

Guest lecture: statistical decision theory, k-Nearest Neighbors, and OCR

Miguel Morin, for Prof. Kamel Jedidi's class

24 November 2025

Summary of “Computing for Business Research”


- ▶ I teach B9122, Computing for Business Research, in the Fall
- ▶ To bridge the gap between:
 - ▶ programming and software engineering
 - ▶ for example, "write a computer program that sorts a vector of integers"
 - ▶ Statistics, regression models, and computational analytics
 - ▶ for example "create a model that predicts a stock price"

Today's lecture

- ▶ Write / run code like this in a shell / terminal / bash:

```
python exercise.py
```

- ▶ Write / run code like this in Python command: `a = list(1, 2, 3)`

- ▶ Answer polls “live” or at PollEverywhere: <https://pollev.com/b9122>
(with this sign )

- ▶ Download today's materials here:

<https://github.com/mm3509/guest-lecture>, or run this in
shell/terminal/bash:

```
cd  
git clone https://github.com/mm3509/guest-lecture  
cd guest-lecture
```

OCR (1)

Statistics

kNN

OCR (2)

■ OCR (1)

■ Statistics

■ kNN

■ OCR (2)

Table of Contents

■ OCR (1)

■ Statistics

■ kNN

■ OCR (2)
MNIST database
Demo




Problem statement: Optical Character Recognition (OCR)

- Definition: convert images (for humans) into text (characters, for computers)

text 123 →  → `text 123`

- What use cases do you see? 

Possible solutions for applications / use cases of OCR

- ▶  zip code mail sorting ([link](#))
- ▶  automatic fines for skipping a red light in Saudi Arabia; congestion charge and tolls (“Automated License Plate Recognition”, [link](#))
- ▶  speed limit recognition in modern cars ([link](#))

OCR: on hold for now

- ▶ We'll return to this example at the end
- ▶ For now: X = input (an image); Y = output (text)

text 123 →  → text 123

Table of Contents

■ OCR (1)

■ Statistics

■ kNN

■ OCR (2)
MNIST database
Demo

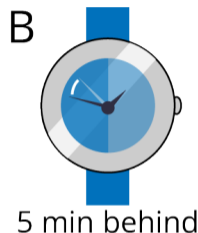
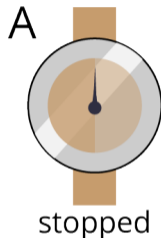
Statistical decision theory

- ▶ Section 2.4 of “The elements of statistical learning”, page 18 ([link to PDF](#))



- ▶ X : input, in \mathbb{R}^N (N real dimensions)
- ▶ Y : output, in \mathbb{R} (real line)
- ▶ $P(X, Y)$: joint probability distribution: **unknown!**
- ▶ $f(x)$ prediction function, that predicts y given x
- ▶ How to compare such functions, i.e. $f(x)$ better than $g(x)$?

Example with watches



A	B	y (true)
...
12am	2.15am	2.20am
12am	3.03am	3.08am
12am	4.12am	4.17am
12am	5.18am	5.23am
12am	7.20am	7.25am
...

- ▶ Both are wrong, but watch B is better. How much better? Can we quantify “wrongness” with a single number / summary statistic?
- ▶ Think, then answer the poll: 📊

Solution: Quantifying wrongness


- ▶ Wrongness of watches A (stopped) and B (5 min behind):
 - ▶ average discrepancy / absolute difference over a day: 3h vs. 5 min
 - ▶ average discrepancy / standard deviation, over a day: 3h28min vs. 5 min
- ▶ These are **loss functions**.

Hands-on exercise with watches

- ▶ Watch C: on average, 5 minutes behind (randomly, between 8 minutes and 2 minutes behind)
- ▶ Watch D: on average, 4 minutes behind (randomly, between 10 minutes behind and 2 minutes ahead)
- ▶ Which one is better (less wrong)?
- ▶ You will now find a statistic (a number) that quantifies / summarizes the “wrongness”



Your turn: find a statistic to summarize watches

- ▶ 1. In Python: open exercise 25 and complete the code to compute a summary statistic for each watch
- ▶ 2. Or, in a spreadsheet: open `watches.csv` and compute a summary statistic
- ▶ 3. Print or save that summary statistic for each watch
- ▶ 4. Fill the poll about which watch is better 

Solution to exercise 25

```
import statistics
n = len(true_time)
for watch in ["c", "d"]:
    measurement = time["watch_" + watch]
    assert n == len(measurement)

    diff = [measurement[i] - true_time[i] for i in range(n)]
    mad = statistics.mean([abs(d) for d in diff])
    rmse = math.sqrt(statistics.mean([d ** 2 for d in diff]))


    print(f"Watch {watch}:")
    print(f" Mean Absolute Deviation (MAD) = {mad:.2f}")
    print(f" root-Mean Squared Error (rMSE) = {rmse:.2f}")
```

Watches C and D: summary statistics

	Watch C	Watch D
Mean Absolute Deviation	5.03	4.67
root-Mean Squared Error	5.43	5.68

- ▶ Watch C: on average, 5 minutes behind (between -8 and -2): more bias, less variance
- ▶ Watch D: on average, 4 minutes behind (between -10 and +2): less bias, more variance
- ▶ if we consider absolute values: watch D is slightly better (less bias)
- ▶ if we consider mean-squared error, watch C is slightly better (less variance)

Squared loss (1)

- ▶ (speed check: )
- ▶ Most widely used loss: squared loss

$$L(y, f(x)) = (y - f(x))^2$$

y : output (time / character); x : input (time / an image); f : prediction function

- ▶ Very convenient:
 - ▶ differentiable everywhere (unlike absolute loss, $|y - f(x)|$)
 - ▶ solution (as we'll see later): conditional mean, easy to compute
- ▶ "Averaged over a day" = expected prediction error of f :

$$EPE(f) = \mathbb{E} \left[(Y - f(X))^2 \right] = \int [y - f(x)]^2 Pr(dx, dy)$$

Squared loss (2)

- Conditioning on X :

$$EPE(f) = \mathbb{E}_X \mathbb{E}_{Y|X} \left[(Y - f(X))^2 | X \right]$$

- We have no control over X , so the optimal solution is to minimize EPE pointwise (minimum of the sum is the sum of the minima):

$$\min \mathbb{E}_{Y|X} \left[(Y - f(x))^2 | X = x \right]$$

Optimal solution

$$\min \mathbb{E}_{Y|X} \left[(Y - f(x))^2 \mid X = x \right]$$

where $f(x)$ is our choice, or prediction for a given x

- To show that the solution is the conditional mean, $m = \mathbb{E}_{Y|X} Y$, insert it and distribute the square $((a + b)^2 = a^2 + 2ab + b^2)$:

$$\begin{aligned} (Y - f(x))^2 &= (Y - m + m - f(x))^2 \\ &= [(Y - m) + (m - f(x))]^2 \\ &= \underbrace{(Y - m)^2}_{a^2} + \underbrace{(m - f(x))^2}_{b^2} + \underbrace{2(Y - m)(m - f(x))}_{2ab} \end{aligned}$$

- Term a^2 : no control over irreducible error $\mathbb{E}_{Y|X} [a] = \mathbb{E}_{Y|X} [Y - \mathbb{E}_{Y|X} [Y]]^2$
- Term b^2 is minimal and zero at $f(x) = m = \mathbb{E}_{Y|X} [Y]$

Conditional mean

- ▶ (speed check: )
- ▶ Term ab is zero by the property of conditional expectations:

$$\begin{aligned}ab &= (Y - m)(m - f(x)) \\ \mathbb{E}_{Y|X}[c] &= \mathbb{E}_{Y|X}[(Y - m)(m - f(x))] \\ &= (\mathbb{E}_{Y|X}[Y - m])(m - f(x)) \\ &= (\mathbb{E}_{Y|X}[Y] - \mathbb{E}_{Y|X}[m]) \mathbb{E}_{Y|X}[m - f(x)] \\ &= 0\end{aligned}$$

- ▶ Squared loss \Rightarrow solution to the minimal prediction error (EPE) is the **conditional mean** $f(x) = m = \mathbb{E}_{Y|X}[Y]$

Summary of watches and math

Loss type (between true and predicted values)	Name	Optimal solution
Absolute	Mean Absolute Deviation (MAD)	Watch D (less bias)
	Absolute loss : $ y - f(x) $	Conditional median
Squared	Mean Squared Error (MSE)	Watch C (less variance)
	Squared loss: $(y - f(x))^2$	Conditional mean

- ▶ Together we did 3 of these cases (2 in Python/spreadsheet, one with math)
- ▶ The main point is: the choice of a loss implies an optimal solution

Table of Contents

■ OCR (1)

■ Statistics

■ **kNN**

■ OCR (2)
MNIST database
Demo

k -nearest neighbors

- ▶ From the conditional expectation:

$$\mathbb{E}(Y|X = x)$$

- ▶ Apply two approximations:
 - ▶ expectation $\mathbb{E} \rightarrow$ average over sample data Avg_i
 - ▶ conditioning on a point $X = x \rightarrow$ on a region $x_i \in N_k(x)$
- ▶ \Rightarrow k -nearest neighbors estimator:

$$\hat{f}(x) = Avg(y_i | x_i \in N_k(x)), \quad N_k(x) = k \text{ data points near } x$$

- ▶ Only parameter is k (hence the name)
- ▶ “Lazy learning”: no training, no data summarization. “The model is the data.”

Comparison with linear regression

- ▶ Alternative approximation: the conditional mean is linear:

$$f(x) \approx x^T \beta$$

- ▶ We can solve for β and find the usual “least-squares”:

$$\beta = [\mathbb{E}(XX^T)]^{-1} \mathbb{E}(XY)$$

- ▶ Both methods approximate conditional expectations by averages:
 - ▶ least squares: $f(x)$ is well approximated by a globally linear function
 - ▶ k -nearest neighbors: $f(x)$ is well approximated by a locally constant function

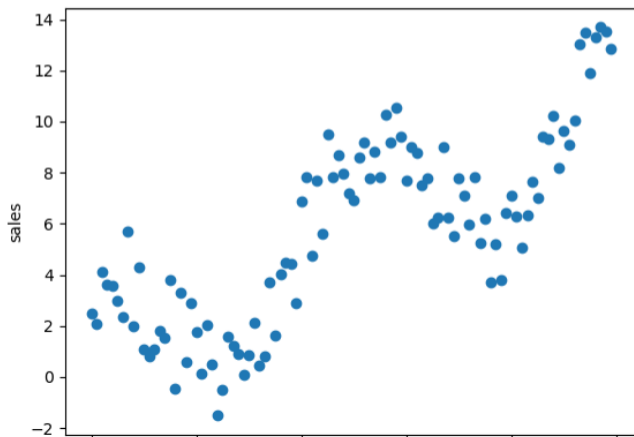
Basis for many methods

A large subset of the most popular techniques in use today are variants of these two simple procedures [(linear regression and k-nearest neighbors)]. (The elements of statistical learning, page 17)


For the rest of today, we will focus on kNN.

Application 1: time-series

Sales of a company by week: what patterns do you see?



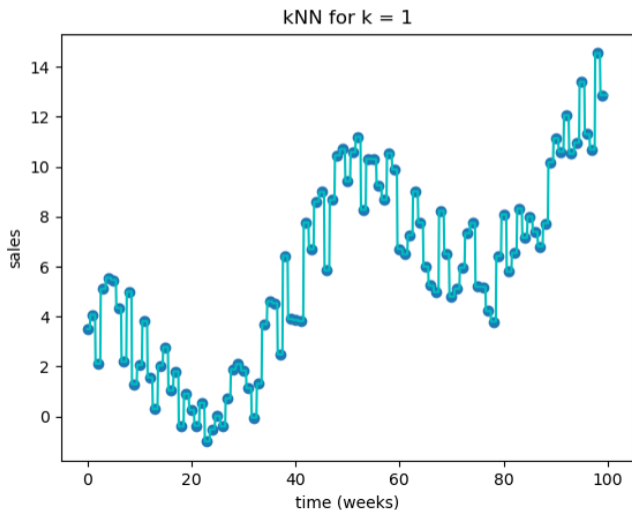
How to apply k-nearest neighbors (kNN) to time-series?

- ▶ X = time (in weeks), Y = sales (in dollars)
- ▶ \rightarrow we should be able to apply kNN to predict sales for a given week
- ▶ How can we apply kNN here? 

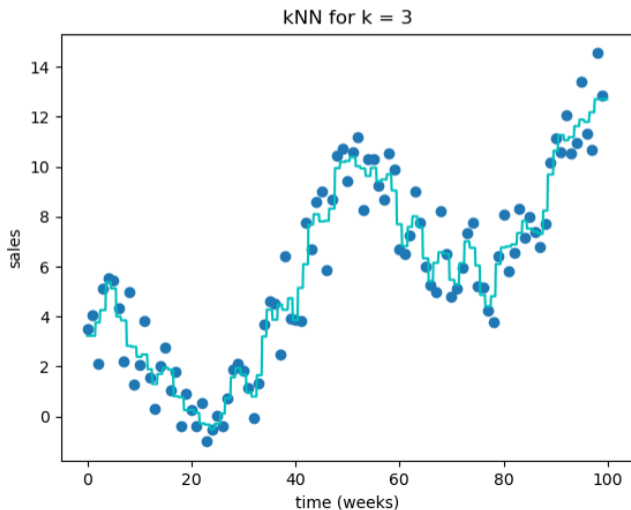
Solution to kNN on time-series data

- ▶ kNN only needs a distance on the inputs
- ▶ In the case of time-series, that distance is between the week we want to predict and the weeks we have data for
- ▶ We choose the k nearest weeks according to that distance
- ▶ And we average their sales

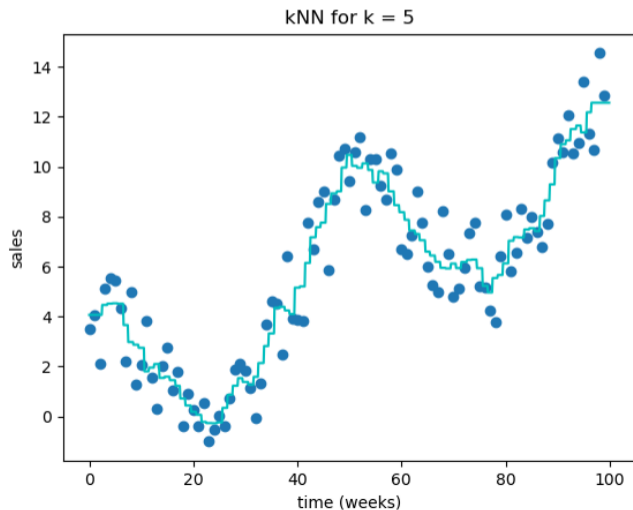
Results: $k = 1$, 1-nearest neighbor



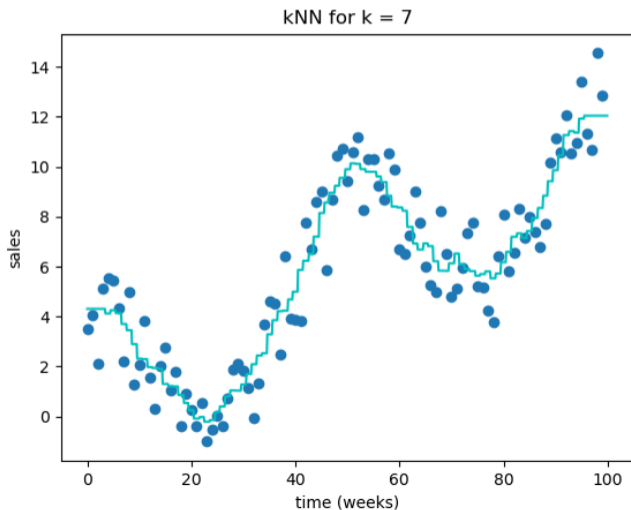
Results: $k = 3$, 3-nearest neighbors



Results: $k = 5$, 5-nearest neighbors



Results: $k = 7$, 7-nearest neighbors



Summary of k-Nearest Neighbors

- ▶ In time-series, kNN = moving average
- ▶ All kNN needs is some definition of distance on inputs (Euclidean distance in this case)

Table of Contents

■ OCR (1)

■ Statistics


■ kNN

■ OCR (2)
MNIST database
Demo

Application 2: OCR with MNIST database

- “Modified National Institute of Standards and Technology” database

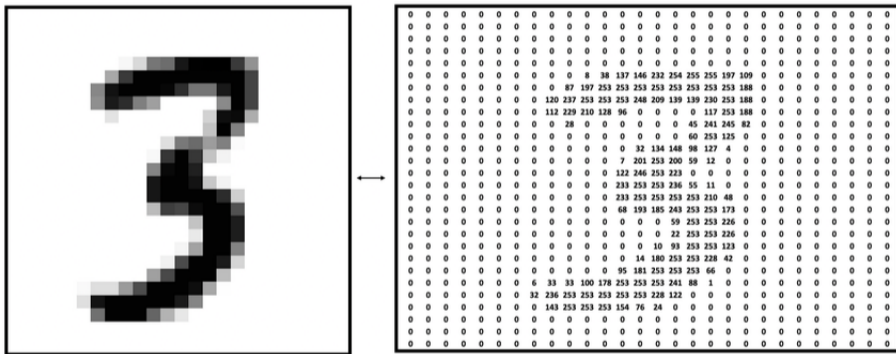


- 60k training images (X) and “true” labels (y)
- How to apply k -nearest neighbors to this problem? 



k -nearest neighbors on MNIST: “training”

- ▶ “Train” the “model:”



- ▶ image = matrix 28x28 of 8-bit integers (0-255)
- ▶ store all MNIST matrices and labels on disk (training data, 47 MB)
- ▶ choose $k = 1$ (for simplicity)

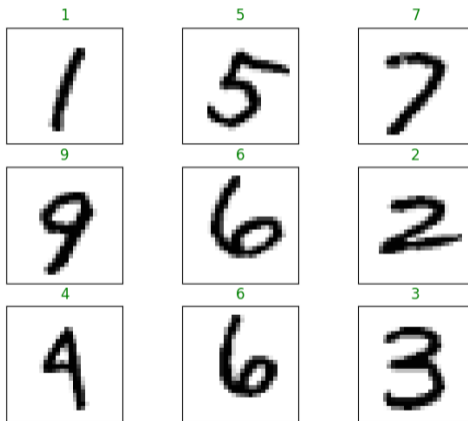
k -nearest neighbors on MNIST

- ▶ Assuming a scanned individual digit, use “model”:
 - ▶ resize and crop image, convert to greyscale pixel intensity, and to an integer
 - ▶ matrix M of size 28x28
 - ▶ compute Euclidean distance of M to each image in the training data:

$$d(M, \text{image}_i) = \sqrt{\sum_{h=1}^{28} \sum_{v=1}^{28} (M_{hv} - (\text{image}_i)_{hv})^2}$$

- ▶ find the “nearest neighbor”, with lowest distance among all 60k images (0.7s)
 - ▶ return the label of the nearest neighbor as the output

Demo: sample images



Data exploration

- ▶ The function `load_data()` from `mnist_ocr_plot.py` downloads data from the internet
- ▶ It returns a 4-element tuple: training X, training y, test X, test y
- ▶ training X and training y are **training** data, used to build / train an algorithm
- ▶ test X and test y are **test** data: they serve to test the algorithm on data it never encountered before
- ▶ all 4 variables are lists (or NumPy arrays):
 - ▶ for example X is a list of NumPy matrices with the images
- ▶ `training_y[i]` is the label for image `training_X[i]`

  Your turn: implement kNN and OCR

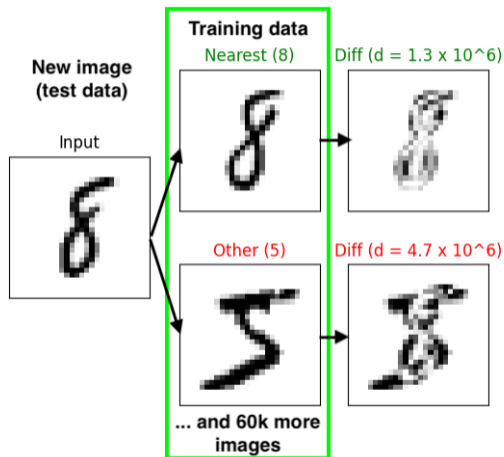
- ▶ 1. Open exercise 28
- ▶ 2. Install the requirements in a shell

```
pip install -r requirements.txt
```
- ▶ 3. Complete the 1-nearest neighbor function marked with `# TODO`
- ▶ 4. Run the file in a shell `python3 mnist_ocr.py` and see the sample images and predictions

Solution to exercise 28

```
def kNN1(x, training_X, training_y, verbose=False):  
    """Simple 1-nearest neighbor algorithm.  
    """  
  
    distances = [distance(x, training_X[i]) for i in range(len(training_X))]  
    nearest = numpy.argmin(distances)  
  
    return training_y[nearest], nearest
```

Demo: correct prediction



$\Rightarrow f(x) = 8$ (correct)

Rest of B9122 course

- ▶ Python (from novice to LeetCode medium-level)
- ▶ “Back-testing”
- ▶ Model selection (choosing the optimal k)
- ▶ Logistic regression
- ▶ Neural networks