance metric?
    - How to pre-process the data?
    - What to do in the case of a tie?
- Variations
    - Condensed Nearest Neighbor
    - Weighted Distance K-NN

K-Nearest Neighbor: Practical Use

Two practical issues about K-NN

- Choosing number of neighbors K
- Choosing distance metric between test sample $\mathbf{x}$ and training data $\mathbf{x_n}$, $n = 1, \ldots, N$

$$\|\mathbf{x} - \mathbf{x_n}\|_p = \left(\sum_{d=1}^{D} |x_d - x_{nd}|^p \right)^{1/p}, \quad p \geq 1$$

The above are called hyperparameters. They are not estimated through the learning process, but are externally set before the beginning of the learning process.

K-Nearest Neighbor: Model parameters and Hyperparameters

- Hyperparameters: Parameters set before the beginning of the learning process
  - For K-NN: number of neighbors K, distance metric
- Hyperparameter tuning: The process of choosing a set of optimal hyperparameters for the learning process
- Model parameters: The parameters learned during the learning process
  - For K-NN: As we said, KNN is non-parametric. Last lecture, parameters were the weights of the linear perceptron. Next lecture, we will list these for linear regression.

K-Nearest Neighbor: Hyperparameter tuning using a validation set

Training Data (or training set)

- N samples/datapoints/instances: $\mathcal{D}^{train} = \{(\mathbf{x_1}, y_1), \dots, (\mathbf{x_N}, y_N)\}$
- Used for learning representation $f : \mathbf{x} \rightarrow y$

Testing Data

- M samples/datapoints/instances: $\mathcal{D}^{test} = \{(\mathbf{x_1}, y_1), \dots, (\mathbf{x_M}, y_M)\}$
- Used to assess how well $f(\cdot)$ will do in predicting an unseen sample

Validation Data (or validation/development set)

- L samples/datapoints/instances: $\mathcal{D}^{dev} = \{(\mathbf{x_1}, y_1), \dots, (\mathbf{x_L}, y_L)\}$
- Used to optimize hyperparameter(s)

Train, test, validation data should not overlap

$$\mathcal{D}^{train} \cap \mathcal{D}^{test} \cap \mathcal{D}^{dev} = \emptyset$$

Recipe for hyperparameter tuning with validation set

- For each hyperparameter value (or combination of values)
  - Train model using $\mathcal{D}^{train}$
  - Evaluate model performance on $\mathcal{D}^{dev}$

    For K-NN: For $K = 1, 3, \ldots$
    - For all $\mathbf{x} \in \mathcal{D}^{dev}$, assign each sample to the majority class determined by its K neighbors from the train data $\mathcal{D}^{train}$ (see previous slide, K-Nearest Neighbor: Representation, K-NN Model)
    - Evaluate accuracy/error rate, etc. using all $\mathbf{x} \in \mathcal{D}^{dev}$
- Chose the model with the best performance on $\mathcal{D}^{dev}$

  For K-NN: Choose K with the best accuracy/error rate, etc.
- Evaluate the model on the test set $\mathcal{D}^{test}$

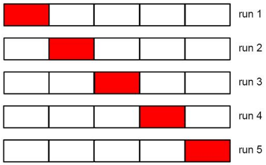
  For K-NN: Similarly as we did for evaluating on $\mathcal{D}^{dev}$

Cross-validation

What if we don't have a validation set?

We perform cross-validation

- Split train data into S equal parts
- Use each part as a validation set and all others as train set
- Chose the hyperparameter value (or combination of values) that results in best average performance.
    - Average computed across validation sets from all folds



Recipe for cross-validation

- Split train data into S equal parts, each noted as $\mathcal{D}_s^{train}$, $s = 1,...,S$
- For each hyperparameter value (e.g. $K = 1, 3, \ldots$)
    - For each $s = 1, \ldots, S$
        - Train model using $\mathcal{D}^{train} \setminus \mathcal{D}_S^{train}$
        - Evaluate model performance (noted as E_s) on $\mathcal{D}_s^{train}$
    - Compute average performance for current hyperparameter $E = \frac{1}{s} \sum_{s=1}^{S} E_s$
- Chose the hyperparameter corresponding to best average performance E
- Use the best hyperparameter to train on a model using all $\mathcal{D}^{train}$
- Evaluate the last model trained on all $\mathcal{D}^{train}$ using the best hyperparameter on $\mathcal{D}^{test}$

How to measure "neighbor nearness" with other distances?

Previously we have used the euclidean distance
(dominated by noise if #useful features is much smaller than #noisy features)

$$nn(\mathbf{x}) = arg \min_{n=1,\ldots,N} \|\mathbf{x} - \mathbf{x_n}\|_2^2 = arg \min_{n=1,\ldots,N} \sum_{d=1}^{D} (x_d - x_{nd})^2$$

We can also use alternative distances

- L_1, Manhattan, or city block distance

$$nn(\mathbf{x}) = arg \min_{n=1,\ldots,N} \|\mathbf{x} - \mathbf{x_n}\|_1 = arg \min_{n=1,\ldots,N} \sum_{d=1}^{D} |x_d - x_{nd}|$$

- Hamming distance (discrete data, # non-matching attributes)

$$nn(\mathbf{x}) = arg \min_{n=1,\ldots,N} \sum_{d=1}^{D} 1(x_d \neq x_{nd})$$

where $1(a) = 1$ if a true, 0 otherwise

How to measure “neighbor nearness” with other distances?

Different distances might result in different sorting between samples

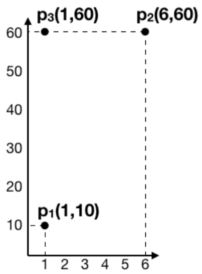
Example 1

- train data: $A(3,1,1)$ belonging to class 1, $B(2,2,2)$ belonging to class 2
- test data: $X(1,1,1)$
- $l_1(A,X) = |2| + |0| + |0| = 2$
  $l_1(B,X) = |1| + |1| + |1| = 3$
  $\rightarrow$ X in class 1
- $l_2(A,X) = \sqrt{|2|^2 + |0| + |0|} = 2$
  $l_2(B,X) = \sqrt{|1|^2 + |1|^2 + |1|^2} = \sqrt{3}$
  $\rightarrow$ X in class 2

How to measure "neighbor nearness" with other distances?

Distances depend on the units of the features

Example: one feature in cm and one in mm



$$dist(\mathbf{p_1}, \mathbf{p_2}) = \sqrt{(60-10)^2 + (6-1)^2} = 2525$$
$$dist(\mathbf{p_1}, \mathbf{p_3}) = \sqrt{(60-10)^2 + (1-1)^2} = 2500$$

$$\text{rel\_change}_{\text{x-axis}} = 100 \cdot \tfrac{6-1}{6}\% = 83\%$$

$$\text{rel\_change}_{\text{dist}} = 100 \cdot \tfrac{2525-2500}{2500}\% = 1\%$$

Proximity can change due to change in feature scales

Larger-scale features tend to inflate distance measure

How to measure “neighbor nearness” with other distances?

Normalize data so that they have comparable range

Value changes across any feature can be equally reflected to the distance metric, when features are normalized

$$x_{nd} := \frac{x_{nd} - \bar{x}_d}{s_d}$$

$$\bar{x}_d = \frac{1}{N} \sum_n x_{nd}, \quad s_d = \frac{1}{N-1} \sum_n (x_{nd} - \bar{x}_d)^2$$

How to measure “neighbor nearness” with other distances?

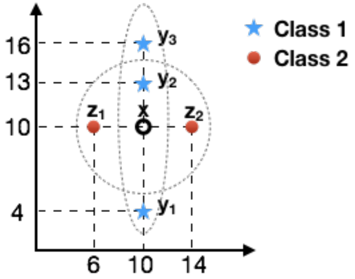
Question

Assume the following train samples:
Class 1: $y_1 = (10, 4)$, $y_2 = (10, 13)$, $y_3 = (10, 16)$
Class 2: $z_1 = (6, 10)$, $z_2 = (14, 10)$
And the following distance metrics:

$$D_1 = (\mathbf{x} - \mathbf{x_k})^T \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} (\mathbf{x} - \mathbf{x_k})$$

$$D_2 = (\mathbf{x} - \mathbf{x_k})^T \begin{bmatrix} 1 & 0 \\ 0 & 1/3 \end{bmatrix} (\mathbf{x} - \mathbf{x_k})$$

In which class would a K-NN classifier (K=3) classify sample $\mathbf{x} = (10, 10)$ using distances D_1 and D_2?



A) Class 1 for both D_1 and D_2
B) Class 2 for both D_1 and D_2
C) Class 1 for D_1, Class 2 for D_2
D) Class 2 for D_1, Class 1 for D_2

How to measure "neighbor nearness" with other distances?

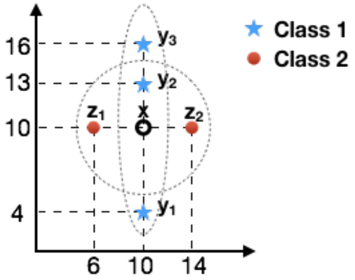
The correct answer is D

$$D_1 = (x_1 - x_{k1})^2 + (x_2 - x_{k2})^2$$
$$D_2 = (x_1 - x_{k1})^2 + \frac{1}{3}(x_2 - x_{k2})^2$$

Train Sample	D_1	D_2
$\mathbf{y_1}$	36	12
$\mathbf{y_2}$	9	3
$\mathbf{y_3}$	36	12
$\mathbf{z_1}$	16	16
$\mathbf{z_2}$	16	16

The 3 nearest neighbors of $\mathbf{x}$ based on D_1 are $\mathbf{y_2}$, $\mathbf{z_1}$, $\mathbf{z_2}$, so K-NN decides Class 2

The 3 nearest neighbors of $\mathbf{x}$ based on D_2 are $\mathbf{y_1}$, $\mathbf{y_2}$, $\mathbf{y_3}$, so K-NN decides Class 1



A) Class 1 for both D_1 and D_2

B) Class 2 for both D_1 and D_2

C) Class 1 for D_1, Class 2 for D_2

D) Class 2 for D_1, Class 1 for D_2

What to do in the case of a tie?

- Randomly assign a class
- Assign the class with the highest frequency in the training set
- Classify the test sample according to the closest training sample (1-NN), or according to the K-1 or K+1 training samples
- Decide by assigning a weight to each vote according to its distance from the test sample (i.e. weighted K-NN)

Outline

- K-Nearest Neighbor (K-NN)
    - Basics
    - Example & Representation
- Practical use of K-NN
    - How to choose the right K and distance metric?
    - How to pre-process the data?
    - What to do in the case of a tie?
- Variations
    - Condensed Nearest Neighbor
    - Weighted Distance K-NN

Condensed Nearest Neighbor

- Approximate way to reduce time/space complexity of K-NN
- Decrease number of stored training samples
- Select the smallest subset $\mathcal{Z}$ of $\mathcal{D}^{train}$, such that when $\mathcal{Z}$ is used, error does not increase
- Subset $\mathcal{Z}$ is determined **once before running K-NN**, therefore the high computational cost is not during testing
- Local search that depends on the order of training samples
    - Non-unique solution

Condensed Nearest Neighbor

Algorithm 1 Pseudocode for Condensed Nearest Neighbor

```
1: Randomly initialize $\mathcal{Z}$ with a sample from $\mathcal{D}^{train}$
2: repeat
3:     for $\mathbf{x} \in \mathcal{D}^{train}$ (in random order) do
4:         Find $\mathbf{x}' \in \mathcal{Z}$ closest to $\mathbf{x}$
5:         if class($\mathbf{x}$)$\neq$class($\mathbf{x}'$) then
6:             add $\mathbf{x}$ to $\mathcal{Z}$
7:         end if
8:     end for
9: until $\mathcal{Z}$ does not change
```

Distance Weighting

- K-NN assumes that similar instances mean similar things
- Hence samples closest to the query point might matter most
- Weight each neighbor by their closeness to the test sample

$$w_k = \frac{\exp\left(-dist(\mathbf{x}, nn_k(\mathbf{x}))\right)}{\sum_{k \in knn(\mathbf{x})} \exp(-dist(\mathbf{x}, nn_k(\mathbf{x})))}$$

$$v_c = \sum_{k \in knn(\mathbf{x})} w_k \cdot \mathbb{I}(y_k = c), \quad c = 1, \ldots, C$$

$$y = f(\mathbf{x}) = arg \max_{c=1,\ldots,C} v_c$$

Summary

- Described a simple learning algorithm (K-NN)
  - non-parametric (instance-based) learning algorithm
  - "similar inputs will have similar outputs"
  - high computational cost for large data
  - guaranteed to approach Bayes error rate under ideal conditions
- Practical issues
  - number of neighbors, type of distance $\rightarrow$ (cross-)validation
- Computationally "cheaper" options (e.g. condensed K-NN)
- Reading materials
  - Alpaydin 8.1-8.5