Applied Machine Learning

Some basic concepts

Reihaneh Rabbany

McGill
School of Computer Science

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 to ML

Let's focus on classification

Learning =	**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 to ML

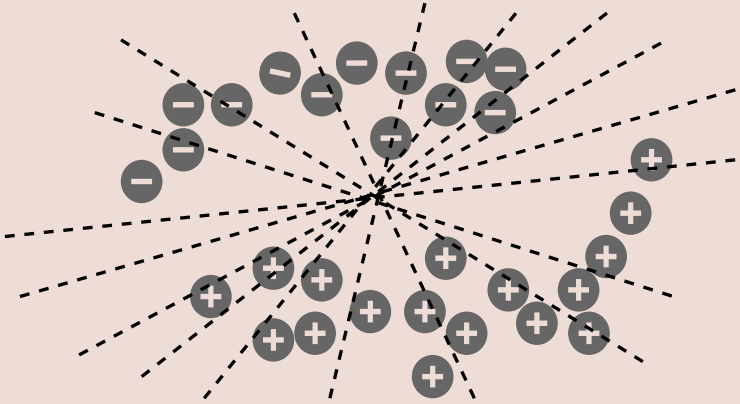
Let's focus on classification

Learning = Representation + Evaluation + Optimization

Model

Hypothesis space

$$f(x) = ax$$



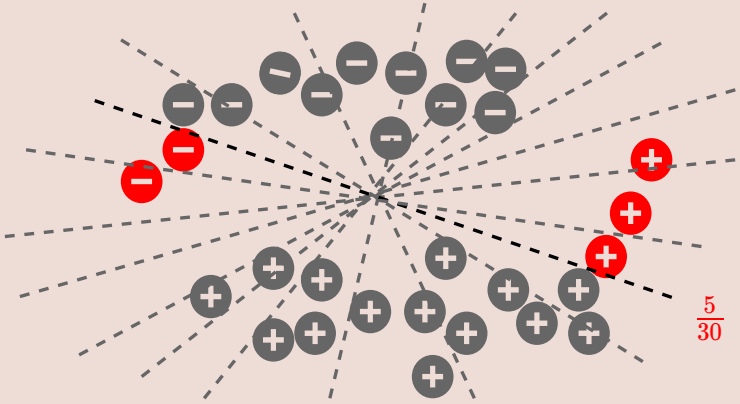
Hypothesis space is determined by the choice of representation

A useful perspective to ML

Let's focus on classification

Learning = Representation + Evaluation + Optimization

Objective function
Cost function
Score function



A useful perspective to ML

Let's focus on classification

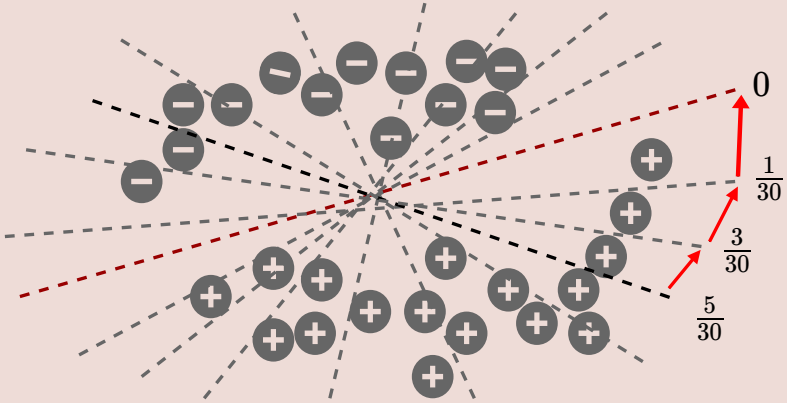
Learning = Representation + Evaluation + Optimization

Objective

Cost

Loss

$$f(x) = ax$$



A useful perspective to ML

Let's focus on classification

Learning =

Different algorithms, different combinations

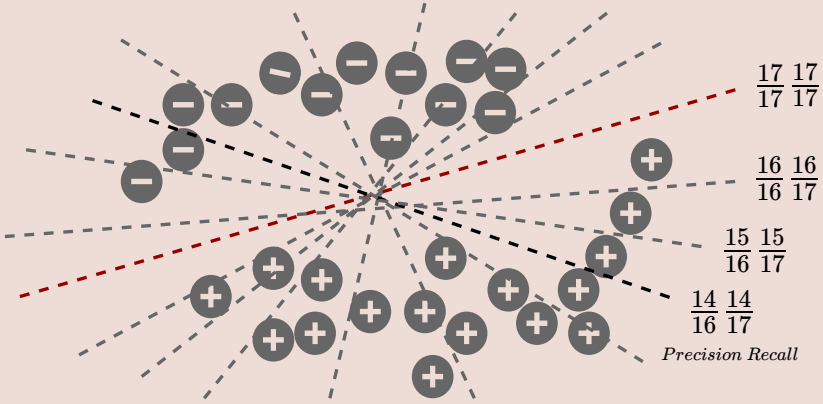
Representation	Evaluation	Optimization
Instances	Accuracy/Error rate	Combinatorial optimization
K-nearest neighbor	Precision and recall	Greedy search
Support vector machines	Squared error	Beam search
Hyperplanes	Likelihood	Branch-and-bound
Naive Bayes	Posterior probability	Continuous optimization
Logistic 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.

Quick note on measuring performance

Let's focus on binary classification

How else to measure?



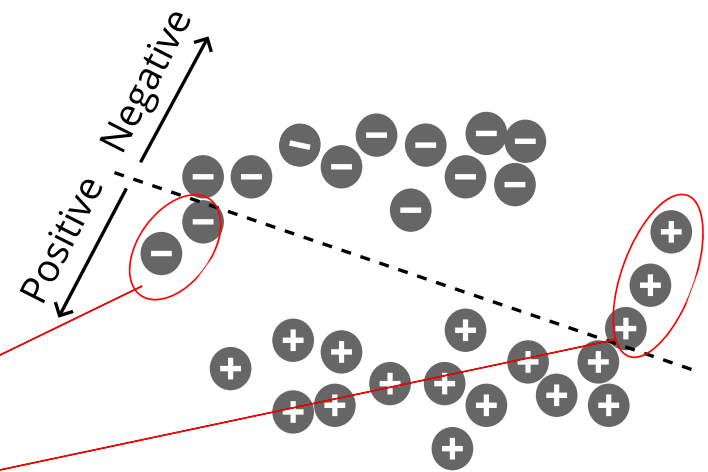
$$Precision = \frac{TP}{RP}$$

$$Recall = \frac{TP}{P}$$

Measuring performance in binary classification

	Truth		Σ
Result	TP	FP	RP
	FN	TN	RN
Σ	P	N	

	Truth		Σ
Result	14	2	16
	3	11	14
Σ	17	13	



Measuring performance in binary classification

	Truth		Σ
Result	TP	FP	RP
	FN	TN	RN
Σ	P	N	

$$RP = TP + FP$$
$$RN = FN + TN$$
$$P = TP + FN$$
$$N = FP + TN$$

$$Accuracy = \frac{TP+TN}{P+N}$$

$$Error\ rate = \frac{FP+FN}{P+N}$$

$$Precision = \frac{TP}{RP}$$

$$Recall = \frac{TP}{P}$$

$$F_1 score = 2\frac{Precision \times Recall}{Precision + Recall}$$

{Harmonic mean}

Measuring performance in binary classification

	Truth		Σ
Result	TP	FP	RP
	FN	TN	RN
Σ	P	N	

$$Accuracy = \frac{TP+TN}{P+N}$$

$$Precision = \frac{TP}{RP}$$

$$Recall = \frac{TP}{P}$$

$$F_1 score = 2\frac{Precision \times Recall}{Precision + Recall}$$ {Harmonic mean}

Less common

$$Miss\ rate = \frac{FN}{P}$$

$$Fallout = \frac{FP}{N}$$

$$False\ discovery\ rate = \frac{FP}{RP}$$

$$Selectivity = \frac{TN}{N}$$

$$False\ omission\ rate = \frac{FN}{RN}$$

$$Negative\ predictive\ value = \frac{TN}{RN}$$

A useful perspective to ML

Let's focus on classification

Learning =

Different algorithms, different combinations

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.

Some terminology

x
- input
- features
- predictors
- independent variable
- covariate

f ML function (hypothesis)

y
- output
- targets
- labels
- predictions
- dependent variable
- response variable

$\mathcal{D}$: training set

x : *D*-dimentional vector

y : a categorical or nominal variable

N : number of training instances

n : index of training instance ($n \in \{1 \dots N\}$)

indexes can placed up or down based on the notation in use, or droped all together.

When up, not to be confused with a power

labeled in supervised learning

$$\mathcal{D} = \{(x^{(n)}, y^{(n)})\}_{n=1}^{N}$$

unlabeled in unsupervised learning

$$\mathcal{D} = \{x^{(n)}\}_{n=1}^{N}$$

Some terminology

$\mathcal{D}$: training set

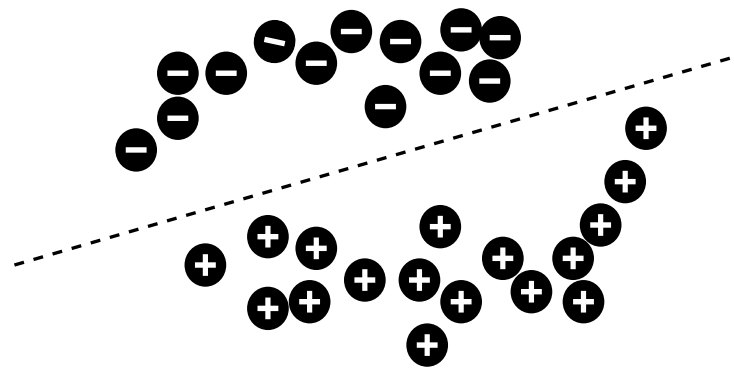
$\boldsymbol{x}$: *D*-dimentional vector

y : a categorical or nominal variable

N : number of training examples

classification: $y \in \{1 \ldots C\}$

binary classification: $y \in \{0, 1\}$



Some terminology

$\mathcal{D}$: training set
x : *D*-dimentional vector
y : a categorical or nominal variable
N : number of training examples

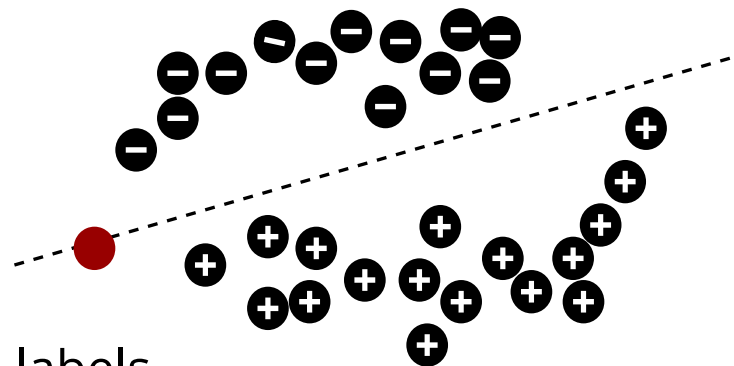
classification: $y \in \{1 \ldots C\}$

binary classification: $y \in \{0, 1\}$

probabilistic predictions

$$p(y|x, \mathcal{D})$$

probability distribution over possible labels



Some terminology

$\mathcal{D}$: training set
x : *D*-dimentional vector
y : a categorical or nominal variable
N : number of training examples

classification: $y \in \{1 \ldots C\}$

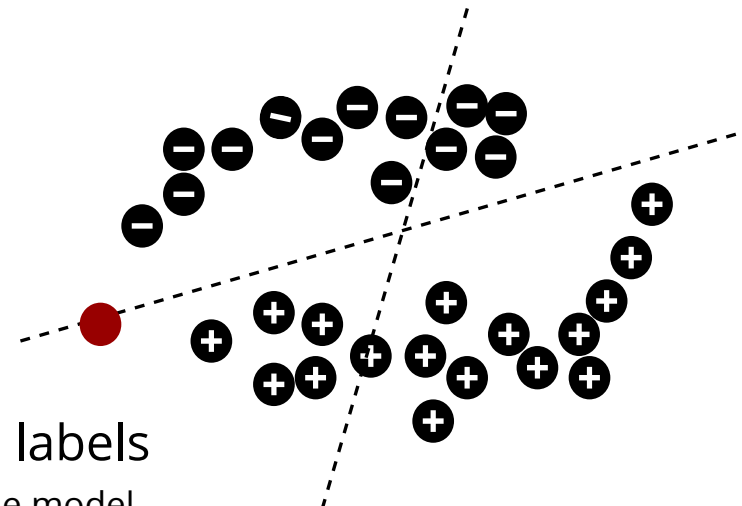
binary classification: $y \in \{0, 1\}$

probabilistic predictions

$$p(y|x, \mathcal{D}, M)$$

probability distribution over possible labels

for this instance, given all the observed data, and the model



Some terminology

$$p(y|x, \mathcal{D})$$

probability distribution over possible labels

for this instance, given all the observed data, and the model

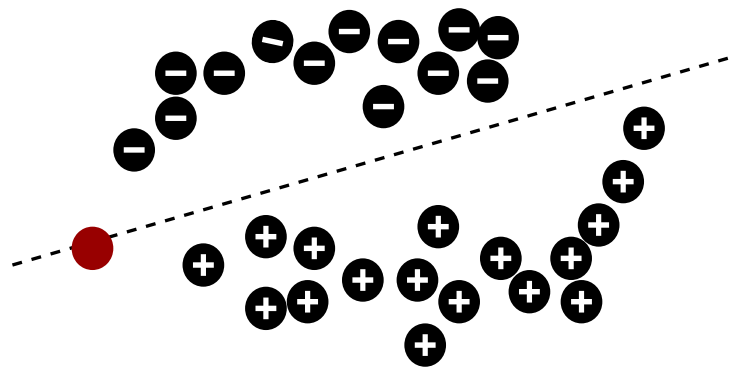
Best guess prediction:

$$\hat{y} = \hat{f}(x) = \arg\max_{c=1}^{C} p(y = c|x, \mathcal{D})$$

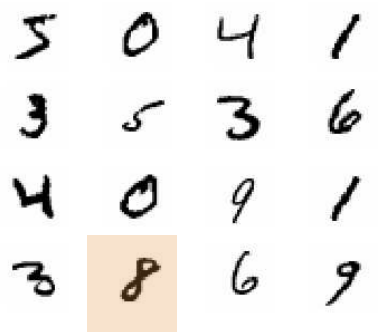
MAP estimate (maximum a posteriori)

most probable class label

mode of the distribution



Digits dataset



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

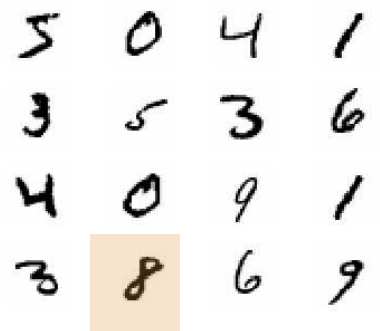
label $y^{(n)} \in \{0, \ldots, 9\}$

$n \in \{1, \ldots, N\}$ sometime we drop (n)

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

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

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

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

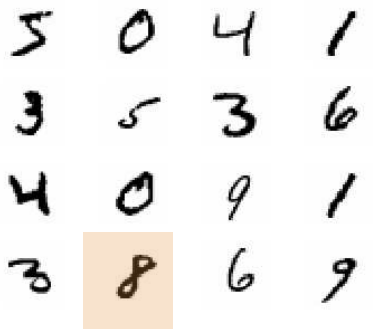
sometime we drop (n)

Classic example: **MNIST**

See https://en.wikipedia.org/wiki/MNIST_database

$N = 60K$

Digits dataset



0	0	0	0	0	0	0	0	0	0	0	0	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
0	0	1	0	0	0	41	160	250	255	235	162	255	238	206	11	13	0	•	•	•	0	0	0	0	0	0	0	0	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												

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

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:

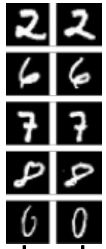
$x \to \mathrm{vec}(x) \in \mathbb{R}^{784}$ input dimension **D**

pretending intensities are real numbers

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 0 0

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

Nearest neighbour classifier



closest instance

new test instance

training: do nothing

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

need a measure of distance

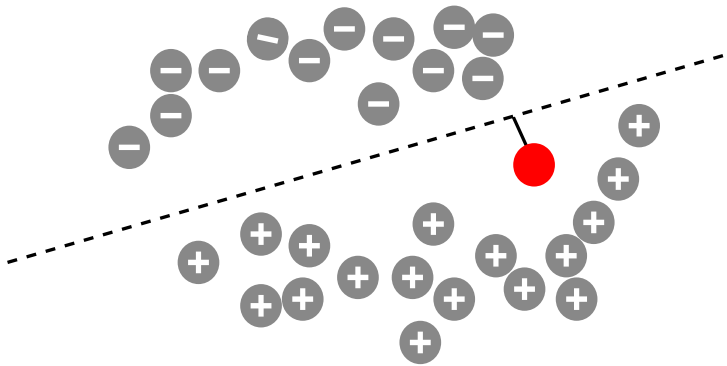
e.g., Euclidean distance $||x - x'||_2 = \sqrt{\sum_{d=1}^{D}(x_d - x'_d)^2}$

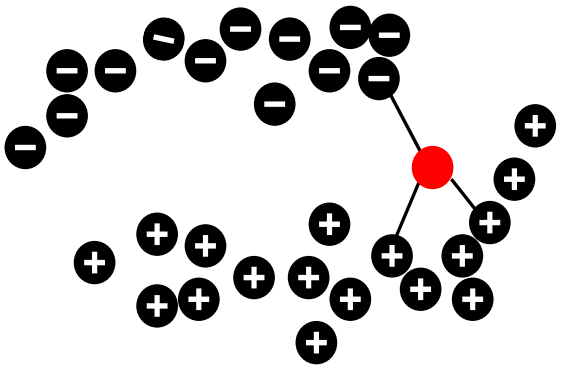
Nearest neighbour classifier

Lazy learning , or memory-based learning or instance-based learning

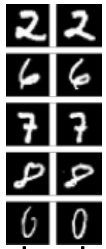
no training phase, locally estimate when a query comes

Useful for large fast-changing datasets, when trained models become obsolete in short time such as online news recommendations





Nearest neighbour classifier



closest instance

new test instance

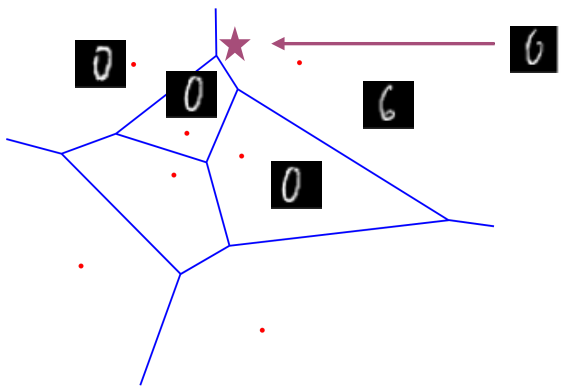
training: do nothing

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

need a measure of distance

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

d indexes the features in an instance



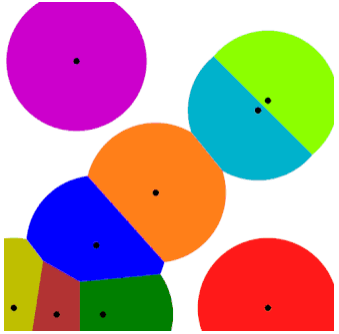
test instance: will be classified as 6

Voronoi diagram shows the decision boundaries

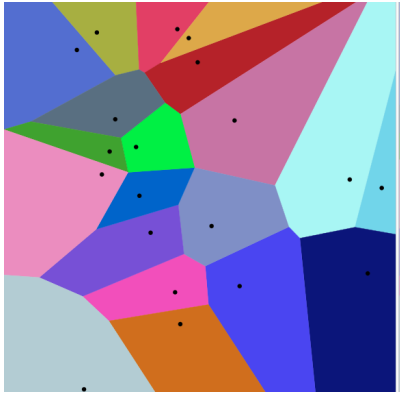
(this example D=2, can't visualize D=784)

The voronoi diagram, tessellation

Each colour shows all points closer to the corresponding seed than to any other seed



images from wiki

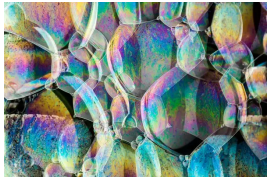


Euclidean distance

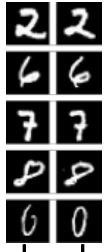
$$||x - x'||_2 = \sqrt{\sum_{d=1}^{D} (x_d - x'_d)^2}$$

Manhattan distance

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



Nearest neighbour classifier



closest instance

new test instance

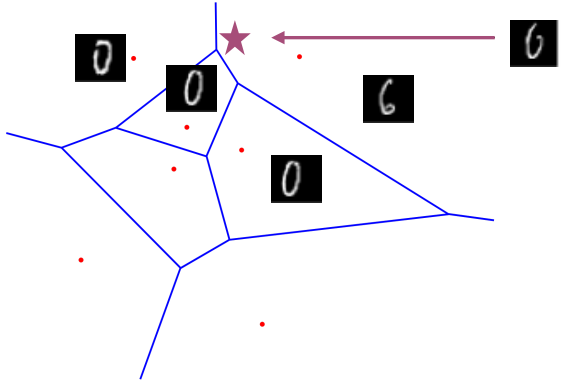
training: do nothing

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

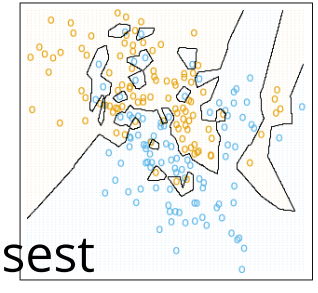
need a measure of distance

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

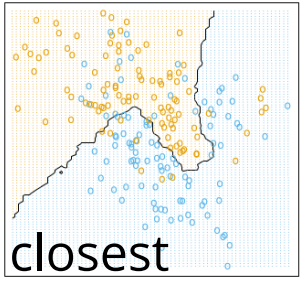
test instance: will be classified as 6



better to use K-nearest neighbours



closest



15 closest

K- nearest neighbours

training: do nothing (lazy learning)

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

$$\underbrace{p(y^{new} = c \mid x_{new})}_{\text{probability of class c}} = \frac{1}{K} \sum_{x' \in \underbrace{\mathrm{KNN}(x^{new})}_{\text{K-nearest neighbours}}} \mathbb{I}(y' = c)$$

$$\mathbb{I}(e) = \begin{cases} 1 & \text{if } e \text{ is true} \\ 0 & \text{if } e \text{ is false} \end{cases}$$

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

this is in contrast with a **parametric model** which has a fixed number of parameters, is faster to use but has stronger assumptions on the nature of the data distribution

K- nearest neighbours

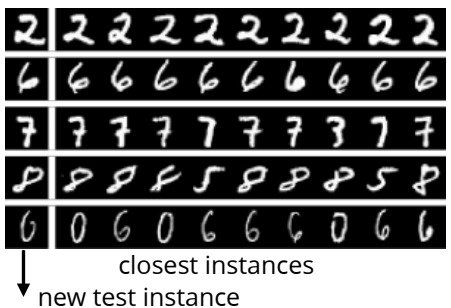
training: do nothing (lazy learning)

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

$$\underbrace{p(y^{new} = c \mid x_{new})}_{\text{probability of class c}} = \frac{1}{K} \sum_{x' \in \underbrace{\mathrm{KNN}(x^{new})}_{\text{K-nearest neighbours}}} \mathbb{I}(y' = c)$$

a non-parametric method: the number of model parameters grows with the data

example K = 9



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

K- nearest neighbours

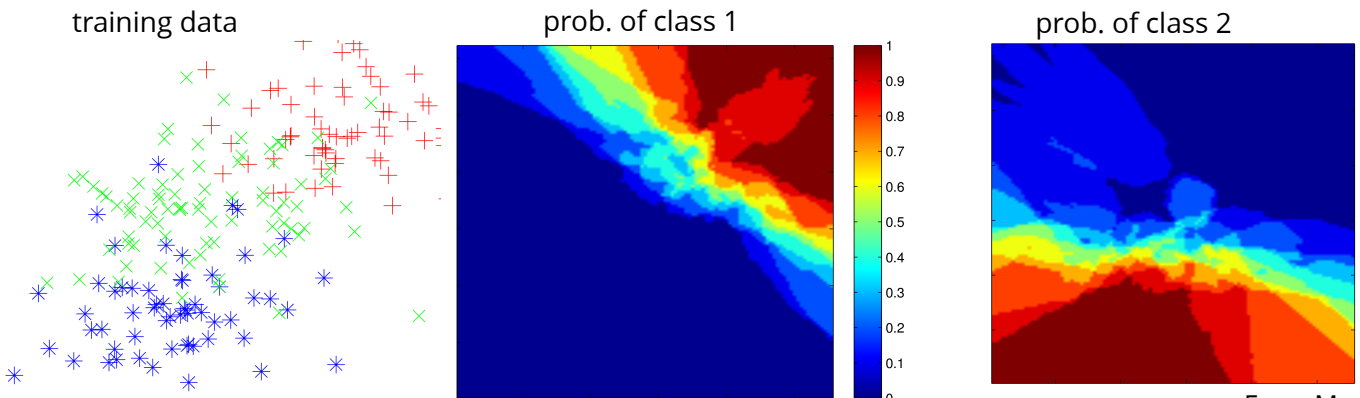
training: do nothing

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



From Murphy's book

K- nearest neighbours

training: do nothing

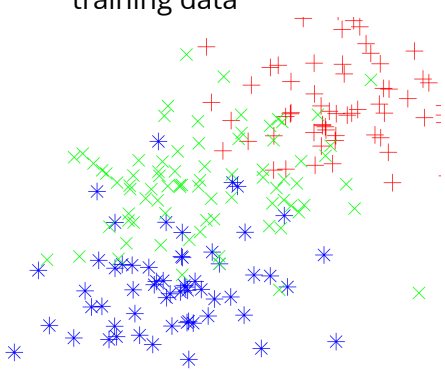
test: predict the lable by finding the **K** closest instances

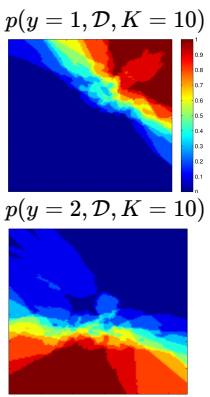
$$p(y^{new} = c \mid x_{new}) = \frac{1}{K} \sum_{x' \in \mathrm{KNN}(x^{new})} \mathbb{I}(y' = c)$$

probability of class c — K-nearest neighbours

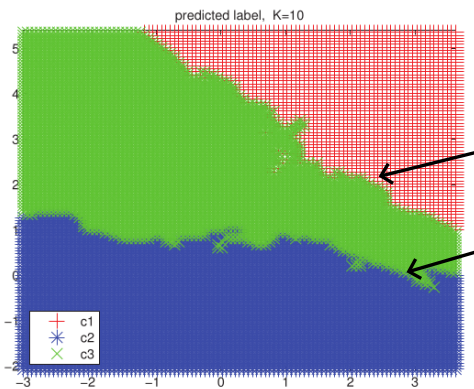
example C=3, D=2, K=10

training data





MAP estimate of class label



Decision boundaries

From Murphy's book

K- nearest neighbours

training: do nothing

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

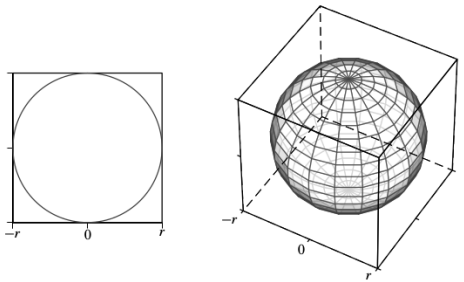
$$\underbrace{p(y^{new} = c \mid x_{new})}_{\text{probability of class c}} = \frac{1}{K} \sum_{x' \in \underbrace{\mathrm{KNN}(x^{new})}_{\text{K-nearest neighbours}}} \mathbb{I}(y' = c)$$

simple and works well if input has low dimentions

(small number of features)

Curse of dimensionality

Hypersphere inscribed inside a hypercube

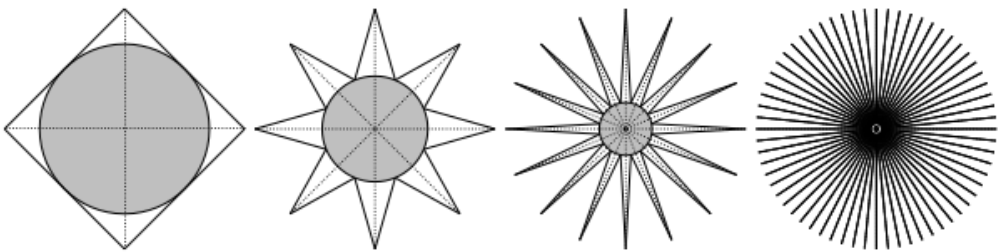


In d dimensions

how many corners? 2^d

how many diagonals? 2^{d-1}

Conceptual view of high-dimensional space



high dimensions are unintuitive!

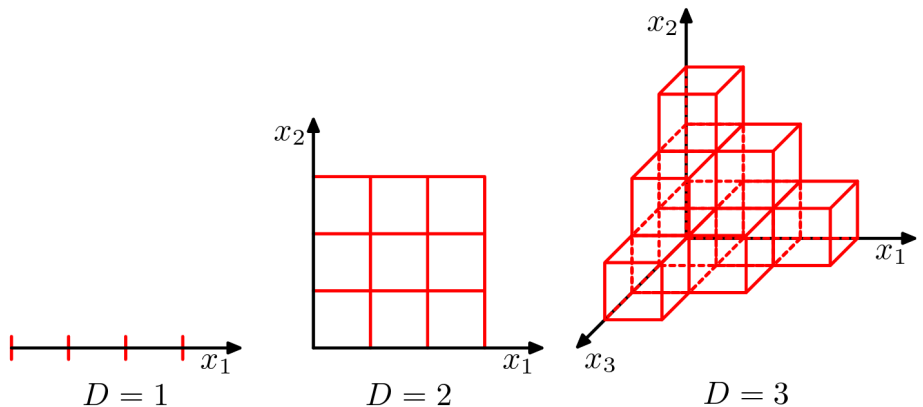
figure from Zaki's book on Data Mining and Analysis

Curse of dimensionality

high dimensions are unintuitive!

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

- need exponentially more instances for K-NN to work
- with same number of instances, the space becomes very sparse

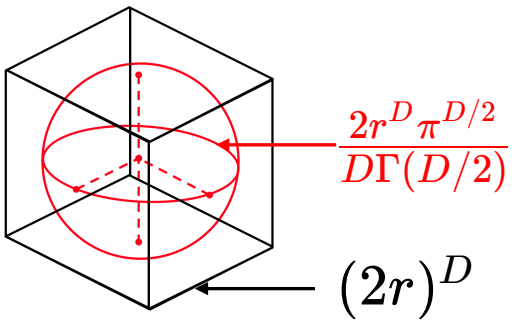


Curse of dimensionality

high dimensions are unintuitive!

assuming a uniform distribution

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



$D = 3$

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

most of the volume is close to the corners

most pairwise instances are similar

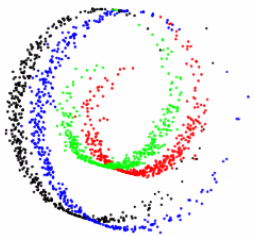
Manifold hypothesis

real-world data is often far from uniform

manifold hypothesis: they lie close to the surface of a manifold

ambient (data) dimension: $D = 3$

manifold dimension: $D^* = 2$



	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

MNIST digit classification results

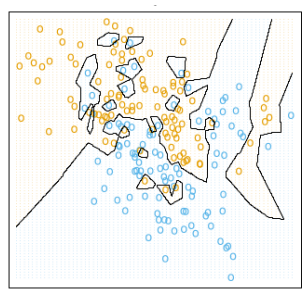
for K-NN the manifold dimension matters $D = 784$

so K-NN can be competitive

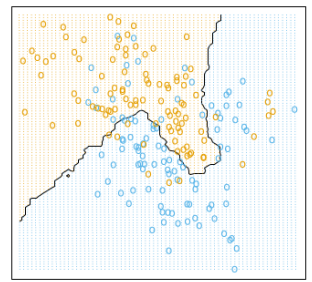
Model selection

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

example



K=1

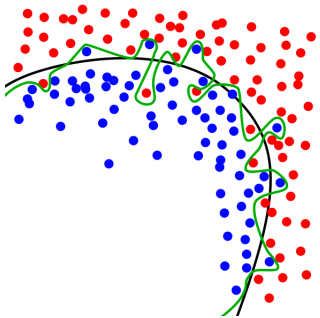


K=15

increasing k makes the classification boundary smoother, and training error might increase

- too small may overfit
- too large many underfit

Overfitting



how to pick the best K?

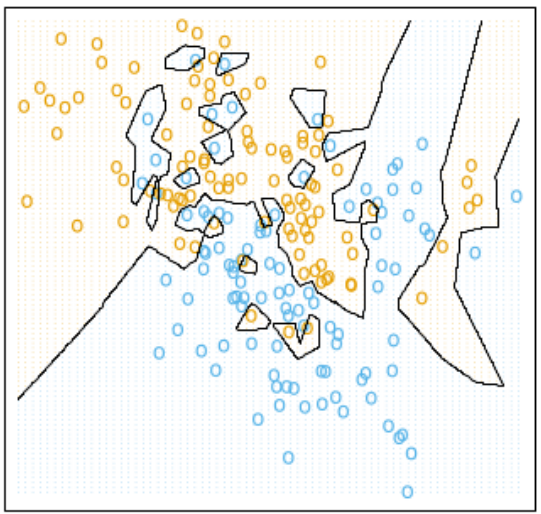
first attempt pick K that gives "best results" on the training set

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

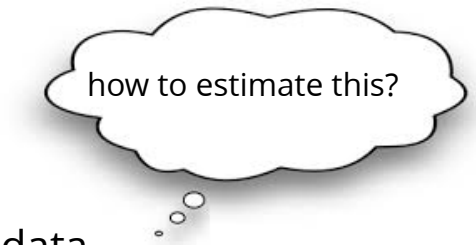
bad idea!

we can overfit the training data

we can have bad performance on new instances



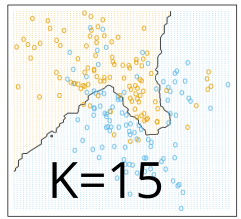
Generalization



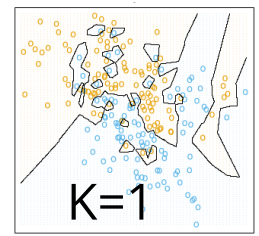
what we care about is generalization error

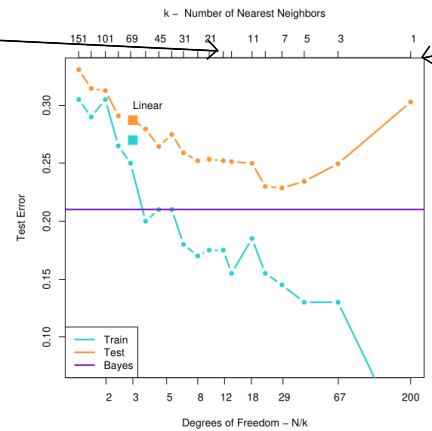
generalization: performance of algorithm on unseen data

K- nearest neighbours



K





From HTF book

choose k so that the model could generalize well, tune the hyperparameter by (estimated) generalization error on data unseen by the training algorithm

Hyperparameter tuning

hyperparameter defines the model and is not learned in the training phase

We split the dataset into

Train dataset

<tumorsize, texture, perimeter>	,	<cancer, size change>
<18.2, 27.6, 117.5>	,	< No , +2 >
<17.9, 10.3, 122.8>	,	< No , -4 >
<20.2, 14.3, 111.2>	,	< Yes , +3 >
<15.5, 15.2, 135.5>	,	< No , 0 >
		.
		.
		.

used to build the model

Test dataset

<tumorsize, texture, perimeter>	,	<cancer, size change>
<12.4, 15.7, 120.1>	,	< No , +5 >
<15.2, 17.2, 113.3>	,	< Yes , +1 >
<19.3, 15.9, 125.4>	,	< No , +2 >
<17.5, 11.9, 122.7>	,	< No , -3 >
		.
		.
		.

used to evaluate the model

🚫 algorithm shouldn't have access to this set when being trained

if tuning hyperparameters, we should still not touch the test set, we need a third split

Hyperparameter tuning

tuning hyperparameters using validation set

We further split the **training dataset** into:

Train dataset

<tumorsize, texture, perimeter>	,	<cancer, size change>
<18.2, 27.6, 117.5>	,	< No , +2 >
<17.9, 10.3, 122.8>	,	< No , -4 >
		.
		.

used to train the model

Validation dataset

<tumorsize, texture, perimeter>	,	<cancer, size change>
<20.2, 14.3, 111.2>	,	< Yes , +3 >
<15.5, 15.2, 135.5>	,	< No , 0 >
		.

used to tune hyperparameters

Test dataset

<tumorsize, texture, perimeter>	,	<cancer, size change>
<12.4, 15.7, 120.1>	,	< No , +5 >
<15.2, 17.2, 113.3>	,	< Yes , +1 >
<19.3, 15.9, 125.4>	,	< No , +2 >
<17.5, 11.9, 122.7>	,	< No , -3 >
		.
		.

used to evaluate the final model

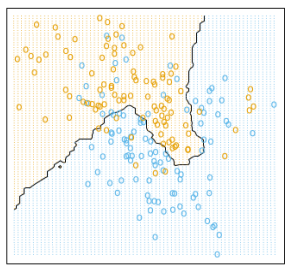
🚫 algorithm shouldn't have access to this set when being trained

Hyperparameter tuning

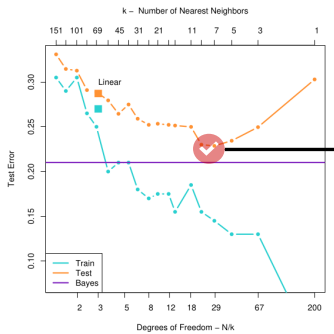
tuning hyperparameters using validation set

We split the dataset into

Train dataset | **Validation dataset** | **Test dataset**



find decision boundary for specific k using training set



search for k with least error on validation set

measure overall performance, generalization error, of the best k

Generalization

what we care about is generalization error

generalization: performance of algorithm on unseen data

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

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

How to split? what to leave out for validation?

Generalization

what we care about is generalization error

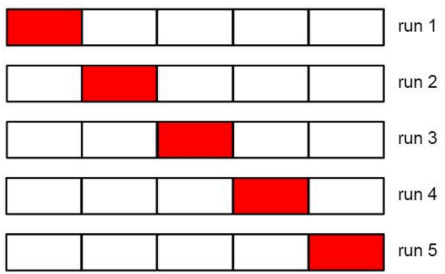
generalization: performance of algorithm on unseen data

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

performance on validation set $\approx$ *generalization 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

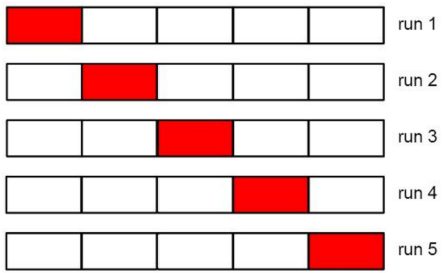


leave-one-out CV: extreme case of k=N

Cross Validation

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

shuffle data if not sure

that the order is random

Both of these are common practices with cross validation:

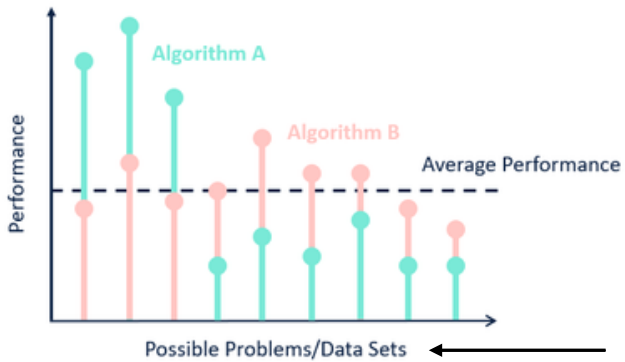
- Model selection (hyperparameter tuning) by leaving out the validation set
- Robust comparison of different ML algorithms (evaluation of performance against baselines, contenders) by leaving out the test set (often when there is no hyperparameter tuning)

Their combination is less common (folding over train+validation) after hyperparameter tuning with CV

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

☹ *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