Applied Machine Learning

Some basic concepts

Siamak Ravanbakhsh

COMP 551 (winter 2020)

Objectives

- learning as representation, evaluation and optimization
- k-nearest neighbors for classification
- curse of dimensionality
- manifold hypothesis
- overfitting & generalization
- cross validation
- no free lunch theorem
- inductive bias

A useful perspective on ML

Let's focus on classification

Learning = Representation + Evaluation + Optimization

from: Domingos, Pedro M. "A few useful things to know about machine learning." *Commun. acm* 55.10 (2012): 78-87.

A useful perspective on ML

Let's focus on classification

Learning = Representation + Evaluation + Optimization

Model

Hypothesis space

the space of functions to choose from is determined by how we represent/define the learner

from: Domingos, Pedro M. "A few useful things to know about machine learning." *Commun. acm* 55.10 (2012): 78-87.

A useful perspective on ML

Let's focus on classification

Learning = Representation + Evaluation + Optimization

Representation	Evaluation	Optimization
Model	Objective function	
Hypothesis space	Cost function	
	Score function	

the criteria for picking the best model

the space of functions to choose from is determined by how we represent/define the learner

from: Domingos, Pedro M. "A few useful things to know about machine learning." *Commun. acm* 55.10 (2012): 78-87.

A useful perspective on ML

Let's focus on classification

Learning = Representation + Evaluation + Optimization

Representation	Evaluation	Optimization
Model	Objective function	Objective
Hypothesis space	Cost function	Cost
	Score function	Loss

procedure for finding the best model

the criteria for picking the best model

the space of functions to choose from is determined by how we represent/define the learner

from: Domingos, Pedro M. "A few useful things to know about machine learning." *Commun. acm* 55.10 (2012): 78-87.

A useful perspective on ML

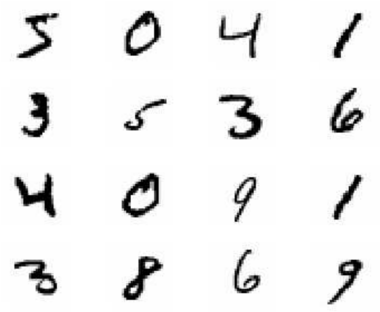
Let's focus on classification

Learning =

Representation	Evaluation	Optimization
Instances	Accuracy/Error rate	Combinatorial optimization
K-nearest neighbor	Precision and recall	Greedy search
Support vector machines	Squared error	Beam search
Hyperplanes	Likelihood	Branch-and-bound
Naive Bayes	Posterior probability	Continuous optimization
Logistic regression	Information gain	Unconstrained
Decision trees	K-L divergence	Gradient descent
Sets of rules	Cost/Utility	Conjugate gradient
Propositional rules	Margin	Quasi-Newton methods
Logic programs		Constrained
Neural networks		Linear programming
Graphical models		Quadratic programming
Bayesian networks		
Conditional random fields		

from: Domingos, Pedro M. "A few useful things to know about machine learning." *Commun. acm* 55.10 (2012): 78-87.

Digits dataset



input $x^{(n)} \in \{0, \ldots, 255\}^{28\times 28}$

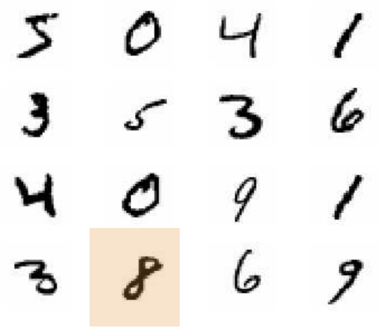
label $y^{(n)} \in \{0, \ldots, 9\}$

$n \in \{1, \ldots, N\}$ indexes the training instance

sometime we drop (n)

image:https://medium.com/@rajatjain0807/machine-learning-6ecde3bfd2f4

Digits dataset



0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	12	0	11	39	137	37	0	152	147	84	0	0	0
0	0	1	0	0	0	41	160	250	255	235	162	255	238	206	11	13	0
0	0	0	16	9	9	150	251	45	21	184	159	154	255	233	40	0	0
10	0	0	0	0	0	145	146	3	10	0	11	124	253	255	107	0	0
0	0	3	0	4	15	236	216	0	0	38	109	247	240	169	0	11	0
1	0	2	0	0	0	253	253	23	62	224	241	255	164	0	5	0	0
6	0	0	4	0	3	252	250	228	255	255	234	112	28	0	2	17	0
0	2	1	4	0	21	255	253	251	255	172	31	8	0	1	0	0	0
0	0	4	0	163	225	251	255	229	120	0	0	0	0	0	11	0	0
0	0	21	162	255	255	254	255	126	6	0	10	14	6	0	0	9	0
3	79	242	255	141	66	255	245	189	7	8	0	0	5	0	0	0	0
26	221	237	98	0	67	251	255	144	0	8	0	0	7	0	0	11	0
125	255	141	0	87	244	255	208	3	0	0	13	0	1	0	1	0	0
145	248	228	116	235	255	141	34	0	11	0	1	0	0	0	1	3	0
85	237	253	246	255	210	21	1	0	1	0	0	6	2	4	0	0	0
6	23	112	157	114	32	0	0	0	0	2	0	8	0	7	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

input $x^{(n)} \in \{0, \ldots, 255\}^{28 \times 28}$ size of the input image in pixels

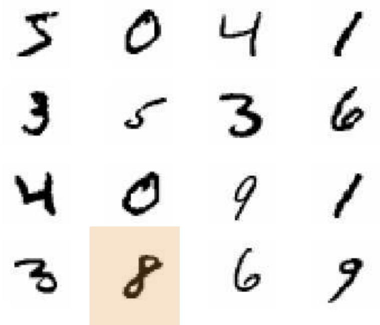
label $y^{(n)} \in \{0, \ldots, 9\}$

$n \in \{1, \ldots, N\}$ indexes the training instance

sometime we drop (n)

image:https://medium.com/@rajatjain0807/machine-learning-6ecde3bfd2f4

Digits dataset



0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	12	0	11	39	137	37	0	152	147	84	0	0	0
0	0	1	0	0	0	41	160	250	255	235	162	255	238	206	11	13	0
0	0	0	16	9	9	150	251	45	21	184	159	154	255	233	40	0	0
10	0	0	0	0	0	145	146	3	10	0	11	124	253	255	107	0	0
0	0	3	0	4	15	236	216	0	0	38	109	247	240	169	0	11	0
1	0	2	0	0	0	253	253	23	62	224	241	255	164	0	5	0	0
6	0	0	4	0	3	252	250	228	255	255	234	112	28	0	2	17	0
0	2	1	4	0	21	255	253	251	255	172	31	8	0	1	0	0	0
0	0	4	0	163	225	251	255	229	120	0	0	0	0	0	11	0	0
0	0	21	162	255	255	254	255	126	6	0	10	14	6	0	0	9	0
3	79	242	255	141	66	255	245	189	7	8	0	0	5	0	0	0	0
26	221	237	98	0	67	251	255	144	0	8	0	0	7	0	0	11	0
125	255	141	0	87	244	255	208	3	0	0	13	0	1	0	1	0	0
145	248	228	116	235	255	141	34	0	11	0	1	0	0	0	1	3	0
85	237	253	246	255	210	21	1	0	1	0	0	6	2	4	0	0	0
6	23	112	157	114	32	0	0	0	0	2	0	8	0	7	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

input $x^{(n)} \in \{0, \dots, 255\}^{28 \times 28}$ size of the input image in pixels

label $y^{(n)} \in \{0, \dots, 9\}$

$n \in \{1, \dots, N\}$ indexes the training instance
sometime we drop (n)

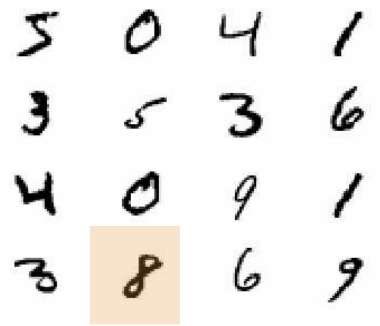
vectorization:

$x \rightarrow \mathrm{vec}(x) \in \mathbb{R}^{784}$ input dimension **D**

pretending intensities are real numbers

image:https://medium.com/@rajatjain0807/machine-learning-6ecde3bfd2f4

Digits dataset



0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	12	0	11	39	137	37	0	152	147	84	0	0	0
0	0	1	0	0	0	41	160	250	255	235	162	255	238	206	11	13	0
0	0	0	16	9	9	150	251	45	21	184	159	154	255	233	40	0	0
10	0	0	0	0	0	145	146	3	10	0	11	124	253	255	107	0	0
0	0	3	0	4	15	236	216	0	0	38	109	247	240	169	0	11	0
1	0	2	0	0	0	253	253	23	62	224	241	255	164	0	5	0	0
6	0	0	4	0	3	252	250	228	255	255	234	112	28	0	2	17	0
0	2	1	4	0	21	255	253	251	255	172	31	8	0	1	0	0	0
0	0	4	0	163	225	251	255	229	120	0	0	0	0	0	11	0	0
0	0	21	162	255	255	254	255	126	6	0	10	14	6	0	0	9	0
3	79	242	255	141	66	255	245	189	7	8	0	0	5	0	0	0	0
26	221	237	98	0	67	251	255	144	0	8	0	0	7	0	0	11	0
125	255	141	0	87	244	255	208	3	0	0	13	0	1	0	1	0	0
145	248	228	116	235	255	141	34	0	11	0	1	0	0	0	1	3	0
85	237	253	246	255	210	21	1	0	1	0	0	6	2	4	0	0	0
6	23	112	157	114	32	0	0	0	0	2	0	8	0	7	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

input $x^{(n)} \in \{0,\ldots,255\}^{28\times 28}$ size of the input image in pixels

label $y^{(n)} \in \{0,\ldots,9\}$

$n \in \{1,\ldots,N\}$ indexes the training instance

sometime we drop (n)

vectorization:

$$x \to \mathrm{vec}(x) \in \mathbb{R}^{784}$$

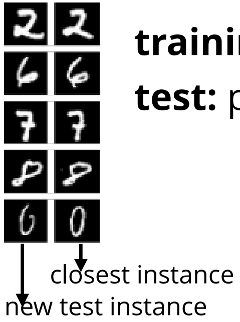
input dimension **D**

pretending intensities are real numbers

note: this ignores the spatial arrangement of pixels, but good enough for now

image:https://medium.com/@rajatjain0807/machine-learning-6ecde3bfd2f4

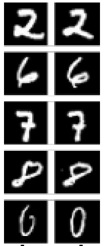
Nearest neighbour classifier



training: do nothing

test: predict the lable by finding the closest image in the training set and

Nearest neighbour classifier

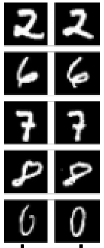


training: do nothing

test: predict the lable by finding the closest image in the training set and

need a measure of distance

Nearest neighbour classifier



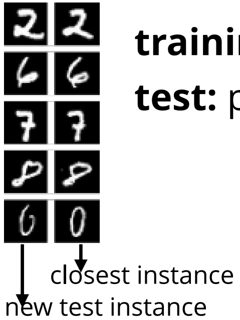
training: do nothing

test: predict the lable by finding the closest image in the training set and

need a measure of distance

e.g., Euclidean distance $||x - x'||_2 = \sqrt{\sum_{d=1}^{D}(x_d - x'_d)^2}$

Nearest neighbour classifier

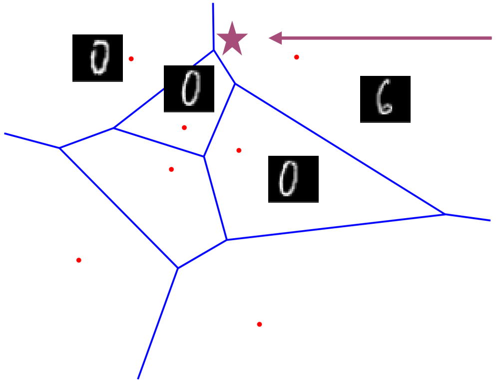


training: do nothing

test: predict the lable by finding the closest image in the training set and

need a measure of distance

e.g., Euclidean distance $||x - x'||_2 = \sqrt{\sum_{d=1}^{D}(x_d - x'_d)^2}$



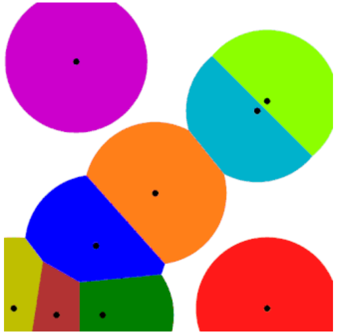
test instance: will be classified as 6

Voronoi diagram shows the decision boundaries

(this example D=2, can't visualize D=784)

the Voronoi Diagram

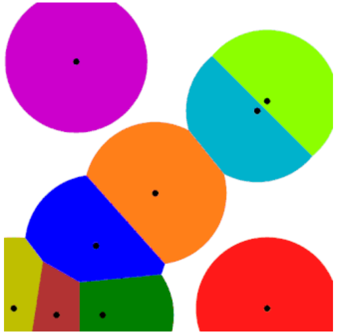
each colour shows all points closer to the corresponding training instance than to any other instance

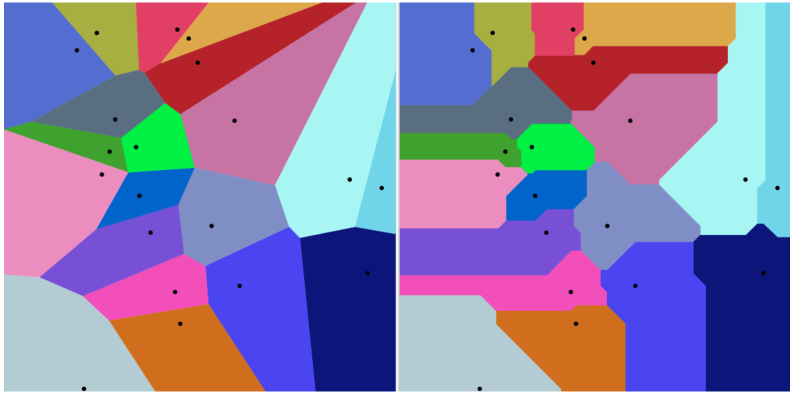


images from wiki

the Voronoi Diagram

each colour shows all points closer to the corresponding training instance than to any other instance





Euclidean distance

$$||x - x'||_2 = \sqrt{\sum_{d=1}^{D}(x_d - x'_d)^2}$$

Manhattan distance

$$||x - x'||_1 = \sum_{d=1}^{D}|x_d - x'_d|$$

images from wiki

K- nearest neighbours

training: do nothing

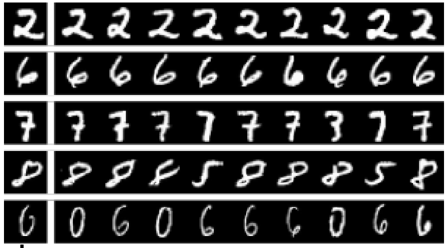
test: predict the lable by finding the **K** closest instances

$$p(y^{new} = c \mid x_{new}) = \frac{1}{K} \sum_{x' \in \mathrm{KNN}(x^{new})} \mathbb{I}(y' = c)$$

probability of class c

K-nearest neighbours

example K = 9



closest instances

new test instance

$p(y = 6 |$  $) = \frac{6}{9}$

K- nearest neighbours

training: do nothing

test: predict the lable by finding the **K** closest instances

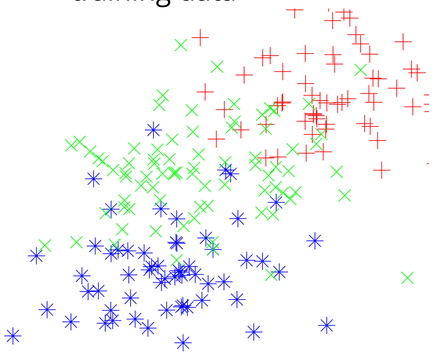
$$p(y^{new} = c \mid x_{new}) = \frac{1}{K} \sum_{x' \in \mathrm{KNN}(x^{new})} \mathbb{I}(y' = c)$$

probability of class c

K-nearest neighbours

example C=3, D=2, K=10

training data



K- nearest neighbours

training: do nothing

test: predict the lable by finding the **K** closest instances

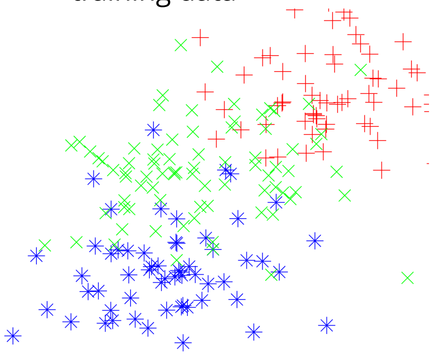
$$p(y^{new} = c \mid x_{new}) = \frac{1}{K} \sum_{x' \in \mathrm{KNN}(x^{new})} \mathbb{I}(y' = c)$$

probability of class c

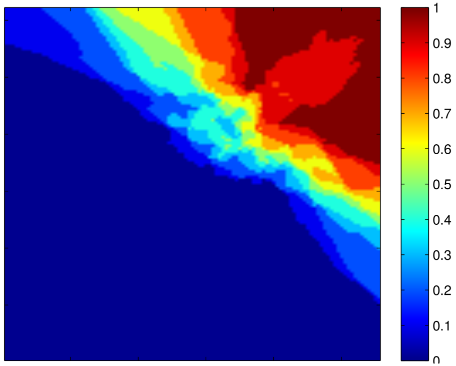
K-nearest neighbours

example C=3, D=2, K=10

training data



prob. of class 1



K- nearest neighbours

training: do nothing

test: predict the lable by finding the **K** closest instances

$$p(y^{new} = c \mid x_{new}) = \frac{1}{K} \sum_{x' \in \mathrm{KNN}(x^{new})} \mathbb{I}(y' = c)$$

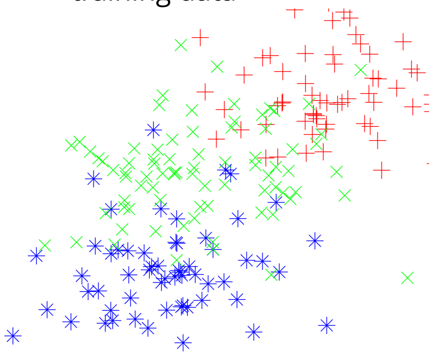
probability of class c

K-nearest neighbours

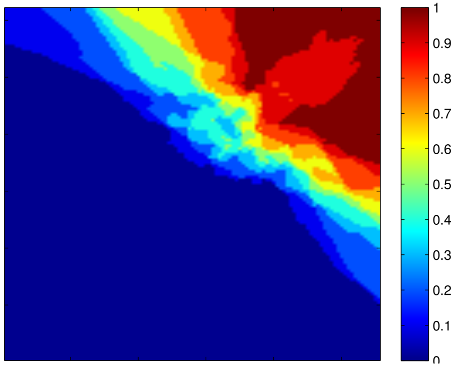
example

C=3, D=2, K=10

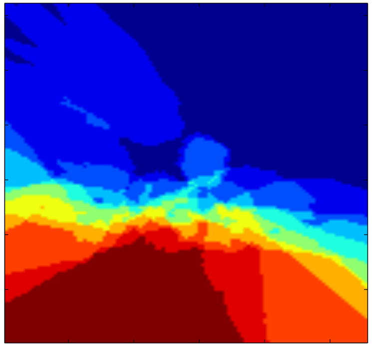
training data



prob. of class 1



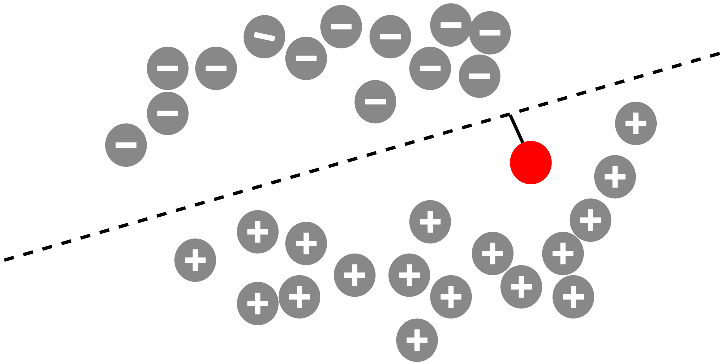
prob. of class 2

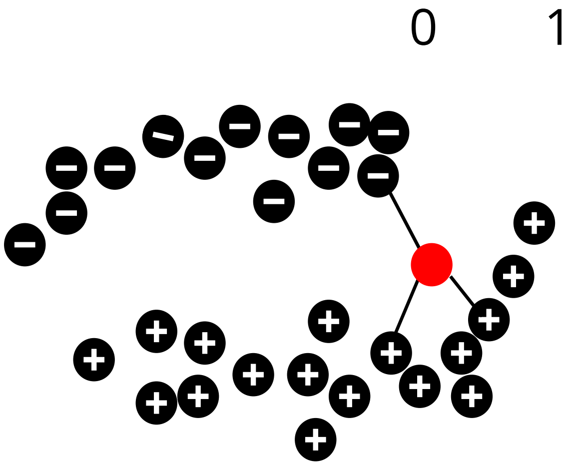


K- nearest neighbours

a **non-parametric method (misnomer)**: the number of model parameters grows with the data

a **lazy-learner**: no training phase, locally estimate when a query comes
useful for fast-changing datasets





Curse of dimensionality

high dimensions are unintuitive!

assuming a uniform distribution $x \in [0,1]^D$

Curse of dimensionality

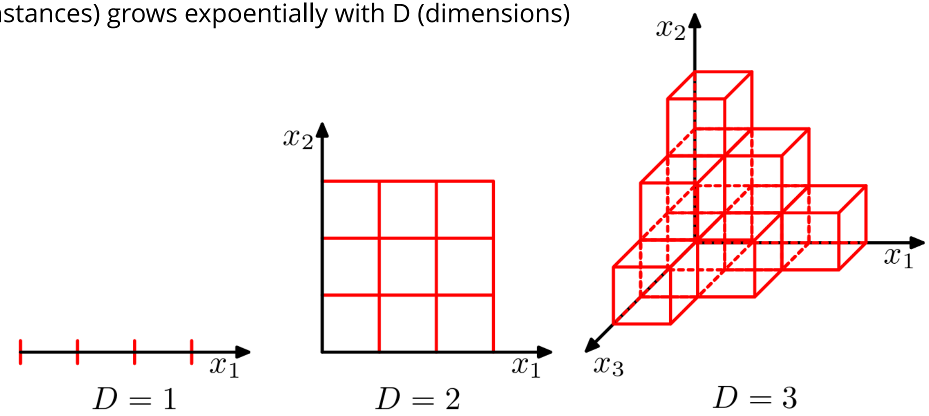
high dimensions are unintuitive!

assuming a uniform distribution $x \in [0,1]^D$

- need exponentially more instances for K-NN

suppose we want to maintain #samples per sub-cube of side 1/3

N (total #training instances) grows expoentially with D (dimensions)



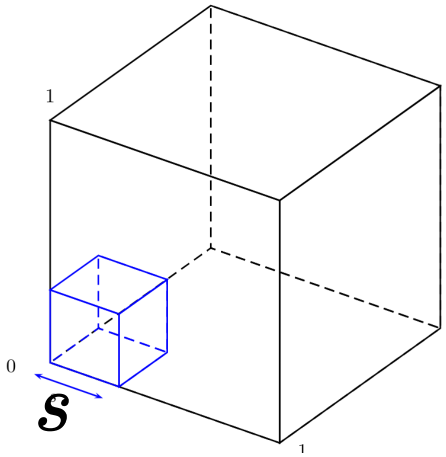
Curse of dimensionality

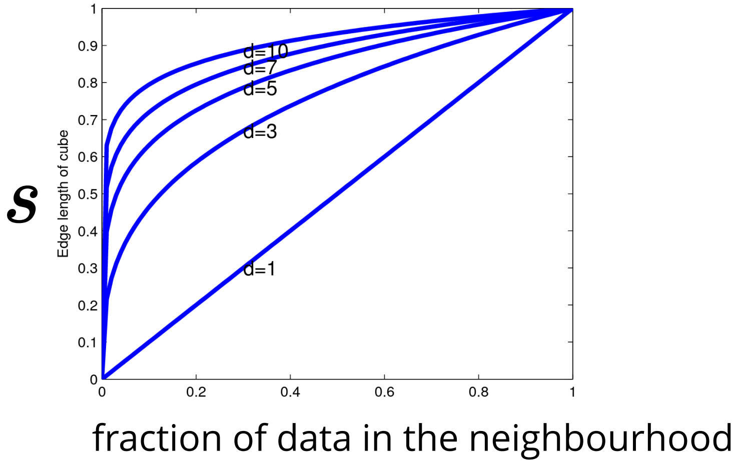
high dimensions are unintuitive!

assuming a uniform distribution $x \in [0,1]^D$

- need exponentially more instances for K-NN

Another way to see this





Curse of dimensionality

high dimensions are unintuitive!

assuming a uniform distribution $x \in [0,1]^D$

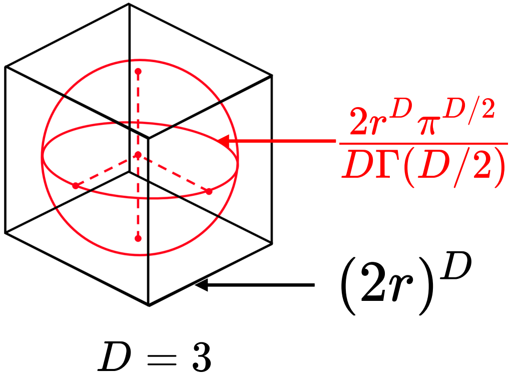
- need exponentially more instances for K-NN
- all instances have similar distances

Curse of dimensionality

high dimensions are unintuitive!

assuming a uniform distribution $x \in [0,1]^D$

- need exponentially more instances for K-NN
- all instances have similar distances

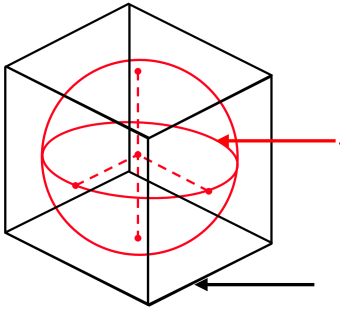


Curse of dimensionality

high dimensions are unintuitive!

assuming a uniform distribution $x \in [0,1]^D$

- need exponentially more instances for K-NN
- all instances have similar distances



$$\frac{2r^D \pi^{D/2}}{D\Gamma(D/2)}$$

$$(2r)^D$$

$$D = 3$$

$$\lim_{D \to \infty} \frac{\mathrm{volum}(\bigcirc)}{\mathrm{volum}(\square)} = 0$$

most of the volume is close to the corners

most pairwise disstances are similar

Curse of dimensionality

high dimensions are unintuitive!

assuming a uniform distribution $x \in [0,1]^D$

- need exponentially more instances for K-NN
- all instances have similar distances

a "conceptual" visualization of the same example

- # corners and the mass in the corners grows quickly

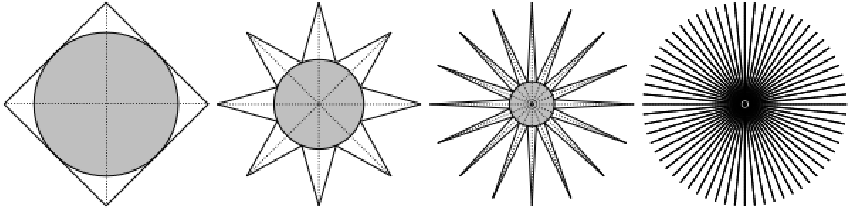


image: Zaki's book on Data Mining and Analysis

Manifold hypothesis

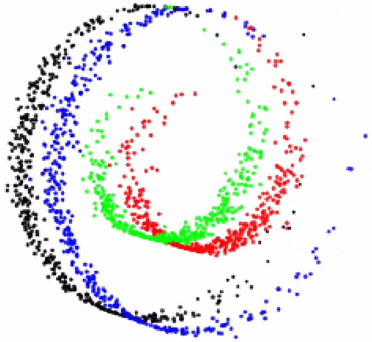
real-world data is often far from uniform

manifold hypothesis: real data lies close to the surface of a manifold

Manifold hypothesis

real-world data is often far from uniform

manifold hypothesis: real data lies close to the surface of a manifold



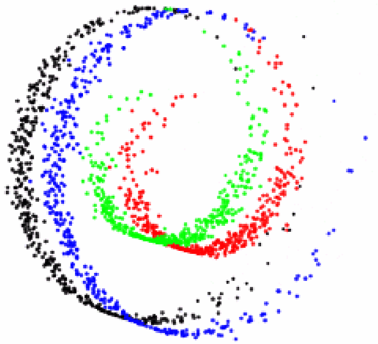
ambient (data) dimension: $D = 3$

manifold dimension: $\hat{D} = 2$

Manifold hypothesis

real-world data is often far from uniform

manifold hypothesis: real data lies close to the surface of a manifold



ambient (data) dimension: $D = 3$

manifold dimension: $\hat{D} = 2$

MNIST digit classification results

for K-NN the manifold dimension matters

so K-NN can be competitive

	Test Error Rate (%)
Linear classifier (1-layer NN)	12.0
K-nearest-neighbors, Euclidean	5.0
K-nearest-neighbors, Euclidean, deskewed	2.4
K-NN, Tangent Distance, 16x16	1.1
K-NN, shape context matching	0.67
1000 RBF + linear classifier	3.6
SVM deg 4 polynomial	1.1
2-layer NN, 300 hidden units	4.7
2-layer NN, 300 HU, [deskewing]	1.6
LeNet-5, [distortions]	0.8
Boosted LeNet-4, [distortions]	0.7

$D = 784$ is the number of pixels

manifold dimension ?

Model selection

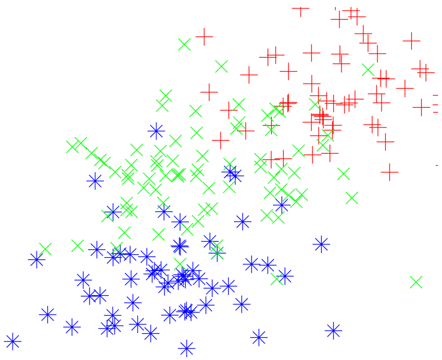
K is a **hyper-parameter**: a model parameter that is not learned by the algorithm

Model selection

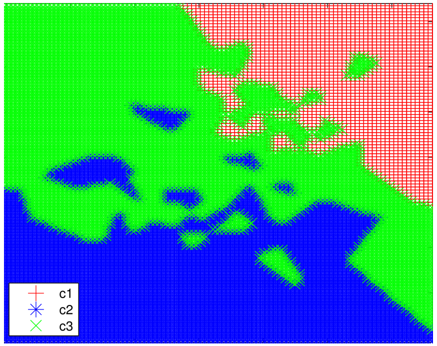
K is a **hyper-parameter**: a model parameter that is not learned by the algorithm

example

training data



K=1

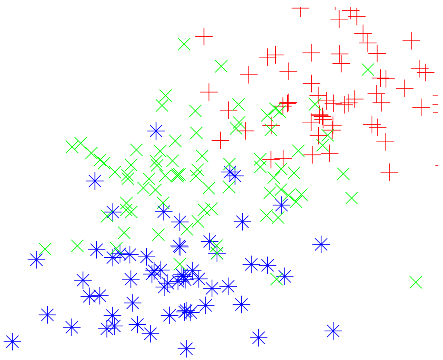


Model selection

K is a **hyper-parameter**: a model parameter that is not learned by the algorithm

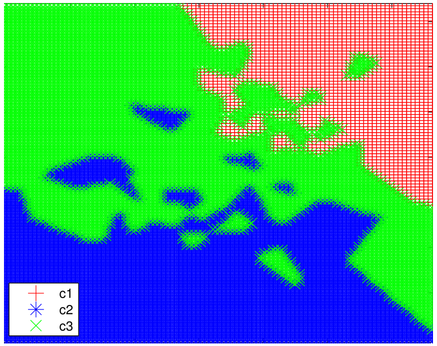
example

training data

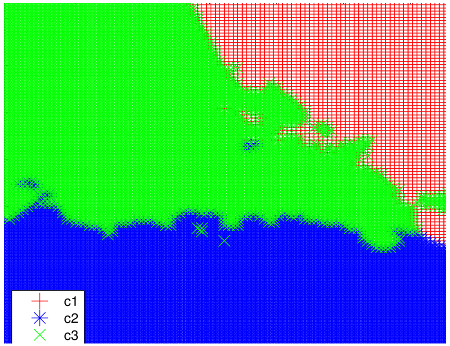


most likely class

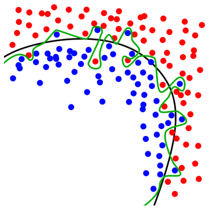
K=1



K=5

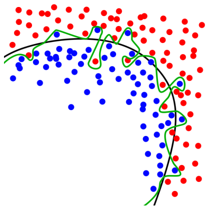


Overfitting



how to pick the best K?

Overfitting

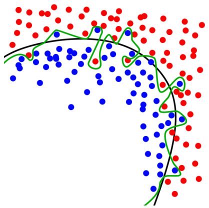


how to pick the best K?

first attempt pick K that gives "best results" on the training set

e.g., misclassification error $\sum_n \mathbb{I}(\arg\max_y p(y \mid x^{(n)}) \neq y^{(n)})$

Overfitting



how to pick the best K?

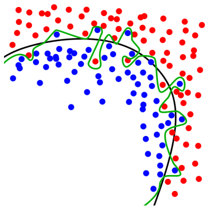
first attempt pick K that gives "best results" on the training set

e.g., misclassification error $\sum_n \mathbb{I}(\arg\max_y p(y \mid x^{(n)}) \neq y^{(n)})$

bad idea!

we can overfit the training data

we can have bad performance on new instances



Overfitting

how to pick the best K?

first attempt pick K that gives "best results" on the training set

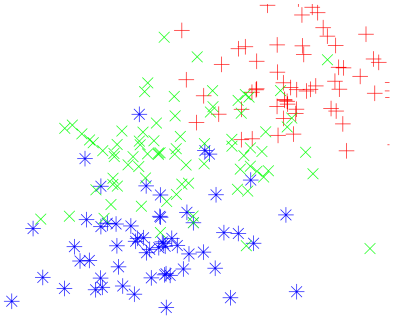
e.g., misclassification error $\sum_n \mathbb{I}(\arg\max_y p(y \mid x^{(n)}) \neq y^{(n)})$

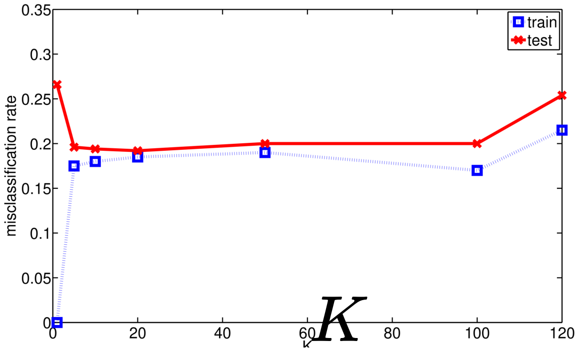
bad idea!

we can overfit the training data

we can have bad performance on new instances

example

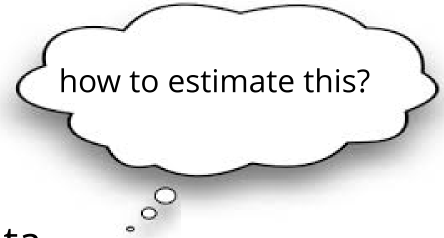




Generalization

what we care about is generalization

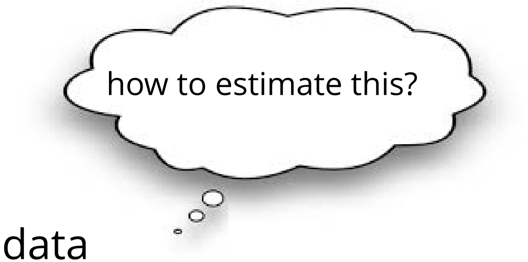
Generalization



what we care about is generalization

expected loss: performance of algorithm on unseen data

Generalization



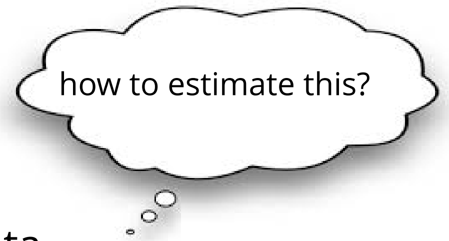
what we care about is generalization

expected loss: performance of algorithm on unseen data

validation set: a subset of available data not used for training

$$\textit{performance on validation set} \approx \textit{expected error}$$

Generalization



what we care about is generalization

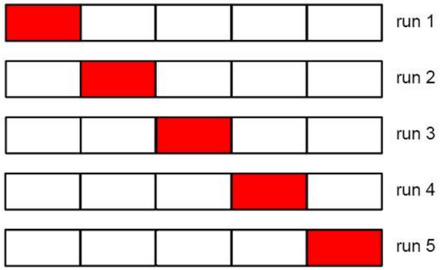
expected loss: performance of algorithm on unseen data

validation set: a subset of available data not used for training

performance on validation set $\approx$ *expected error*

k-fold cross validation(CV)

- partition the data into k *folds*
- use k-1 for training, and 1 for validation
- average the validation error over all folds



Generalization

how to estimate this?

what we care about is generalization

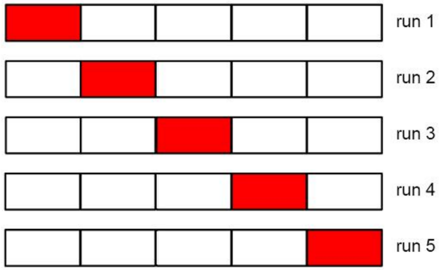
expected loss: performance of algorithm on unseen data

validation set: a subset of available data not used for training

performance on validation set $\approx$ *expected error*

k-fold cross validation(CV)

- partition the data into k *folds*
- use k-1 for training, and 1 for validation
- average the validation error over all folds



leave-one-out CV: extreme case of k=N

Train-validation-test split

We often use a 3-way split of the data

(e.g., 80%-10%-10% split)

test set:

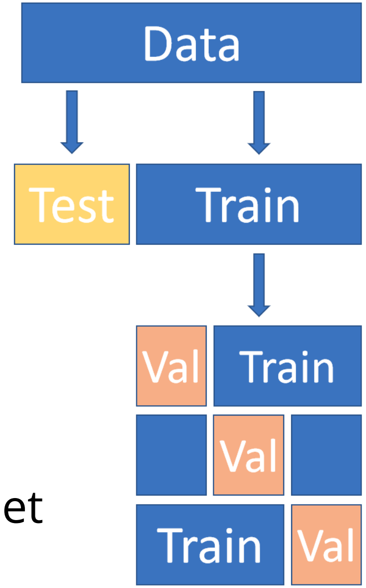
- for final evaluation

validation set (*aka* development set):

- for hyper-parameter tuning

training set:

- to train the model



we can use k-fold *cross validation* with train+validation set

No free lunch

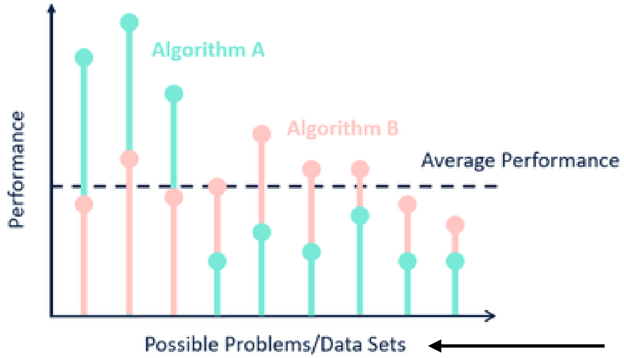
there is no single algorithm that performs well on all class of problems

image: https://community.alteryx.com/t5/Data-Science-Blog/There-is-No-Free-Lunch-in-Data-Science/ba-p/347402

No free lunch

there is no single algorithm that performs well on all class of problems

consider **any** two binary classifiers (**A** and **B**)
they have the same average performance (test accuracy) on *all possible problems*



produce labels using a random (binary) function

image: https://community.alteryx.com/t5/Data-Science-Blog/There-is-No-Free-Lunch-in-Data-Science/ba-p/347402

Inductive Bias

there is no single algorithm that performs well on all class of problems

image: https://community.alteryx.com/t5/Data-Science-Blog/There-is-No-Free-Lunch-in-Data-Science/ba-p/347402

Inductive Bias

there is no single algorithm that performs well on all class of problems

😕 *how is learning possible at all?*
🙂 *because world is not random, there are regularities, induction is possible!*

image: https://community.alteryx.com/t5/Data-Science-Blog/There-is-No-Free-Lunch-in-Data-Science/ba-p/347402

Inductive Bias

there is no single algorithm that performs well on all class of problems

😕 *how is learning possible at all?*
🙂 *because world is not random, there are regularities, induction is possible!*

ML algorithms need to make assumptions about the problem **inductive bias**

image: https://community.alteryx.com/t5/Data-Science-Blog/There-is-No-Free-Lunch-in-Data-Science/ba-p/347402

Inductive Bias

there is no single algorithm that performs well on all class of problems

😕 *how is learning possible at all?*

🙂 *because world is not random, there are regularities, induction is possible!*

ML algorithms need to make assumptions about the problem **inductive bias**

strength and **correctness** of assumptions are important in having good performance

related to bias - variance trade off that we will discuss later

image: https://community.alteryx.com/t5/Data-Science-Blog/There-is-No-Free-Lunch-in-Data-Science/ba-p/347402

Inductive Bias

there is no single algorithm that performs well on all class of problems

how is learning possible at all?
because world is not random, there are regularities, induction is possible!

ML algorithms need to make assumptions about the problem inductive bias

strength and **correctness** of assumptions are important in having good performance
related to bias - variance trade off that we will discuss later

examples

manifold hypothesis in KNN (and many other methods)
close to linear dependencies in linear regression
conditional independence and causal structure in probabilistic graphical models

image: https://community.alteryx.com/t5/Data-Science-Blog/There-is-No-Free-Lunch-in-Data-Science/ba-p/347402

Summary

ML algorithms involve a choice of **model**, **objective** and **optimization**

we saw **K-NN** method for classification

curse of dimensionality: exponentially more data needed in higher dims.

manifold hypothesis to the rescue!

what we care about is **generalization** of ML algorithms

estimated using **cross validation**

there ain't no such thing as a **free lunch**

the choice of **inductive bias** is important for good generalization