CSCI567 Machine Learning (Spring 2025)

Haipeng Luo

University of Southern California

Jan 17, 2025

Outline

Outline

Overview

Nature of this course

- Covers both classical machine learning methods and recent advancements (supervised learning, unsupervised learning, reinforcement learning, etc.), in a systemic and rigorous way
- Particular focuses are on the conceptual understanding and derivation of these methods

Learning objectives:

- Hone skills on grasping abstract concepts and thinking critically to solve problems with ML techniques
- Solidify your knowledge with hand-on programming tasks
- Prepare you for studying advanced ML techniques

Teaching logistics

Lectures: Friday, 1:00-3:20pm

Discussions: Friday, 3:30-4:20pm (by TAs, same locations)

Online platforms

Web: https://haipeng-luo.net/courses/CSCI567/2025_spring

- general information (schedule, slides, homework, etc.)

Piazza: https://piazza.com/usc/spring2025/csci567

- main discussion forum
- everyone has to enroll!

DEN: https://courses.uscden.net/d2l/login

- recorded lectures/discussions

Gradescope: https://www.gradescope.com

- submit homework

Vocareum: https://www.vocareum.com/

- programming project

Teaching staff

4 TAs

- Dongze Ye
- Xiao Fu
- Soumita Hait
- Robby Costales

2 graders (for grading homework only)

- Joonyoung (Aaron) Bae
- Mounika Mukkamalla

Emails and office hours are on the course website

- note: location for office hours might vary during the semester

Prerequisites

- Undergraduate level training in **probability and statistics, linear algebra, (multivariate) calculus**

  Important: attend today's discussion session to see if you have the required background

- Programming: Python and necessary packages (e.g. numpy)

  not an intro-level CS course, no training of basic programming skills.

Slides and readings

Lectures

Lecture slides/handouts will be posted before the class (and possibly slightly updated after).

Readings

- No required textbooks
- Main recommended readings:
    - Probabilistic Machine Learning: An Introduction by Kevin Murphy
    - Elements of Statistical Learning by Hastie, Tibshirani and Friedman
- More: see course website

Grade

Structure:

- 40%: 4 written assignments
- 40%: 2 quizzes
- 20%: 1 programming project

Initial cut-offs (for A and B):

- B- = [70,75), B = [75, 80), B+ = [80, 86)
- A- = [86, 92), A = [92, 100]

Important: final cut-offs will NOT be released. If adjusted they could only be LOWER.

Homework

4 written assignments (problem sets):

- submit through gradescope (scanned copy or typeset with LaTeX etc.)
- graded based on correctness; solutions/rubrics will be released
- finding solutions online or from other sources → *zero grade*
- 3 late days in total, at most *one* can be used for each assignment
- A two-day window for re-grading (regarding *factual errors*)

Quizzes

First one on **03/07**, second one on **05/02**. In class, 1:00-3:20.

- for special arrangements, inform us within the first two weeks

Format/logistic

- double-seating, individual effort, close-book,
- multiple-choice and general problems that are similar to HW
- sample quizzes will be available

Programing Project

Done on **Vocareum**

- easy-to-use platform to submit your code for auto-grading
- you will be invited to register next week
- consists of about 10 tasks (in Python) with detailed descriptions
- skeleton provided, only need to fill in some key components
- you can make *unlimited submissions* and see your grade immediately
- the project is available throughout the semester (*due on 05/13*, no late days)

Academic honesty and integrity

Zero tolerance for plagiarism and other unacceptable violations:

- finding solutions online, including using chatbots such as ChatGPT
- uploading any material from the course to the Internet

Learn how to ask questions effectively

Very important communication skills.

Bad examples from the past:

- My code passes some cases, but not the others, why? (and it was an anonymous post!)
- I couldn't get the same result as in Slide X, why?

Bottom line: *help us help you by asking informative questions!*

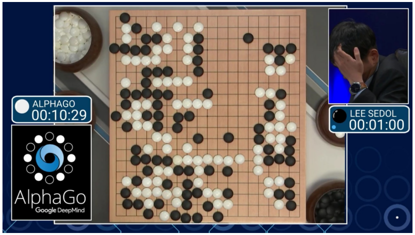
Outline

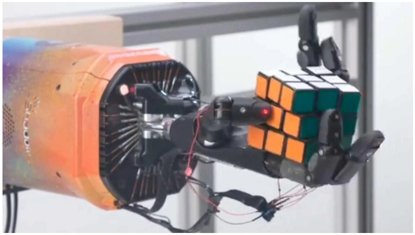
Machine learning: the driving force of AI

Recent amazing AI advances: generative AI

Machine learning: the driving force of AI

Recent amazing AI advances: intelligent planning





Machine learning: the driving force of AI

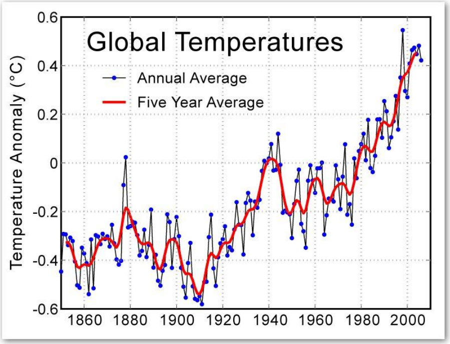
Recent amazing AI advances: AI for science

What is machine learning?

One possible definition (cf. Murphy's book)
a set of methods that can automatically *detect patterns* in data, and then use the uncovered patterns to *predict future data*, or to perform other kinds of *decision making under uncertainty*

Example: detect patterns

How the temperature has been changing?

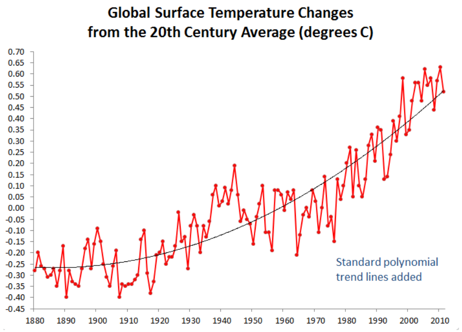


Patterns

- Seems going up
- Repeated periods of going up and down.

How do we describe the pattern?

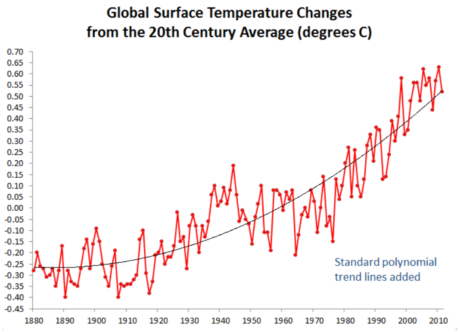
Build a model: fit the data with a polynomial function



- The model is not accurate for individual years
- But collectively, the model captures the major trend

Predicting future

What is temperature of 2030?



- Again, the model is probably inaccurate for that specific year
- But it might be close enough

What we have learned from this example?

Key ingredients in machine learning

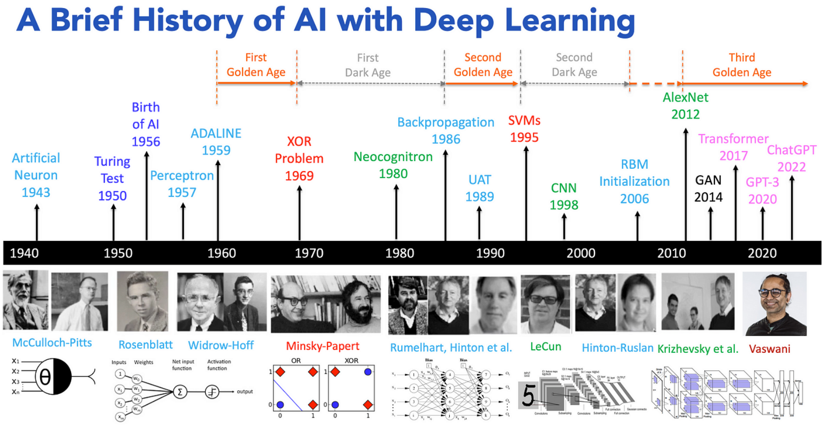
- Data
  collected from past observation (we often call them *training data*)
- Modeling
  devised to capture the patterns in the data
  - The model does not have to be true — "All models are wrong, but some are useful" by George Box.
- Prediction
  apply the model to forecast what is going to happen in future

A rich history of applying statistical learning methods

Recognizing flowers (by R. Fisher, 1936)

Types of Iris: setosa, versicolor, and virginica

Huge success with the rise of “deep” learning



What is in machine learning?

Different flavors of learning problems

- Supervised learning
  Aim to predict (as in previous examples)
- Unsupervised learning
  Aim to discover hidden patterns and explore data
- Decision making (e.g. reinforcement learning)
  Aim to act optimally under uncertainty
- often mixed together in one application!

The main focus and goal of this course

- Supervised learning (before Quiz 1)
- Unsupervised learning and reinforcement learning (after Quiz 1)

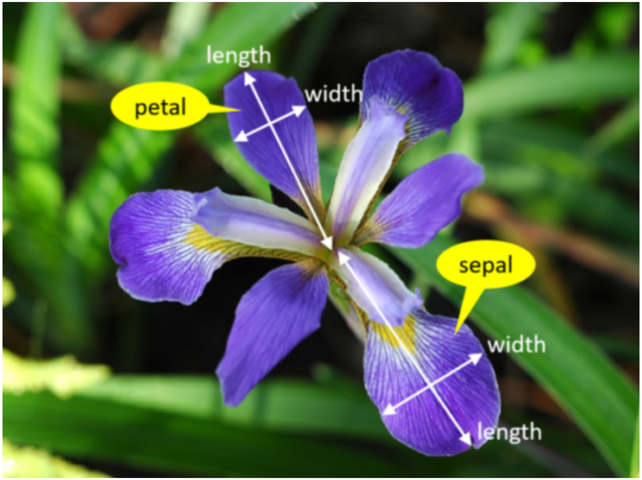
Outline

Recognizing flowers

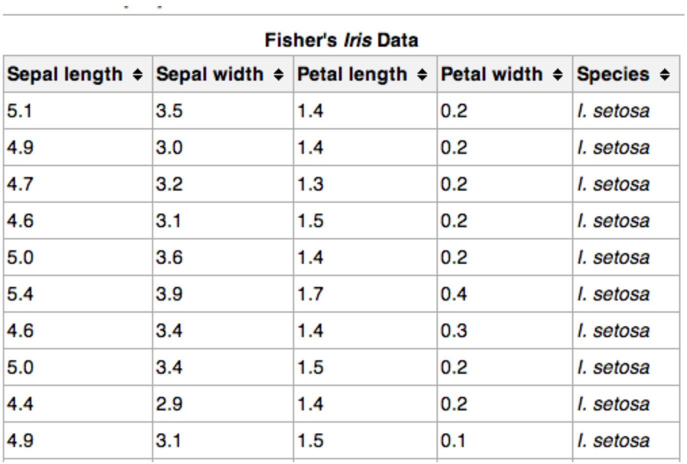
Types of Iris: setosa, versicolor, and virginica

Measuring the properties of the flowers

Features and attributes: the widths and lengths of sepal and petal



Often, data is conveniently organized as a table

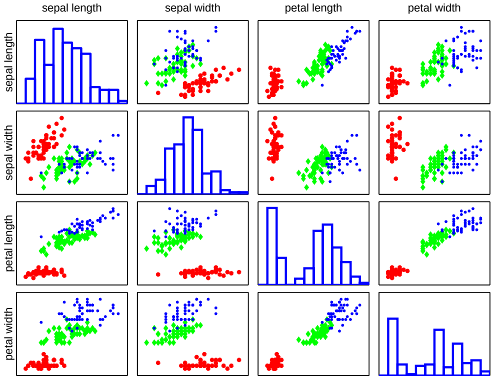

Fisher's *Iris* Data

Sepal length ⬍	Sepal width ⬍	Petal length ⬍	Petal width ⬍	Species ⬍
5.1	3.5	1.4	0.2	*I. setosa*
4.9	3.0	1.4	0.2	*I. setosa*
4.7	3.2	1.3	0.2	*I. setosa*
4.6	3.1	1.5	0.2	*I. setosa*
5.0	3.6	1.4	0.2	*I. setosa*
5.4	3.9	1.7	0.4	*I. setosa*
4.6	3.4	1.4	0.3	*I. setosa*
5.0	3.4	1.5	0.2	*I. setosa*
4.4	2.9	1.4	0.2	*I. setosa*
4.9	3.1	1.5	0.1	*I. setosa*

Pairwise scatter plots of 131 flower specimens

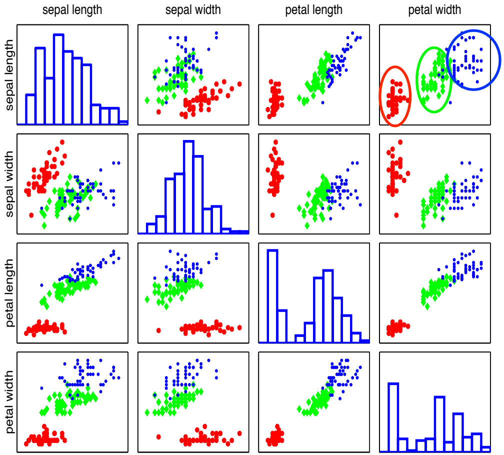
Visualization of data helps identify the right learning model to use

Each colored point is a flower specimen: setosa, versicolor, virginica



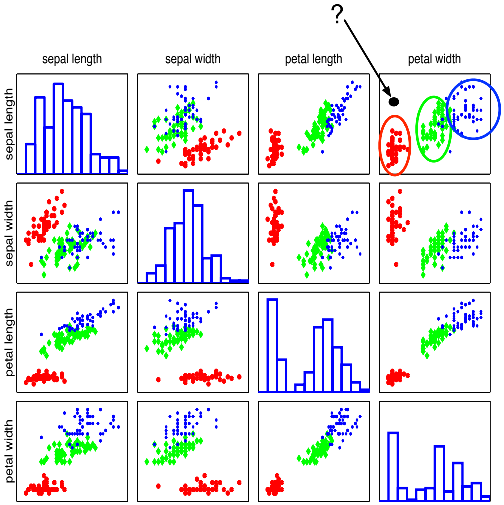
Different types seem well-clustered and separable

Using two features: petal width and sepal length



Labeling an unknown flower type

Closer to red cluster: so predict setosa



General setup for multi-class classification

Training data (set)

- N samples/instances: $\mathcal{D}^{\text{TRAIN}} = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \cdots, (\boldsymbol{x}_\mathsf{N}, y_\mathsf{N})\}$
- Each $\boldsymbol{x_n} \in \mathbb{R}^\mathsf{D}$ is called a feature vector.
- Each $y_n \in [\mathsf{C}] = \{1, 2, \cdots, \mathsf{C}\}$ is called a label/class/category.
- They are used to learn a *classifier* $f : \mathbb{R}^\mathsf{D} \rightarrow [\mathsf{C}]$ for future prediction.

Special case: binary classification

- Number of classes: $\mathsf{C} = 2$
- Conventional labels: $\{0, 1\}$ or $\{-1, +1\}$ (instead of $\{1, 2\}$)

Nearest neighbor classification (NNC)

The index of the **nearest neighbor** of a point $\boldsymbol{x}$ is

$$\mathsf{nn}(\boldsymbol{x}) = \underset{n \in [\mathsf{N}]}{\operatorname{argmin}} \|\boldsymbol{x} - \boldsymbol{x}_n\|_2 = \underset{n \in [\mathsf{N}]}{\operatorname{argmin}} \sqrt{\sum_{d=1}^{\mathsf{D}} (x_d - x_{nd})^2}$$

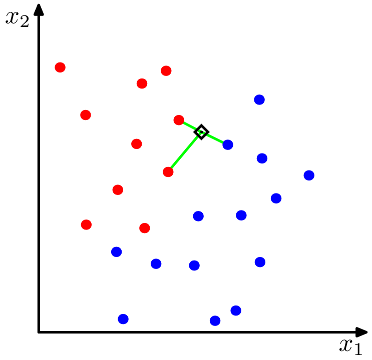
where $\| \cdot \|_2$ is the L_2/Euclidean distance.

Classification rule

$$f(\boldsymbol{x}) = y_{\mathsf{nn}(\boldsymbol{x})}$$

Visual example

In this 2-dimensional example, the nearest point to $\boldsymbol{x}$ is a red training instance, thus, $\boldsymbol{x}$ will be labeled as red.



(a)

Example: classify Iris with two features

Training data

ID (n)	petal width (x_1)	sepal length (x_2)	category (y)
1	0.2	5.1	setoas
2	1.4	7.0	versicolor
3	2.5	6.7	virginica
⋮	⋮	⋮	

A new specimen with unknown category:

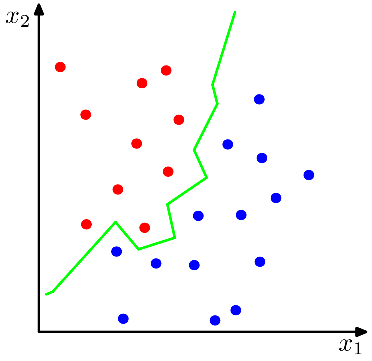
petal width $= 1.8$ and sepal length $= 6.4$ (i.e. $\boldsymbol{x} = (1.8, 6.4)$)
Calculating distance $\|\boldsymbol{x} - \boldsymbol{x}_n\|_2 = \sqrt{(x_1 - x_{n1})^2 + (x_2 - x_{n2})^2}$

ID	distance
1	2.06
2	0.72
3	0.76

Thus, the prediction is *versicolor*.

Decision boundary

For every point in the space, we can determine its label using the NNC rule. This gives rise to a *decision boundary* that partitions the space into different regions.



(b)

Is NNC doing the right thing for us?

Intuition

We should compute accuracy — the percentage of data points being correctly classified, or the error rate — the percentage of data points being incorrectly classified. (accuracy + error rate = 1)

Defined on the training data set

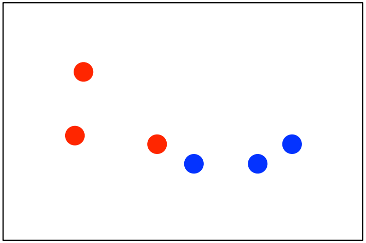
$$A^{\text{TRAIN}} = \frac{1}{\mathsf{N}} \sum_n \mathbb{I}[f(\boldsymbol{x}_n) == y_n], \quad \varepsilon^{\text{TRAIN}} = \frac{1}{\mathsf{N}} \sum_n \mathbb{I}[f(\boldsymbol{x}_n) \neq y_n]$$

where $\mathbb{I}[\cdot]$ is the indicator function.

Is this the right measure?

Example

Training data



What are A^{TRAIN} and $\varepsilon^{\text{TRAIN}}$?

$$A^{\text{TRAIN}} = 100\%, \quad \varepsilon^{\text{TRAIN}} = 0\%$$

For every training data point, its nearest neighbor is itself.

Test Error

Does it mean nearest neighbor is a very good algorithm?

Not really, having zero training error is simple!

We should care about accuracy when predicting unseen data

Test/Evaluation data

- $\mathcal{D}^{\text{TEST}} = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \cdots, (\boldsymbol{x}_{\mathsf{M}}, y_{\mathsf{M}})\}$
- A fresh dataset, *not* overlap with training set.
- Test accuracy and test error

$$A^{\text{TEST}} = \frac{1}{\mathsf{M}} \sum_m \mathbb{I}[f(\boldsymbol{x}_m) == y_m], \quad \varepsilon^{\text{TEST}} = \frac{1}{\mathsf{M}} \sum_m \mathbb{I}[f(\boldsymbol{x}_m) \neq y_m]$$

- Good measurement of a classifier's performance

Variant 1: measure nearness with other distances

Previously, we use the Euclidean distance

$$\mathsf{nn}(\boldsymbol{x}) = \operatorname*{argmin}_{n \in [\mathsf{N}]} \|\boldsymbol{x} - \boldsymbol{x}_n\|_2$$

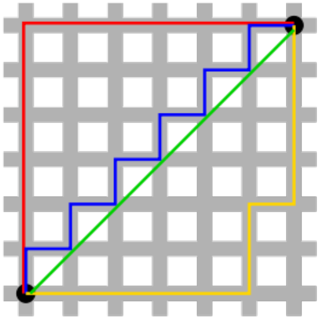
Many other alternative distances

E.g., the following L_1 distance (i.e., city block distance, or Manhattan distance)

$$\|\boldsymbol{x} - \boldsymbol{x}_n\|_1 = \sum_{d=1}^{\mathsf{D}} |x_d - x_{nd}|$$

More generally, L_p distance (for $p \geq 1$):

$$\|\boldsymbol{x} - \boldsymbol{x}_n\|_p = \left(\sum_d |x_d - x_{nd}|^p \right)^{1/p}$$



Green line is Euclidean distance. Red, Blue, and Yellow lines are L_1 distance

Variant 2: K-nearest neighbor (KNN)

Increase the number of nearest neighbors to use?

- 1st-nearest neighbor: $\mathsf{nn}_1(\boldsymbol{x}) = \operatorname{argmin}_{n \in [\mathsf{N}]} \|\boldsymbol{x} - \boldsymbol{x}_n\|_2$
- 2nd-nearest neighbor: $\mathsf{nn}_2(\boldsymbol{x}) = \operatorname{argmin}_{n \in [\mathsf{N}] \backslash \{\mathsf{nn}_1(\boldsymbol{x})\}} \|\boldsymbol{x} - \boldsymbol{x}_n\|_2$
- 3rd-nearest neighbor: $\mathsf{nn}_3(\boldsymbol{x}) = \operatorname{argmin}_{n \in [\mathsf{N}] \backslash \{\mathsf{nn}_1(\boldsymbol{x}), \mathsf{nn}_2(\boldsymbol{x})\}} \|\boldsymbol{x} - \boldsymbol{x}_n\|_2$

The set of K-nearest neighbor

$$\mathsf{knn}(\boldsymbol{x}) = \{\mathsf{nn}_1(\boldsymbol{x}), \mathsf{nn}_2(\boldsymbol{x}), \cdots, \mathsf{nn}_K(\boldsymbol{x})\}$$

Note: we have

$$\|\boldsymbol{x} - \boldsymbol{x}_{\mathsf{nn}_1(\boldsymbol{x})}\|_2 \leq \|\boldsymbol{x} - \boldsymbol{x}_{\mathsf{nn}_2(\boldsymbol{x})}\|_2 \cdots \leq \|\boldsymbol{x} - \boldsymbol{x}_{\mathsf{nn}_K(\boldsymbol{x})}\|_2$$

How to classify with K neighbors?

Classification rule

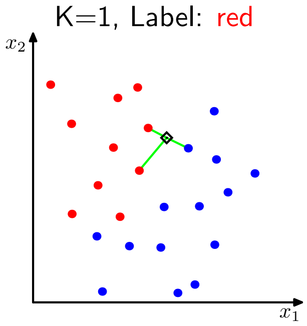
- Every neighbor votes: naturally $\boldsymbol{x}_n$ votes for its label y_n.
- Aggregate everyone's vote on a class label c

$$v_c = \sum_{n \in \mathsf{knn}(\boldsymbol{x})} \mathbb{I}(y_n == c), \quad \forall \quad c \in [\mathsf{C}]$$

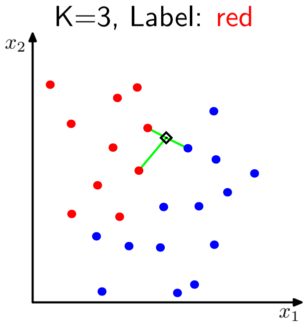
- Predict with the majority

$$f(\boldsymbol{x}) = \operatorname*{argmax}_{c \in [\mathsf{C}]} v_c$$

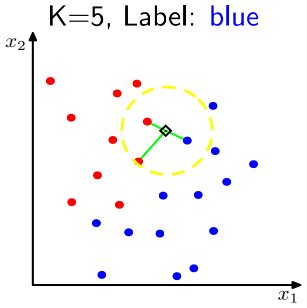
Example



(a)

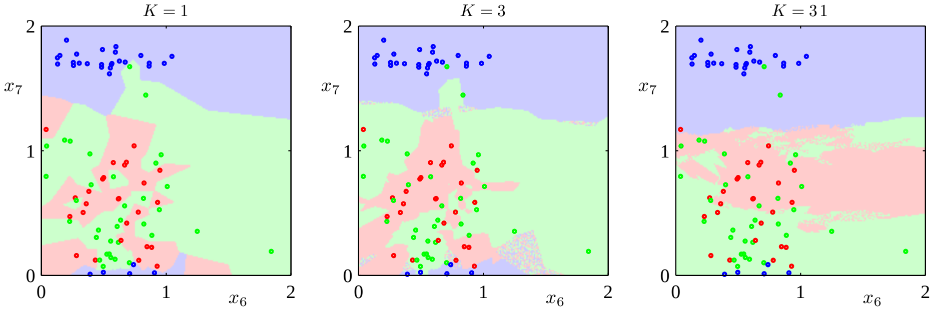


(a)



(a)

Decision boundary



When K increases, the decision boundary becomes smoother.

What happens when $K = N$?

Variant 3: Preprocessing data

One issue of NNC: *distances depend on units of the features!*

One solution: preprocess data so it looks more “normalized”.

Example:

- compute the means and standard deviations in each feature

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

- Scale the feature accordingly

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

Many other ways of normalizing data.

Which variants should we use?

Hyper-parameters in NNC

- The distance measure (e.g. the parameter p for L_p norm)
- K (i.e. how many nearest neighbor?)
- Different ways of preprocessing

Most algorithms have hyper-parameters. Tuning them is a significant part of applying an algorithm.

Tuning via a validation dataset

Training data

- N samples/instances: $\mathcal{D}^{\text{TRAIN}} = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \cdots, (\boldsymbol{x}_\mathsf{N}, y_\mathsf{N})\}$
- They are used to learn $f(\cdot)$

Test data

- M samples/instances: $\mathcal{D}^{\text{TEST}} = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \cdots, (\boldsymbol{x}_\mathsf{M}, y_\mathsf{M})\}$
- They are used to evaluate how well $f(\cdot)$ will do.

Validation/Development data

- L samples/instances: $\mathcal{D}^{\text{DEV}} = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \cdots, (\boldsymbol{x}_\mathsf{L}, y_\mathsf{L})\}$
- They are used to optimize hyper-parameter(s).

These three sets should *not* overlap!

Recipe

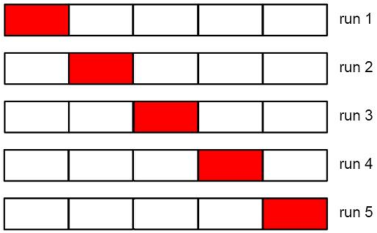
- For each possible value of the hyperparameter (e.g. $K = 1, 3, \cdots$)
  - Train a model using $\mathcal{D}^{\mathrm{TRAIN}}$
  - Evaluate the performance of the model on $\mathcal{D}^{\mathrm{DEV}}$
- Choose the model with the best performance on $\mathcal{D}^{\mathrm{DEV}}$
- Evaluate the model on $\mathcal{D}^{\mathrm{TEST}}$

S-fold Cross-validation

What if we do not have a validation set?

- Split the training data into S equal parts.
- Use each part *in turn* as a validation dataset and use the others as a training dataset.
- Choose the hyper-parameter leading to best *average* performance.

S $= 5$: 5-fold cross validation



Special case: S = N, called leave-one-out.

Cross-validation recipe

- Split the training data into S equal parts. Denote each part as $\mathcal{D}_s^{\text{TRAIN}}$.
- For each possible value of the hyper-parameter (e.g. $K = 1, 3, \cdots$)
    - For every $s \in$ [S]
        - Train a model using $\mathcal{D}_{\backslash s}^{\text{TRAIN}} = \mathcal{D}^{\text{TRAIN}} - \mathcal{D}_s^{\text{TRAIN}}$
        - Evaluate the performance of the model on $\mathcal{D}_s^{\text{TRAIN}}$
    - Average the S performance metrics
- Choose the hyper-parameter with the best averaged performance
- **Use the best hyper-parameter to train a model using all $\mathcal{D}^{\textbf{train}}$**
- Evaluate the model on $\mathcal{D}^{\text{TEST}}$

Summary

Advantages of NNC

- Simple, easy to implement (wildly used in practice)

Disadvantages of NNC

- Computationally intensive for large-scale problems: $O(ND)$ for each prediction *naively*.
- Need to *"carry"* the training data around. This type of method is called *nonparametric*.
- Choosing the right hyper-parameters can be involved.

Summary

Typical steps of developing a machine learning system:

- Collect data, split into training, validation, and test sets.
- Train a model with a machine learning algorithm. Most often we apply cross-validation to tune hyper-parameters.
- Evaluate using the test data and report performance.
- Use the model to predict future/make decisions.

Outline

How good is NNC really?

To answer this question, we proceed in 3 steps

1. Define *more carefully* a performance metric for a classifier.
2. Hypothesize an ideal classifier - *the best possible one*.
3. Compare NNC to the ideal one.

Why does test error make sense?

Test error makes sense only when training set and test set are correlated.

Most standard assumption: every data point $(\boldsymbol{x}, y)$ (from $\mathcal{D}^{\mathrm{TRAIN}}$, $\mathcal{D}^{\mathrm{DEV}}$, or $\mathcal{D}^{\mathrm{TEST}}$) is an *independently and identically distributed (i.i.d.)* sample of an unknown joint distribution $\mathcal{P}$.

- often written as $(\boldsymbol{x}, y) \overset{i.i.d.}{\sim} \mathcal{P}$

Test error of a fixed classifier is therefore a *random variable*.

Need a more "certain" measure of performance (so it's easy to compare different classifiers for example).

Expected error

What about the **expectation** of this random variable?

$$\mathbb{E}[\epsilon^{\mathrm{TEST}}] = \frac{1}{M}\sum_{m=1}^{M}\mathbb{E}_{(\boldsymbol{x_m},y_m)\sim\mathcal{P}}\mathbb{I}[f(\boldsymbol{x}_m)\neq y_m] = \mathbb{E}_{(\boldsymbol{x},y)\sim\mathcal{P}}\mathbb{I}[f(\boldsymbol{x})\neq y]$$

- i.e. the expected error/mistake of f

Test error is a proxy of expected error. *The larger the test set, the better the approximation.*

What about the expectation of training error? Is training error a good proxy of expected error?

Expected risk

More generally, for a loss function $L(y', y)$,

- e.g. $L(y', y) = \mathbb{I}[y' \neq y]$, called *0-1 loss*. **Default**
- many more other losses as we will see.

the *expected risk* of f is defined as

$$R(f) = \mathbb{E}_{(\boldsymbol{x},y)\sim\mathcal{P}} L(f(\boldsymbol{x}), y)$$

Bayes optimal classifier

What should we predict for $\boldsymbol{x}$, *knowing* $\mathcal{P}(y|\boldsymbol{x})$?

Bayes optimal classifier: $f^*(\boldsymbol{x}) = \operatorname{argmax}_{c\in[C]} \mathcal{P}(c|\boldsymbol{x})$.

The optimal risk: $R(f^*) = \mathbb{E}_{\boldsymbol{x}\sim\mathcal{P}_{\boldsymbol{x}}}[1 - \max_{c\in[C]} \mathcal{P}(c|\boldsymbol{x})]$ where $\mathcal{P}_{\boldsymbol{x}}$ is the marginal distribution of $\boldsymbol{x}$.

It is easy to show $R(f^*) \leq R(f)$ for any f.

For special case $C = 2$, let $\eta(\boldsymbol{x}) = \mathcal{P}(0|\boldsymbol{x})$, then

$$R(f^*) = \mathbb{E}_{\boldsymbol{x}\sim\mathcal{P}_{\boldsymbol{x}}}[\min\{\eta(\boldsymbol{x}), 1 - \eta(\boldsymbol{x})\}].$$

Comparing NNC to Bayes optimal classifier

Come back to the question: how good is NNC?

Theorem (Cover and Hart, 1967)

Let f_N be the 1-nearest neighbor binary classifier using N training data points, we have (under mild conditions)

$$R(f^*) \leq \lim_{N\to\infty} \mathbb{E}[R(f_N)] \leq 2R(f^*)$$

i.e., expected risk of NNC in the limit is at most twice of the best possible.

A pretty strong guarantee.
In particular, $R(f^*) = 0$ implies $\mathbb{E}[R(f_N)] \to 0$.

Proof sketch

Fact: $\boldsymbol{x}_{\mathsf{nn}(\boldsymbol{x})} \to \boldsymbol{x}$ as $N \to \infty$ with probability 1

$$\begin{aligned}
\mathbb{E}[R(f_N)] &= \mathbb{E}[\mathbb{E}_{(\boldsymbol{x},y)\sim\mathcal{P}}\mathbb{I}[f_N(\boldsymbol{x}) \neq y]] \\
&\to \mathbb{E}_{\boldsymbol{x}\sim\mathcal{P}_{\boldsymbol{x}}}\mathbb{E}_{y,y' \overset{i.i.d.}{\sim} \mathcal{P}(\cdot|\boldsymbol{x})}[\mathbb{I}[y' \neq y]] \\
&= \mathbb{E}_{\boldsymbol{x}\sim\mathcal{P}_{\boldsymbol{x}}}\mathbb{E}_{y,y' \overset{i.i.d.}{\sim} \mathcal{P}(\cdot|\boldsymbol{x})}[\mathbb{I}[y' = 0 \text{ and } y = 1] + \mathbb{I}[y' = 1 \text{ and } y = 0]] \\
&= \mathbb{E}_{\boldsymbol{x}\sim\mathcal{P}_{\boldsymbol{x}}}[\eta(\boldsymbol{x})(1-\eta(\boldsymbol{x})) + (1-\eta(\boldsymbol{x}))\eta(\boldsymbol{x})] \\
&= 2\mathbb{E}_{\boldsymbol{x}\sim\mathcal{P}_{\boldsymbol{x}}}[\eta(\boldsymbol{x})(1-\eta(\boldsymbol{x}))] \\
&\leq 2\mathbb{E}_{\boldsymbol{x}\sim\mathcal{P}_{\boldsymbol{x}}}[\min\{\eta(\boldsymbol{x}), (1-\eta(\boldsymbol{x}))\}] \\
&= 2R(f^*)
\end{aligned}$$

This kind of ML theory is not covered/required in this course!