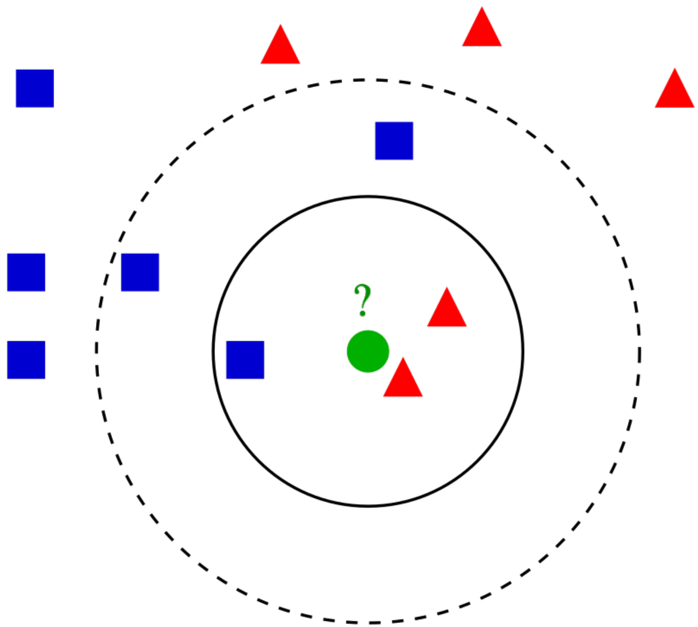
III. Nearest Neighbor Learning



"If something is similar in some respects,
it's likely to be similar in other respects."

-Patrick Winston

Image Source

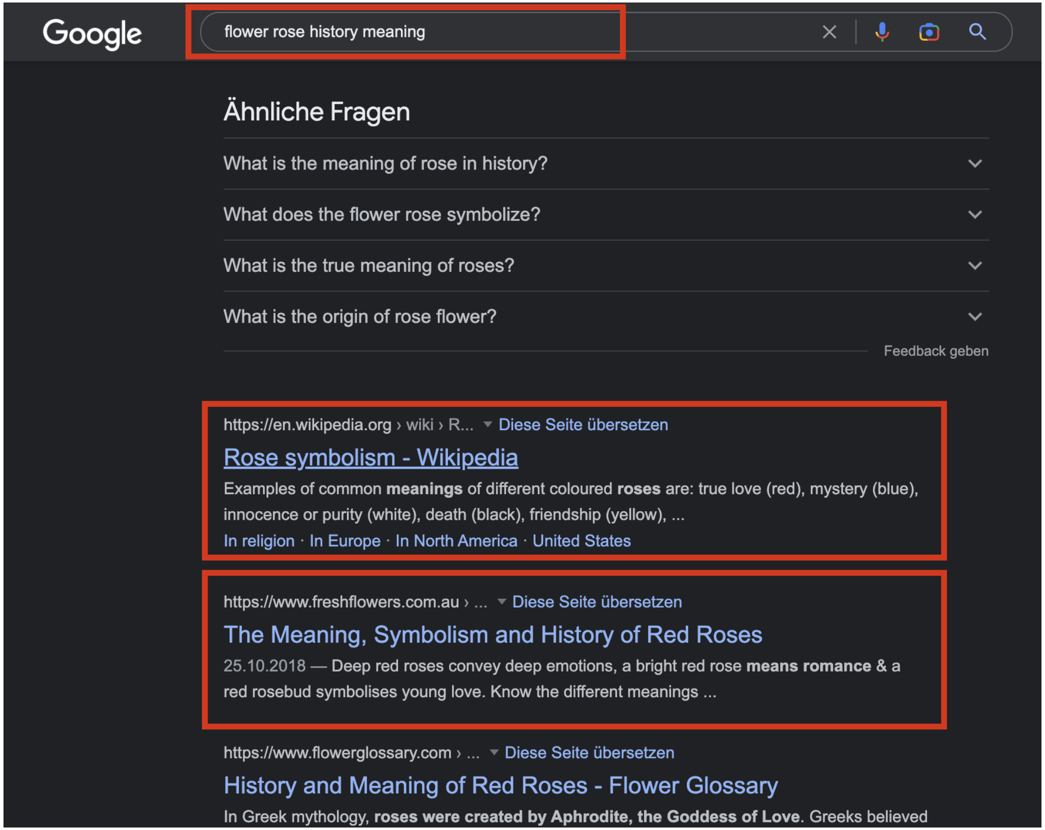
Introduction

Motivation



Introduction

Motivation



Nearest Neighbor Learning

Principle

General properties:

- *Lazy* learning
  The majority of computation is deferred to the prediction phase
- Instance-based prediction
  There is no explicit model the prediction is based on; instead, prediction is based directly on the instances in the training data set
- Non-parametric
  No assumption about analytical form of $f(\mathbf{x})$.
- From a Bayesian perspective: Discriminative model
  Models posterior $p(f(\mathbf{x}) = c|\mathbf{x})$ directly.

Common Use-cases:

- Recommender Systems: Collaborative Filtering
- Outlier Detection: Nearest Neighbor far away?

(1-)Nearest Neighbor Algorithm

Pseudocode

Algorithm:	*NN*	Nearest Neighbor.
Input:	D	Training examples of the form $(\mathbf{x}, y)$ with target value y.
	d	Distance measure in feature space; $d : X \times X \rightarrow \mathbb{R}^+$.
	$\mathbf{q}$	Query data point.
Output:	$h(\mathbf{q})$	Prediction for $\mathbf{q}$.

NN – *Training*(D)

FOR i **IN** $1, \ldots, n$ **DO** // $|D| = n$
    store_training_example$((\mathbf{x}, y)_i)$

NN – *Prediction*$(D, d, \mathbf{q})$

closest_index $= \emptyset$, closest_distance $= \infty$
FOR i **IN** $1, \ldots, n$ **DO**
    distance $= d(\mathbf{q}, \mathbf{x}_i)$
    IF distance $<$ closest_distance **DO**
        closest_distance $=$ distance
        closest_index $= i$
RETURN y_i

Nearest Neighbor Learning

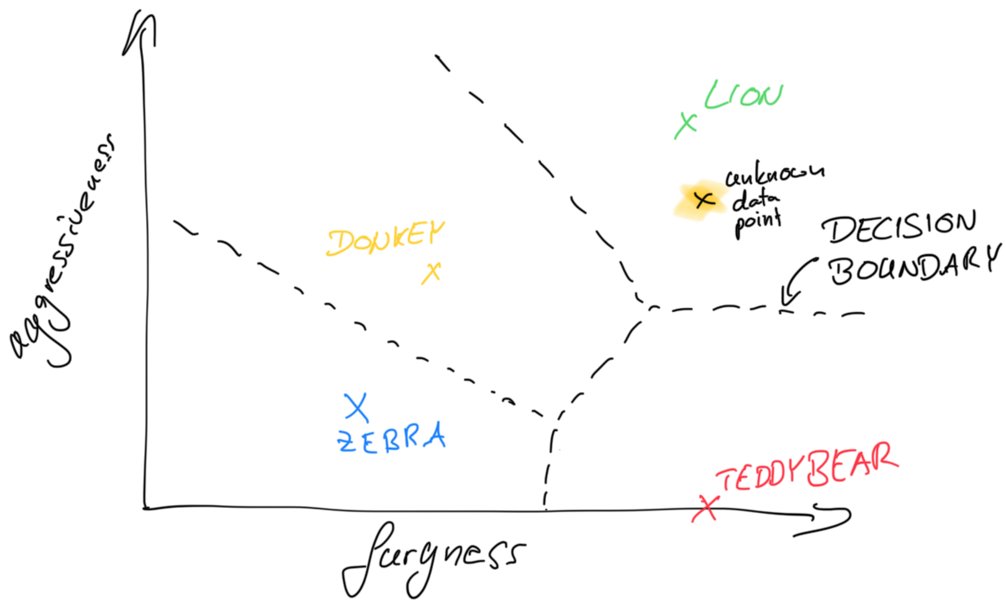
Illustration of Decision Boundary (1)

- Assuming Euclidean distance: $d(\mathbf{a}, \mathbf{b}) = \sqrt{\sum_{k=1}^{d} (a_k - b_k)^2}$
- Decision boundary between any two training examples is a straight line.

  A point on the line is *equidistant* from both training examples.
- Decision boundary globally is a set of connected convex polyhedra.

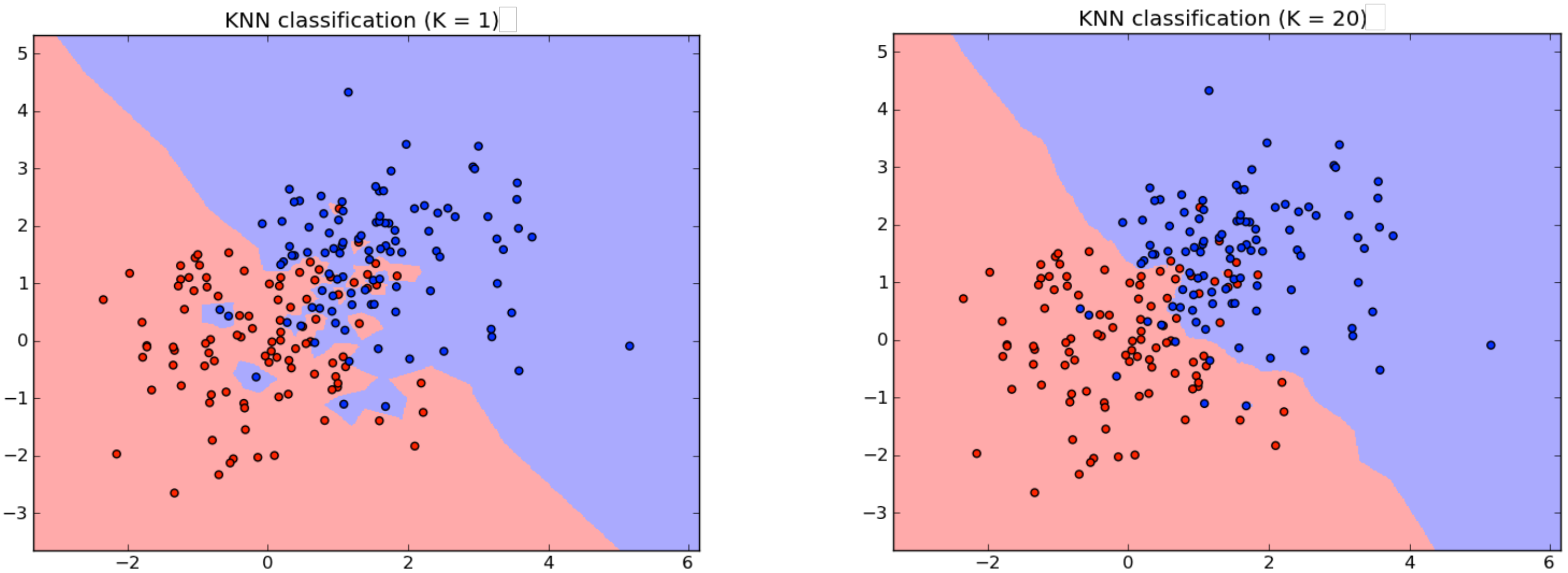
  In 2-d also called: "Voronoi Tessellation"

  Obtained by removing line segements that separate examples from the same class.



Nearest Neighbor Learning

Illustration of decision boundary (2)

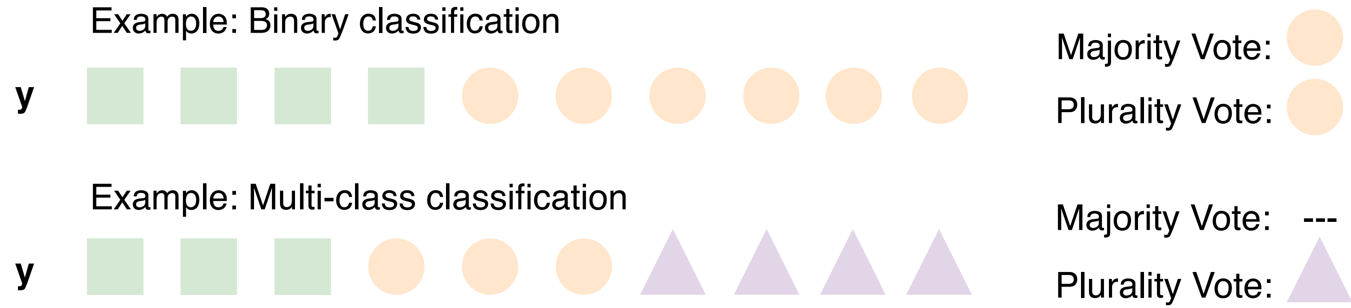


k-Nearest Neighbor Learning

Classification

The k-NNs prediction is based on the k most similar ($\approx$ nearest) training points.
The predicted class label is obtained via plurality voting among k neighbors.

Oftentimes called "majority" voting. But no absolute "majority" required in multi-class settings.

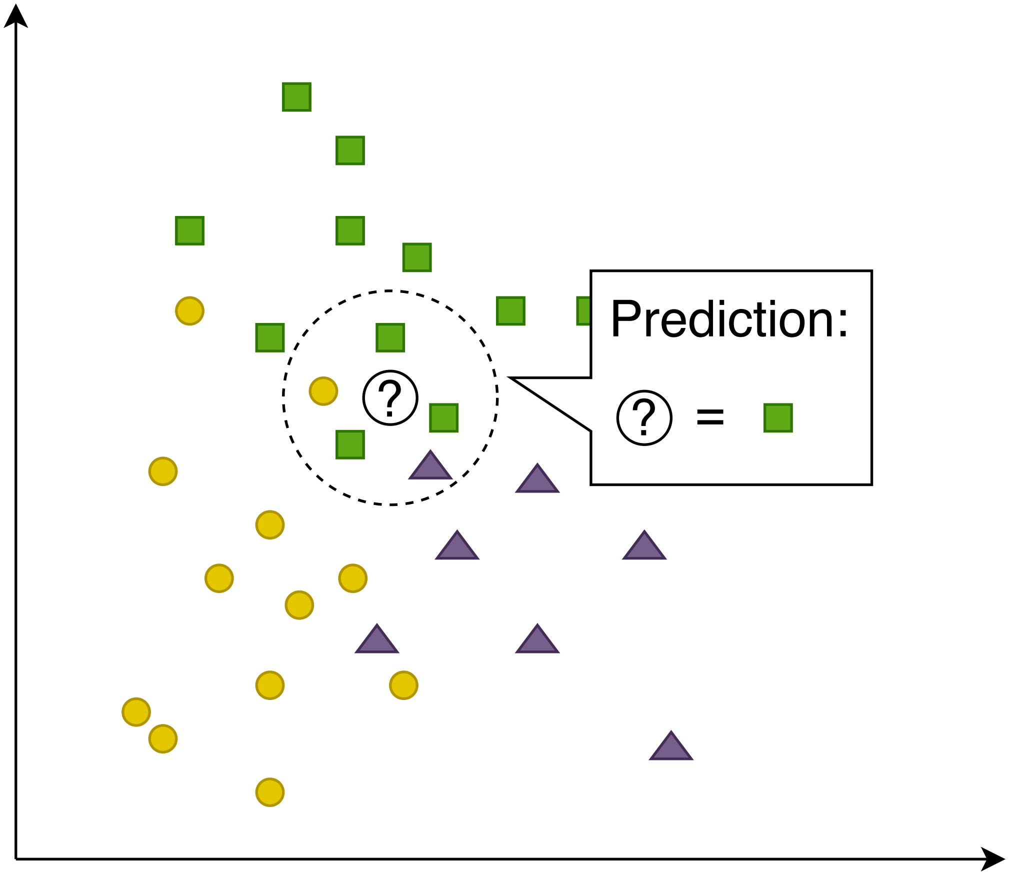


Given a target function $f : \mathbb{R}^d \rightarrow \{1, \dots, C\}$, that assigns a class label $y \in \{1, \dots, C\}$ to every data point $\mathbf{x}$: $f(\mathbf{x}) = y$, the set of k nearest neighbors is denoted by $D_{\mathbf{q}} = \{(\mathbf{x}_i, y_i)\}_{i=1,\dots,k}$.
For a given query point $\mathbf{q}$, the plurality vote of a k-NN classifier is

$$h(\mathbf{q}) = \underset{c \in \{1,\dots,C\}}{\operatorname{argmin}} \sum_{i=1}^{k} \mathbb{1}[c = f(\mathbf{x}_i)]$$

k-Nearest Neighbor Learning

Classification: Plurality voting



5-Nearest Neighbor plurality voting in a 3-class problem.

k-Nearest Neighbor Learning

Regression

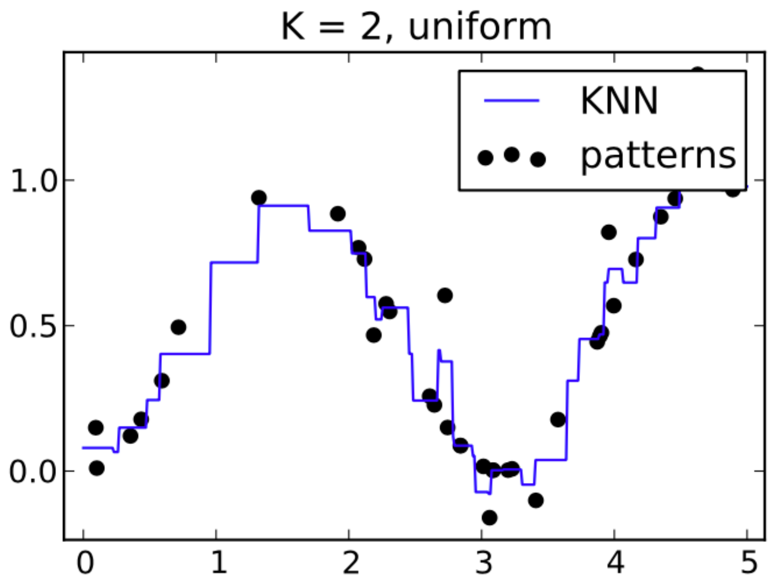
In a regression setting, the target function $f : \mathbb{R}^d \rightarrow \mathbb{R}$ maps to real values. A common approach for combining the target values of the k neighbors is to compute the mean

$$h(\mathbf{q}) = \frac{1}{k} \sum_{i=1}^{k} f(\mathbf{x}_i)$$

Alternative aggregation functions:

- Median: Less sensitive to outliers
- Weighted: Target values weighted by a function of the distance.
  E.g. the weight of neighbor i is inversely proportional to its distance to $\mathbf{q}$:

$$w_i = \frac{1}{d(\mathbf{q}, \mathbf{x}_i)^2}$$



k-Nearest Neighbour (kNN) classifier

Pseudocode

Algorithm:	*KNN*	k-Nearest Neighbor.
Input:	D	Training examples of the form $(\mathbf{x}, f(\mathbf{x}))$ with class value y_i for $\mathbf{x}_i$.
	k	Number of nearest neighbors to consider.
	d	A distance measure in feature space; $d : X \times X \to \mathbb{R}^+$.
	$\mathbf{q}$	A query data point to classify.
Output:	$h(\mathbf{q})$	Predicted class of $\mathbf{q}$.

KNN$(D, k, d, \mathbf{q})$

$U = \emptyset$, $\mathcal{N}_k = \emptyset$
FOR $(\mathbf{x}_i, y_i)$ **IN** D **DO**
  $U = U \cup \{(d(\mathbf{q}, \mathbf{x}_i), y_i)\}$
ENDDO
$U =$ *sort_distances_ascending*(U)
$\mathcal{N}_k =$ *select_top_k*(U, k)
$h(\mathbf{q}) =$ *find_most_frequent_class*$(\mathcal{N}_k)$
RETURN $h(\mathbf{q})$

k-Nearest Neighbour (kNN) classifier

Pseudocode

Algorithm:	*KNN*	k-Nearest Neighbor.
Input:	D	Training examples of the form $(\mathbf{x}, f(\mathbf{x}))$ with class value y_i for $\mathbf{x}_i$.
	k	Number of nearest neighbors to consider.
	d	A distance measure in feature space; $d : X \times X \to \mathbb{R}^+$.
	$\mathbf{q}$	A query data point to classify.
Output:	$h(\mathbf{q})$	Predicted class of $\mathbf{q}$.

KNN$(D, k, d, \mathbf{q})$

$U = \emptyset$, $\mathcal{N}_k = \emptyset$
FOR $(\mathbf{x}_i, y_i)$ **IN** D **DO**
  $U = U \cup \{(d(\mathbf{q}, \mathbf{x}_i), y_i)\}$
ENDDO
$U =$ *sort_distances_ascending*(U)
$\mathcal{N}_k =$ *select_top_k*(U, k)
$h(\mathbf{q}) =$ *find_most_frequent_class*$(\mathcal{N}_k)$
RETURN $h(\mathbf{q})$

Issues/Bottlenecks?

Improving Computational Performance

Runtime Complexity

The Landau-notation ("Big-$\mathcal{O}$") provides a formalism to express an upper bound (worst case) on the asymptotic growth rate of a function.

- In computer science, the Big-$\mathcal{O}$ notation is used to quantify runtime (time complexity) or memory usage (space complexity) of an algorithm.
- In a naive implementation, the k-NN prediction phase has to execute $n \cdot d$ comparisons of constant cost. Since typically $n >> d$, the complexity of the nearest neighbor algorithm is $\mathcal{O}(n)$.
- *...for every single query point!*

Strategies for faster prediction:

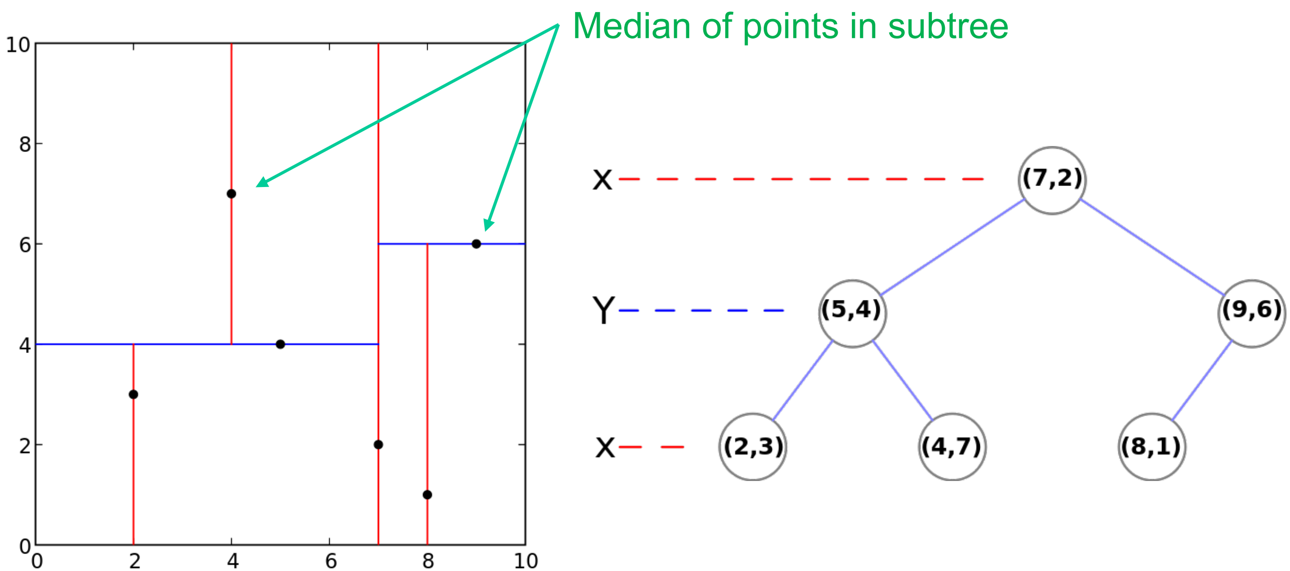
- Simple tweak to avoid sorting: Priority Queue
- Store training examples in range based data structures such as e.g. KD-Trees or Ball-Trees
- Faster distance metric or heuristic
- Pruning
- Parallelization

Improving Computational Performance

Optimized Storage of Training Examples

Use a k-d tree to index regions in feature space:

- Multi-dimensional Binary Search Tree (BST)
- Maintain leaf counts on nodes



Improving Computational Performance

Pruning

Editing:

- Permanently remove training examples that do not affect decision boundary.

- Single examples surrounded by examples from a different class don't affect decision boundary in plurality voting.

Prototypes:

- Replace selected examples by "prototypes" that summarize multiple examples in dense regions.
- E.g. k-Means clustering to obtain "prototypes"

Improving Predictive Performance

Model Selection

There are a couple of ways to tune the predictive performance of the k-NN algorithm:

- Choosing the value of k
- Scaling feature axes
- Choice of distance measure
- Weighting scheme for distance measure

$\rightarrow$ Model selection (aka hyper-parameter tuning) using cross-validation

Distance Measures

Consider two points $\mathbf{u}, \mathbf{v}$ in feature space $X \subset \mathbb{R}^l$:

- Manhattan distance (L_1-Norm)

$$d(\mathbf{u}, \mathbf{v}) = \sum_{i=1}^{l} |u_i - v_i| \tag{1}$$

- Euclidean distance (L_2-Norm)

$$d(\mathbf{u}, \mathbf{v}) = \sqrt{\sum_{i=1}^{l} (u_i - v_i)^2} \tag{2}$$

- Maximum distance (L_∞-Norm)

$$d(\mathbf{u}, \mathbf{v}) = \text{max}\{|u_i - v_i|\}_{i=1}^{l} \tag{3}$$

- Minkowski distance (L_p-Norm): A generalized version of L-Norms.

$$d(\mathbf{u}, \mathbf{v}) = \sqrt[p]{\sum_{i=1}^{l} |u_i - v_i|^p} \tag{4}$$

Distance Measures

- Cosine similarity[1]

$$cos(\theta) = \frac{\mathbf{a}^T\mathbf{b}}{\|\mathbf{a}\| \cdot \|\mathbf{b}\|} \tag{5}$$

- Jaccard Index
- Hamming distance
- Mahalanobis distance

[1] Not a proper distance metric; Violates triangle inequality.

k-Nearest Neighbor Learning

Probabilistic interpretation

- The k-Nearest Neighbor algorithm estimates the conditional probabilities $p(\mathrm{c}|\mathbf{x})$ by counting the data points in class c in the neighbourhood $\mathcal{N}_k(\mathbf{x})$ of $\mathbf{x}$:

$$p(\mathrm{c}|\mathbf{x}) = \frac{|\{\mathbf{x}' \in \mathcal{N}_k(\mathbf{x}) | f(\mathbf{x}') = \mathrm{c}\}|}{k} \tag{6}$$

- Bias-variance trade-off
  - Small k leads to large variance in test errors and unreliable classification
  - Large k leads to large bias and inaccurate classification

Challenges

- Spread problem: Input variables have different variances.
  $\rightarrow$ Normalize values on each dimension to unit variance $x' = \frac{x}{\sigma_x^2}$
- Irrelevant dimension problem: The output is independent of one of the input variables.
  Confusing results, since the irrelevant variable contributes to the distance measure in the same way as any relevant variable.
  $\rightarrow$ Exclude irrelevant variable
- Irrelevant data problem: The output does not depend on the input variables at all.
  Can we predict whether a person is going to go bankcrupt based on their physical appearance?
  $\rightarrow$ "No cake without flour".

Summary

- kNN is a lazy, instance-based, non-parametric supervised learning algorithm
  Non-parametric: No assumptions about the analytical form of $f(x) = y$.
- Able to represent highly non-linear decision boundaries
- Efficient implementation requires special data structures
- Distance metric needs to be chosen carefully (validated)
- Simple and (surprisingly) great for small training data sets

$\rightarrow$ kNN in *scikit-learn*: https://scikit-learn.org/stable/modules/neighbors.html