knn

why kNN is an important algorithm

instance-based learning

shares elements of human reasoning

how knn works

- lazy learning:

  no explicit model, pushes work to prediction time

Let $T = \{(x_j, y_j)\}$ where $j \in \{1,\ldots,n\}$

Classify Example e

  prior queue pq of size k

  forall $(x_j, y_j) \in T$

    if $(dist(e, x_j) < pq.max)$ then $pq.enqueue(x_j, y_j)$

  prediction $=$ combine$(y_j \in pq)$

  return prediction

- distance
  1. similarity vs. distance
  2. distance measures
     - hamming distance: # features examples differ on
     - manhattan
     - euclidean
     - value difference metric:

- **Value difference metric:** Attribute values are close if they make similar predictions

$$\text{vdm}(x_{i.k}, x_{j,k}) = \sum_{c=1}^{|Y|} |P(y_c|x_{i,k}) - P(y_c|x_{j,k})|$$

here $x_{i,k}$ means the value i of attribute k

- jaccard: set comparisons

- **Jaccard Similarity**: $\text{sim}(S_i, S_j) = \frac{S_i \cap S_j}{S_i \cup S_j}$
- **Jaccard Distance**: $1 - \text{sim}(S_i, S_j)$

- edit distance (for strings)
- cosine similarity (for vectors)

$$\text{sim}(x_i, x_j) = \frac{\sum_{k=1}^{d} x_{i,k} x_{j,k}}{\sqrt{\sum_{k=1}^{d} x_{i,k}^2}\sqrt{\sum_{k=1}^{d} x_{j,k}^2}}$$

- how to combine
  - for predicting classes

    distance weighted kNN

$$f(x_q) = \underset{y}{\text{argmax}} \sum_{i=1}^{k} w_i\, \mathbb{I}[y_j = y] \quad \text{where} \quad w_i = \frac{1}{dist(x_q, x_i)^2}$$

Weight of each neighbor

Indicator: 1 if $y_j = y$ else it is 0

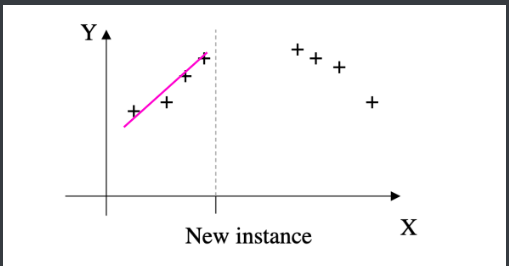
"argmax" means to find a y value which would be assigned to the sample, that maximizes this function;

  - for predicting real values

    depends.

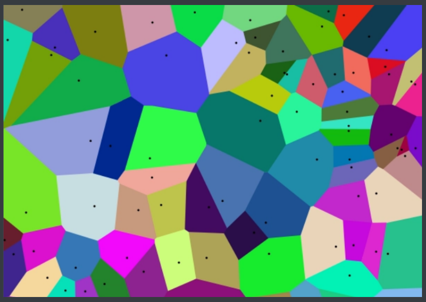
    1. obvious choice would be average
    2. could also build local model (learn model on k nearest neighbors),

       for example, a new linear model would work in case of:

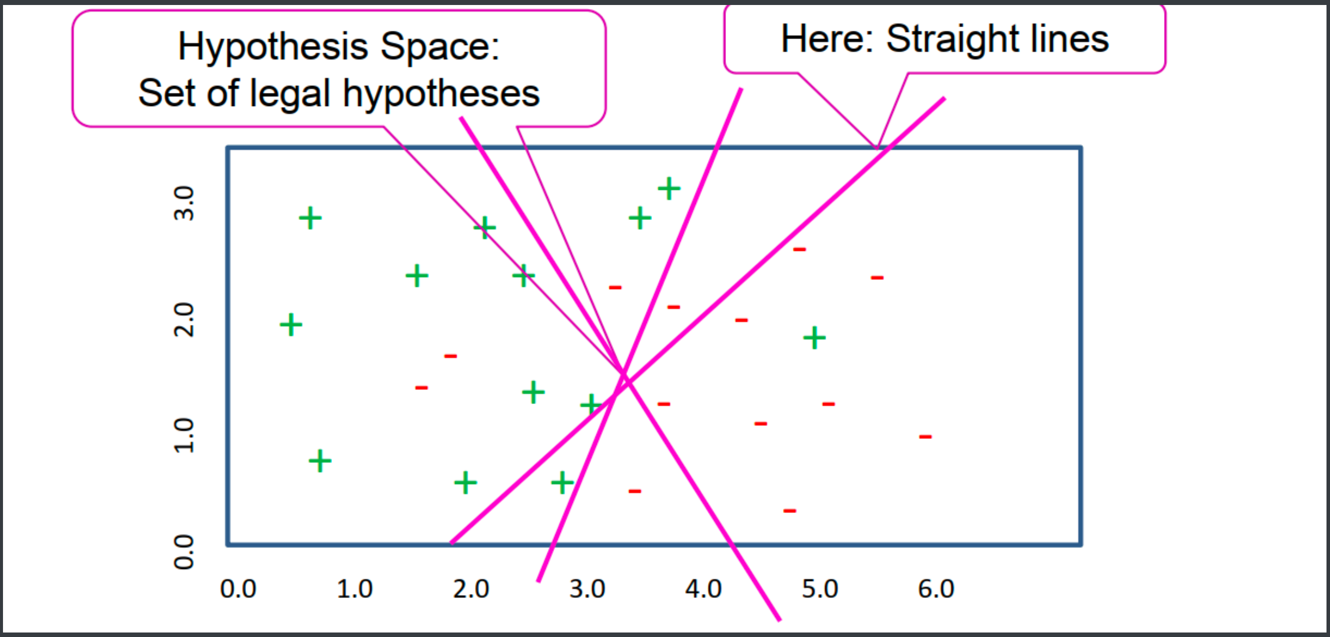


expressiveness

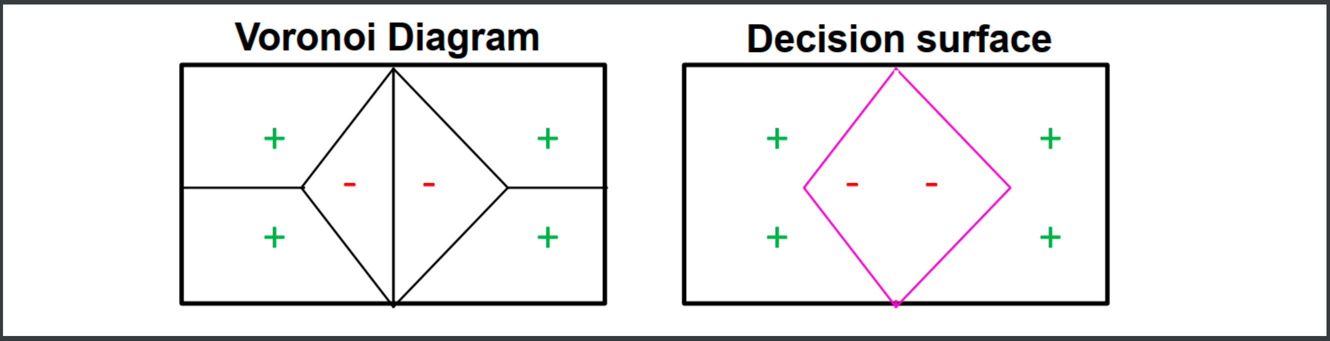
- voronoi diagram: understanding 1nn



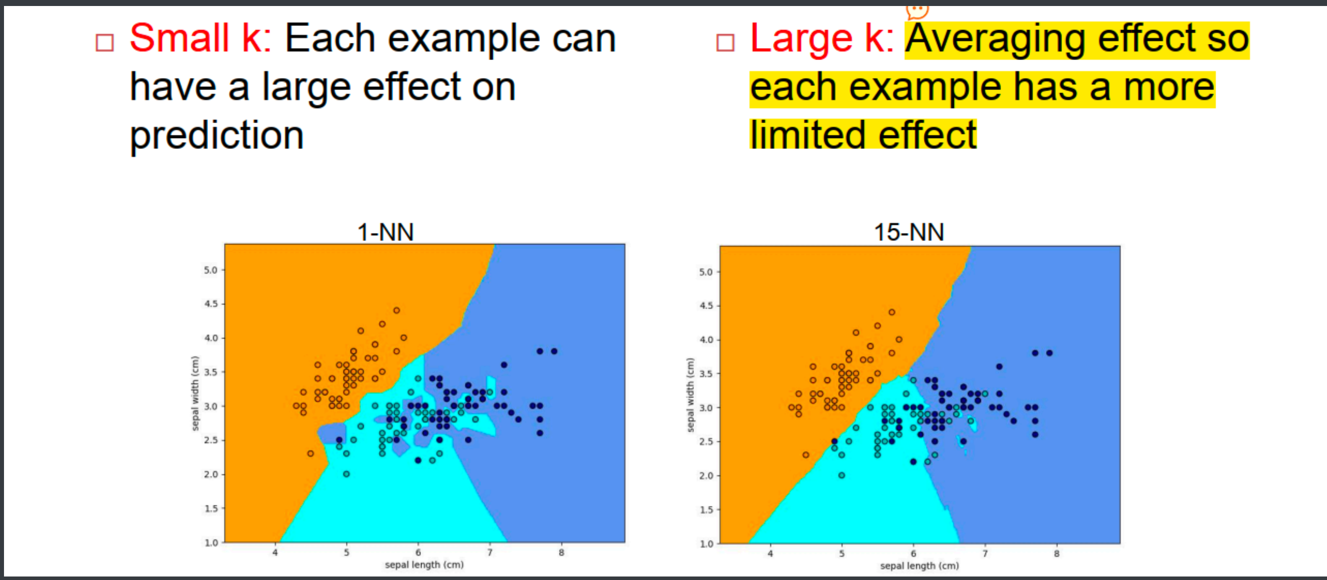
- hypothesis space: set of legal hypotheses



decision surface: separates regions that make different predictions



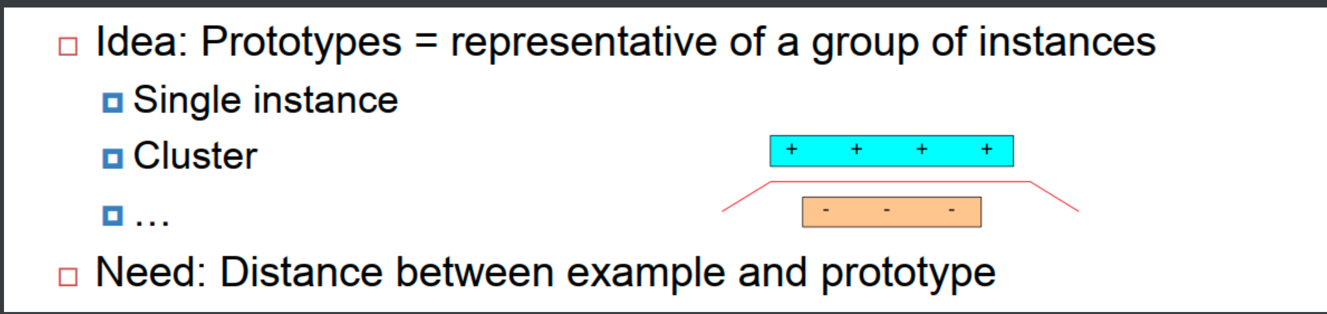
- effect of k



- prototypes

  motivation is **jagged decision surfaces**, which might lead to overfitting

  solution is prototypes: representative of a group of instances



potential pitfalls & how to address

1. different scales of feature values

   e.g. one of the feature is price (like 10000euro), the other is weight (like 10kg)

   solution: normalization

   solution 1: min-max normalization (linear transformation)

   solution 2: standardize (normal distribution)

2. irrelevant features

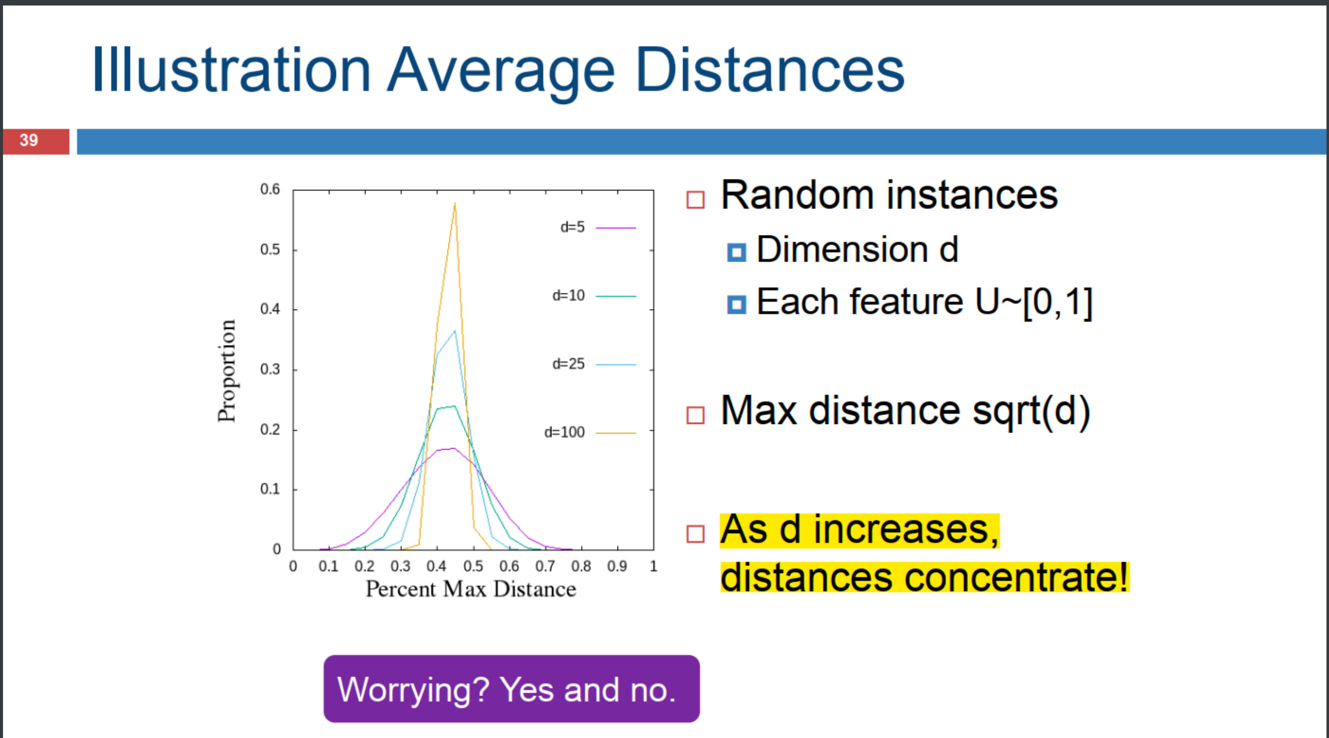
   for example, whether an animal wears clothes

   solution:

   use domain language / weight the features in some way

3. in high-dimension

- in very high dimensions, basically all examples are equi-distant
- nearest neightbors are easily misled with high-dimensional spaces



yes:

"nearest neighbors" doesn't really make any difference

no:

this is happening under **uniform distribution**, but in reality there's always some correlation among features

4. prediction time

lots of data lead to slow execution

solutions:

solution 1: efficient retrieval structures (e.g. KD tree)

solution 2: locality sensitive hashing (but does not guarantee correct set)

solution 3: product quantization (compressed data & approximated distance)