CSCI567 Machine Learning (Fall 2018)

Prof. Haipeng Luo

U of Southern California

Aug 22, 2018

Outline

Outline

Enrollment

- There was some delay unfortunately. It is closing by the end of *today*.
- Two offerings: on-campus and DEN. You only need to *attend one*.
- Two sections: a lecture and a discussion section. You need to *attend both*. Discussion section is starting next week.

Teaching staff

- Instructors:

  Lecture: Haipeng Luo
  Discussion: Dr. Victor Adamchik and Dr. Kim Peters
  and a lot of help from Dr. Michael Shindler

- TAs:

  Chin-Cheng Hsu
  Shamim Samadi
  Chi Zhang
  Ke Zhang
  and more...

- More course producers and graders
- Office hours: TBA

Online platforms

A course website:
`http://www-bcf.usc.edu/~haipengl/courses/CSCI567/2018_fall`

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

Piazza: `https://piazza.com/usc/fall2018/20183csci567/home`

- main discussion forum
- everyone has to enroll

D2L: `https://courses.uscden.net/d2l/login`

- lecture videos
- submit written assignments
- grade posting

GitHub: `https://github.com/`

- submit programming assignments
- everyone needs to have a GitHub account

Required preparation

- Undergraduate courses in probability and statistics, linear algebra, multivariate calculus
- Programming: Python and necessary packages, `git`

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

Slides and readings

Lectures
Lecture slides will be posted before or soon after class.

Readings

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

Grade

- 15%: 5 written assignments
- 25%: 5 programming assignments
- 60%: 2 exams

Homework assignments

Five, each consists of

- problem set (3%)
  - submit one PDF to D2L (scan copy or typeset with LaTeX etc.)
  - graded based on effort
- programming tasks (5%)
  - submit through GitHub
  - graded by scripts

Policy

Collaboration:

- Allowed, but only at high-level
- Each has to make a separate submission
- State clearly who you collaborated with (or obtained help from)

Late submissions:

- A total of 5 grace days for the semester
- fill a form to apply within 24 hours of due time
- in place of "excused late" submissions, not in addition to
- no grace period
- late submissions without using grace days will be scored as 0
- your responsibility to keep track

Exams

- One midterm: *10/03, 5:00-7:00 PM*
- One final: *11/28, 5:00-7:00 PM*
- Location: TBA
- Individual effort, closed-book and closed-notes
- *Request for a different date/time must be submitted within first two weeks or asap in case of emergence*

Academic honesty and integrity

Plagiarism and other unacceptable violations

- Neither ethical nor in your self-interest
- Zero-tolerance

Teaching philosophy

The nature of this course

- Describe basic concepts and tools
- Describe algorithms and their development with intuition and rigor

Expectation on you

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

Feel free to interrupt and ask questions!

Important things for you to do

- Take a look at the course website
- Enroll in Piazza
- Get a GitHub account
- Brush up your Python skills

Questions?

Outline

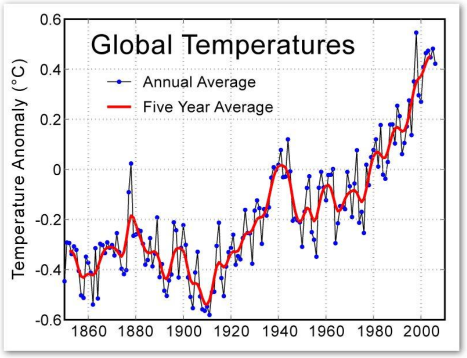
What is machine learning?

One possible definition[1]
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*

cf. Murphy's book

Example: detect patterns

How the temperature has been changing?

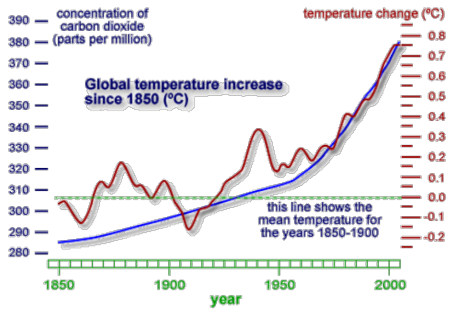


Patterns

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

How do we describe the pattern?

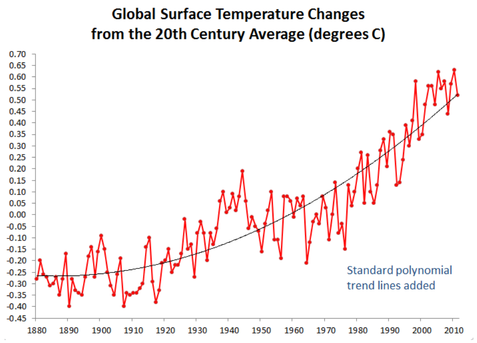
Build a model: fit the data with a polynomial function



- The model is not accurate for individual years
- But collectively, the model captures the major trend
- Still, not able to model the pattern of the *repeated up and down*

Predicting future

What is temperature of 2010?



- Again, the model is not accurate for that specific year
- But then, it is close to the actual one

What we have learned from this example?

Key ingredients in machine learning

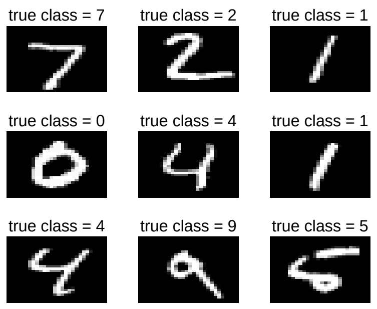
- Data
  collected from past observation (we often call them *training data*)
- Modeling
  devised to capture the patterns in the data
  - The model does not have to be true — "All models are wrong, but some are useful" by George Box.
- Prediction
  apply the model to forecast what is going to happen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

A rich history of applying statistical learning methods

Recognizing flowers (by R. Fisher, 1936)
Types of Iris: setosa, versicolor, and virginica

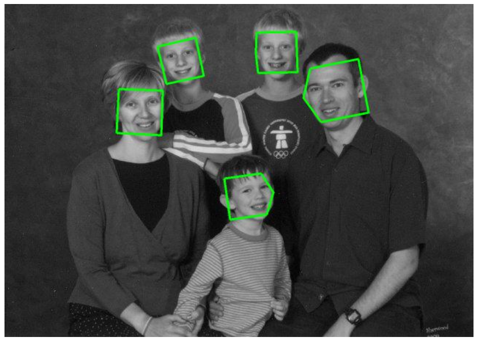
Huge success 20 years ago

Recognizing handwritten zipcodes (AT&T Labs, late 1990s)



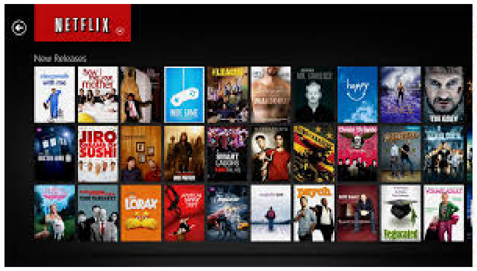
More modern ones, in your social life

Recognizing your friends on Facebook



It might be possible to know about you than yourself

Recommending what you might like



Why is machine learning so hot?

- **Tons of consumer applications**:
  - speech recognition, information retrieval and search, email and document classification, stock price prediction, object recognition, biometrics, etc
  - Highly desirable expertise from industry: Google, Facebook, Microsoft, Uber, Twitter, IBM, Linkedin, Amazon, $\cdots$
- **Enable scientific breakthrough**
  - Climate science: understand global warming cause and effect
  - Biology and genetics: identify disease-causing genes and gene networks
  - Social science: social network analysis; social media analysis
  - Business and finance: marketing, operation research
  - Emerging ones: healthcare, energy, $\cdots$

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 and latent patterns and explore data
- Reinforcement learning
  Aim to act optimally under uncertainty
- Many other paradigms

The focus and goal of this course

- Supervised learning (before midterm)
- Unsupervised learning (after midterm)

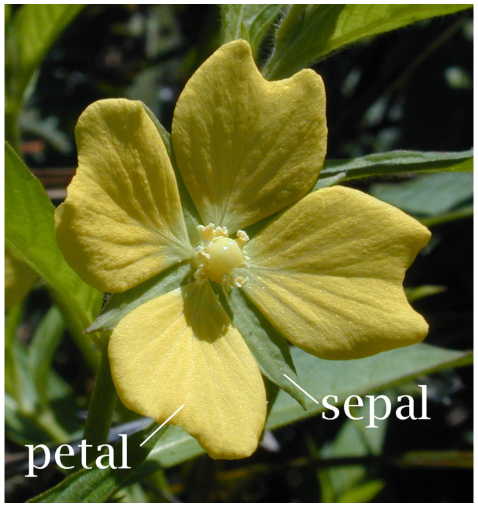
Outline

Recognizing flowers

Types of Iris: setosa, versicolor, and virginica

Measuring the properties of the flowers

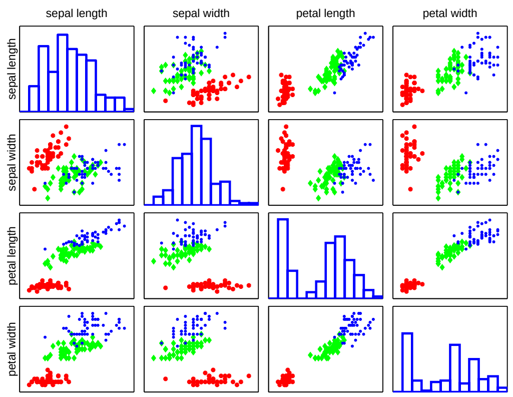
Features and attributes: the widths and lengths of sepal and petal



Pairwise scatter plots of 131 flower specimens

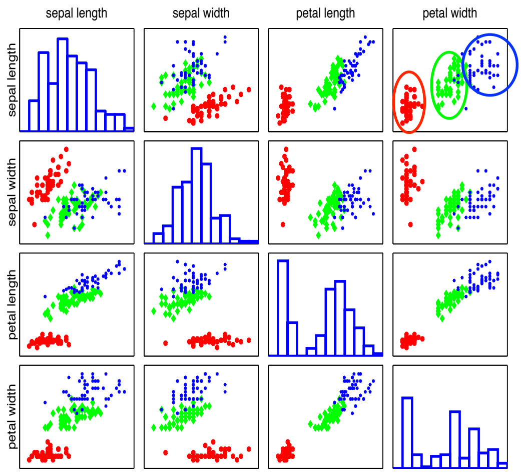
Visualization of data helps identify the right learning model to use

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



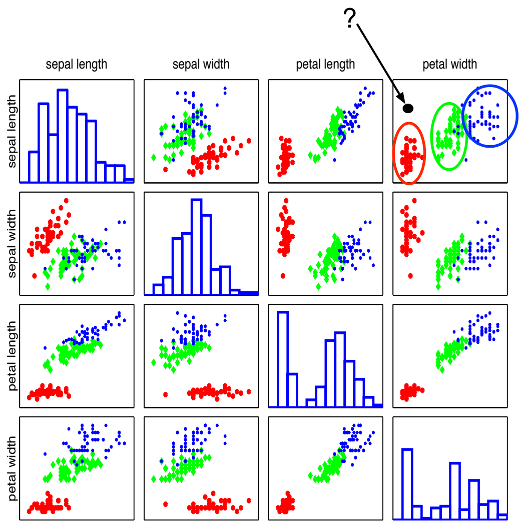
Different types seem well-clustered and separable

Using two features: petal width and sepal length



Labeling an unknown flower type

Closer to red cluster: so labeling it as 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 for learning $f : \mathbb{R}^\mathsf{D} \rightarrow [\mathsf{C}]$ for future prediction.

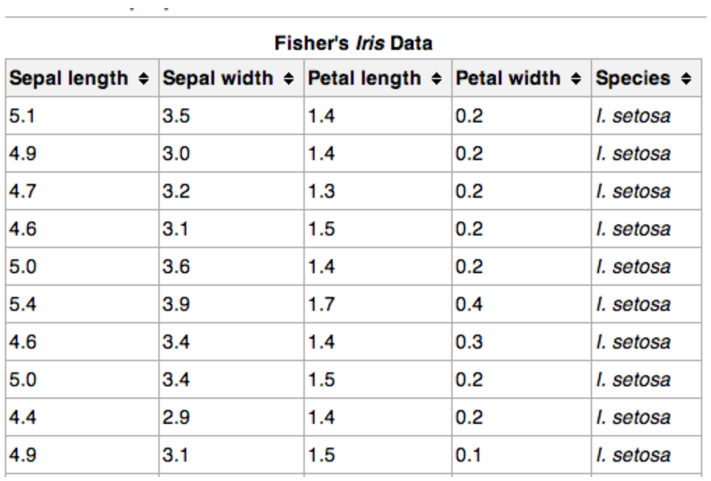
Special case: binary classification

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

Often, data is conveniently organized as a table

Ex: Iris data (click here for all data)

- 4 features
- 3 classes



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*

Nearest neighbor classification (NNC)

Nearest neighbor

$$\boldsymbol{x}(1) = \boldsymbol{x}_{\mathsf{nn}(\boldsymbol{x})}$$

where $\mathsf{nn}(\boldsymbol{x}) \in [\mathsf{N}] = \{1, 2, \cdots, \mathsf{N}\}$, i.e., the index to one of the training instances,

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

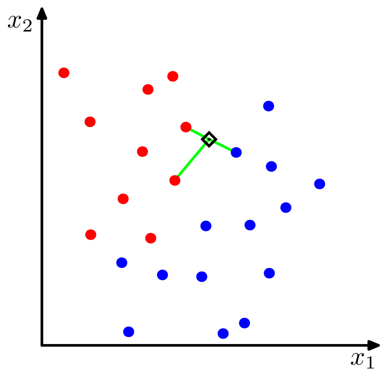
where $\|\cdot\|_2$ is the L_2/Euclidean distance.

Classification rule

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

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
⋮	⋮	⋮	

Flower with unknown category

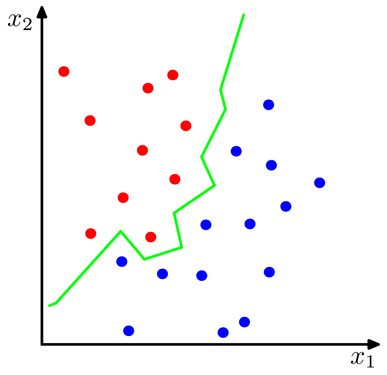
petal width = 1.8 and sepal width = 6.4 (i.e. $\boldsymbol{x} = (1.8, 6.4)$)
Calculating distance $\|\boldsymbol{x} - \boldsymbol{x}_n\|_2 = \sqrt{(x_1 - x_{n1})^2 + (x_2 - x_{n2})^2}$

ID	distance
1	1.75
2	0.72
3	0.76

Thus, the category 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

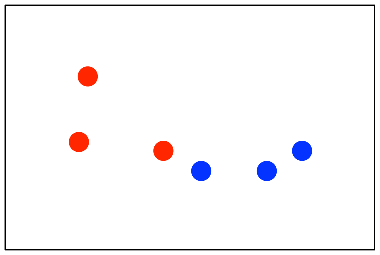
$$A^{\text{TRAIN}} = \frac{1}{\mathsf{N}} \sum_n \mathbb{I}[f(\boldsymbol{x}_n) == y_n], \quad \varepsilon^{\text{TRAIN}} = \frac{1}{\mathsf{N}} \sum_n \mathbb{I}[f(\boldsymbol{x}_n) \neq y_n]$$

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

$$\mathsf{nn}(\boldsymbol{x}) = \arg\min_{n \in [\mathsf{N}]} \|\boldsymbol{x} - \boldsymbol{x}_n\|_2$$

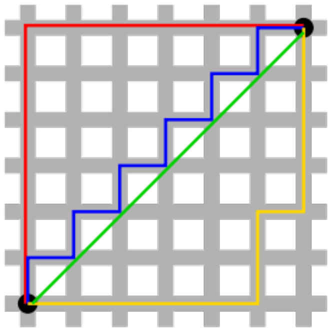
Many other alternative distances

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?

- 1-nearest neighbor: $\mathsf{nn}_1(\boldsymbol{x}) = \arg\min_{n \in [\mathsf{N}]} \|\boldsymbol{x} - \boldsymbol{x}_n\|_2$
- 2-nearest neighbor: $\mathsf{nn}_2(\boldsymbol{x}) = \arg\min_{n \in [\mathsf{N}] - \mathsf{nn}_1(\boldsymbol{x})} \|\boldsymbol{x} - \boldsymbol{x}_n\|_2$
- 3-nearest neighbor: $\mathsf{nn}_3(\boldsymbol{x}) = \arg\min_{n \in [\mathsf{N}] - \mathsf{nn}_1(\boldsymbol{x}) - \mathsf{nn}_2(\boldsymbol{x})} \|\boldsymbol{x} - \boldsymbol{x}_n\|_2$

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: with $\boldsymbol{x}(k) = \boldsymbol{x}_{\mathsf{nn}_k(\boldsymbol{x})}$, we have

$$\|\boldsymbol{x} - \boldsymbol{x}(1)\|_2^2 \leq \|\boldsymbol{x} - \boldsymbol{x}(2)\|_2^2 \cdots \leq \|\boldsymbol{x} - \boldsymbol{x}(K)\|_2^2$$

How to classify with K neighbors?

Classification rule

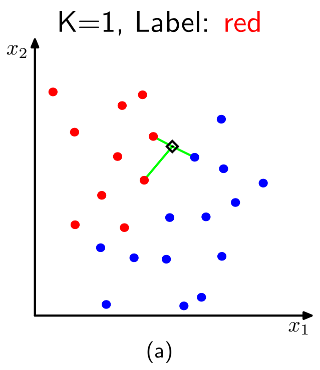
- Every neighbor votes: naturally $\boldsymbol{x}_n$ votes for its label y_n.
- Aggregate everyone's vote on a class label c

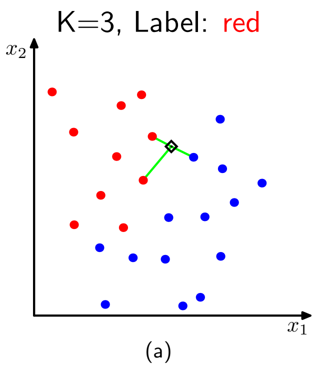
$$v_c = \sum_{n \in \text{knn}(\boldsymbol{x})} \mathbb{I}(y_n == c), \quad \forall \quad c \in [\mathsf{C}]$$

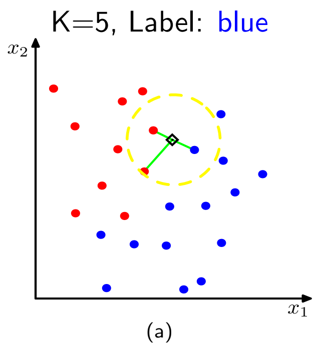
- Predict with the majority

$$y = f(\boldsymbol{x}) = \arg\max_{c \in [\mathsf{C}]} v_c$$

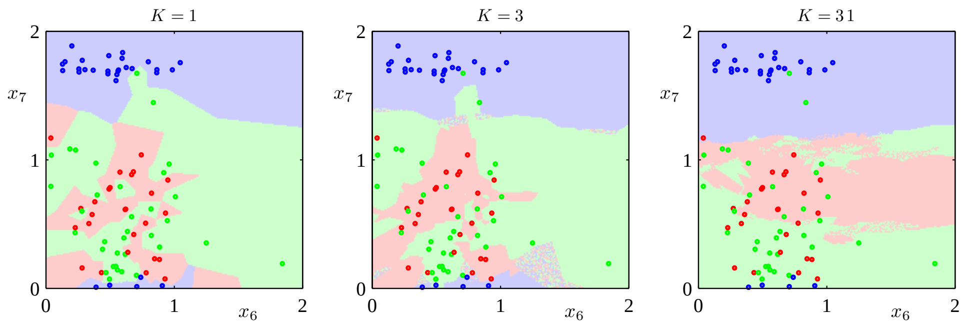
Example







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

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

- 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 development 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 for learning $f(\cdot)$

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 for assessing how well $f(\cdot)$ will do.

Development/Validation 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

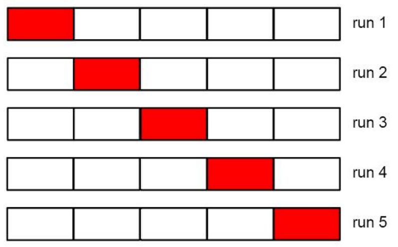
- For each possible value of the hyperparameter (e.g. $K = 1, 3, \cdots$)
  - Train a model using $\mathcal{D}^{\text{TRAIN}}$
  - Evaluate the performance of the model on $\mathcal{D}^{\text{DEV}}$
- Choose the model with the best performance on $\mathcal{D}^{\text{DEV}}$
- Evaluate the model on $\mathcal{D}^{\text{TEST}}$

S-fold Cross-validation

What if we do not have a development set?

- Split the training data into S equal parts.
- Use each part *in turn* as a development 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 $\mathcal{D}^{\text{train}}$**
- Evaluate the model on $\mathcal{D}^{\text{TEST}}$

Mini-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.

Mini-summary

Typical steps of developing a machine learning system:

- Collect data, split into training, development, 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 (x, y) (from $\mathcal{D}^{\text{TRAIN}}$, $\mathcal{D}^{\text{DEV}}$, or $\mathcal{D}^{\text{TEST}}$) is an *independent and identically distributed (i.i.d.)* sample of an unknown joint distribution $\mathcal{P}$.

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

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?

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

- 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 x, *knowing $\mathcal{P}(y|x)$*?

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

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

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

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

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

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)

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

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: $x(1) \to x$ with probability 1

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