

Lecture 02

Nearest Neighbor Methods

STAT 451: Intro to Machine Learning, Fall 2021

Sebastian Raschka

Lecture 2 (Nearest Neighbors)

Topics

- 1. Intro to nearest neighbor models**
2. Nearest neighbor decision boundary
3. K-nearest neighbors
4. Big-O & k-nearest neighbors runtime complexity
5. Improving k-nearest neighbors: modifications and hyperparameters
6. K-nearest neighbors in Python

Applications of Nearest Neighbor Methods



Saudi Computer Society, King Saud University
Applied Computing and Informatics

(<http://computer.org.sa>)
www.ksu.edu.sa
www.sciencedirect.com



ORIGINAL ARTICLE

Automated web usage data mining and recommendation system using K-Nearest Neighbor (KNN) classification method



D.A. Adeniyi, Z. Wei, Y. Yongquan *

The major problem of many on-line web sites is the presentation of many choices to the client at a time; this usually results to strenuous and time consuming task in finding the right product or information on the site. In this work, we present a study of automatic web usage data mining and recommendation system based on current user behavior through his/her click stream data on the newly developed Really Simple Syndication (RSS) reader website, in order to provide relevant information to the individual without explicitly asking for it. **The K-Nearest-Neighbor (KNN) classification** method has been trained to be used on-line and in Real-Time to identify clients/visitors click stream data, matching it to a particular user group and recommend a tailored browsing option that meet the need of the specific user at a particular time. [...]

Distance Metric Learning for Large Margin Nearest Neighbor Classification

Weinberger, Kilian Q., John Blitzer, and Lawrence K. Saul. "Distance metric learning for large margin nearest neighbor classification." *Advances in Neural Information Processing Systems*. 2006.

Kilian Q. Weinberger, John Blitzer and Lawrence K. Saul

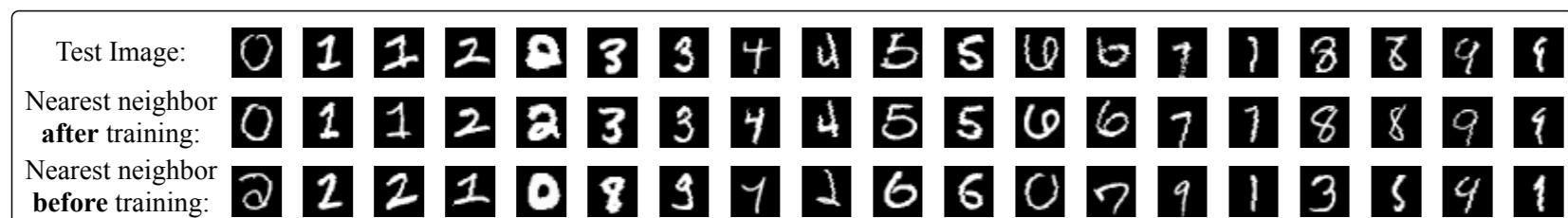
Department of Computer and Information Science, University of Pennsylvania

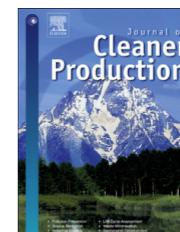
Levine Hall, 3330 Walnut Street, Philadelphia, PA 19104

{kilianw, blitzer, lsaul}@cis.upenn.edu



We show how to learn a Mahalanobis distance metric for k-nearest neighbor (kNN) classification by semidefinite programming. The metric is trained with the goal that the k-nearest neighbors always belong to the same class while examples from different classes are separated by a large margin. On seven data sets of varying size and difficulty, we find that metrics trained in this way lead to significant improvements in kNN classification—for example, **achieving a test error rate of 1.3% on the MNIST handwritten digits**. As in support vector machines (SVMs), the learning problem reduces to a convex optimization based on the hinge loss. Unlike learning in SVMs, however, our framework requires no modification or extension for problems in multiway (as opposed to bi- nary) classification.





Remaining useful life estimation of lithium-ion cells based on *k*-nearest neighbor regression with differential evolution optimization



Yapeng Zhou ^a, MiaoHua Huang ^{a,*}, Michael Pecht ^b

^a Hubei Key Laboratory of Advanced Technology for Automotive Components, Wuhan University of Technology, Wuhan, 430070, PR China

^b Center for Advanced Life Cycle Engineering, University of Maryland, College Park, MD, 20742, USA

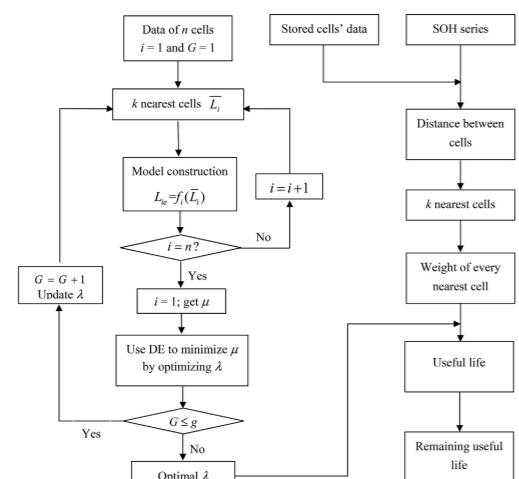


Fig. 1. Flowchart of parameter optimization and RUL estimation.



Fig. 2. Cell test bench.

different jellyroll configurations. All the cells have a LiCoO₂ cathode and graphite anode, and the electrolyte material contains LiPF₆, EC, and DEC, and the rated voltage is 3.7 V. The cathode and anode layers of groups A and B are wrapped around orthogonal rotation center. Cells were charged with constant current and constant voltage protocol and discharged with constant current to 2.7 V under 24 °C. The detailed specifications and charge/discharge method of these cells are shown in Table 1. As shown in Table 1, cells of group B were discharged with constant current of 0.5C, and a rate of x C is a current equal of multiplying x and the rated capacity. Groups A and B are used to validate the feasibility and online applicability of this method, respectively.

The detailed experiment procedure is as follows:

1. Program the charge/discharge with Bits Pro software on computer.
2. Connect the cells to the circuit, and put them into the thermal chamber.
3. Turn on the thermal chamber and set the temperature at 24 °C, and rest 1 h.
4. Start the charge/discharge cycling with the Bits Pro software.
5. Terminate the cycling when the SOH reaches 80%.

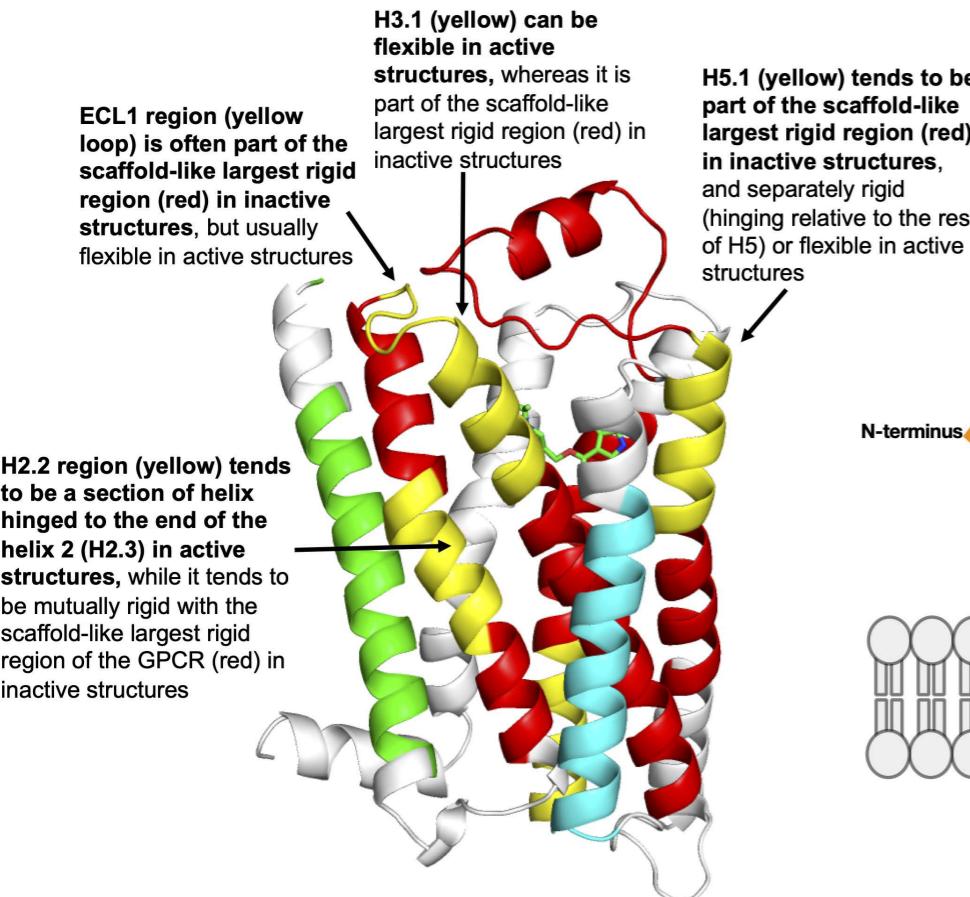
Note that there is an interval of 5 min between each charge and discharge. The capacity was calculated by integrating the discharge

Remaining useful life estimation is of great importance to customers who use battery-powered products. This paper develops a remaining useful life estimation model based on **k-nearest neighbor regression** by incorporating data from all the cells in a battery pack. A differential evolution technique is employed to optimize the parameters in the estimation model. In this approach, remaining useful life is estimated from a weighted average of the useful life of several nearest cells that share a similar degradation trend to the cell whose remaining useful life needs to be estimated. The developed method obtains a remaining useful life estimation result with average error of 9 cycles, and the best estimation only has an error of 2 cycles. [...]

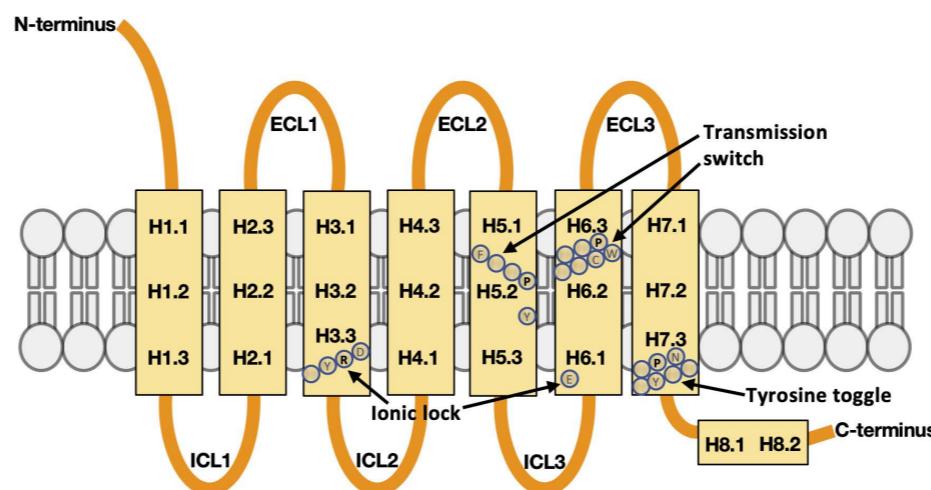
Article

Machine Learning to Identify Flexibility Signatures of Class A GPCR Inhibition

Joseph Bemister-Buffington¹, Alex J. Wolf¹, Sebastian Raschka^{1,2,*}  and Leslie A. Kuhn^{1,3,*} 



We show that machine learning can pinpoint features distinguishing inactive from active states in proteins, in particular identifying key ligand binding site flexibility transitions in GPCRs that are triggered by biologically active ligands. [...] However, considering the flexible versus rigid state identified by graph-theoretic ProFlex rigidity analysis for each helix and loop segment with the ligand removed, followed by feature selection and **k-nearest neighbor classification**, was sufficient to identify four segments surrounding the ligand binding site whose flexibility/rigidity accurately predicts whether a GPCR is in an active or inactive state ...

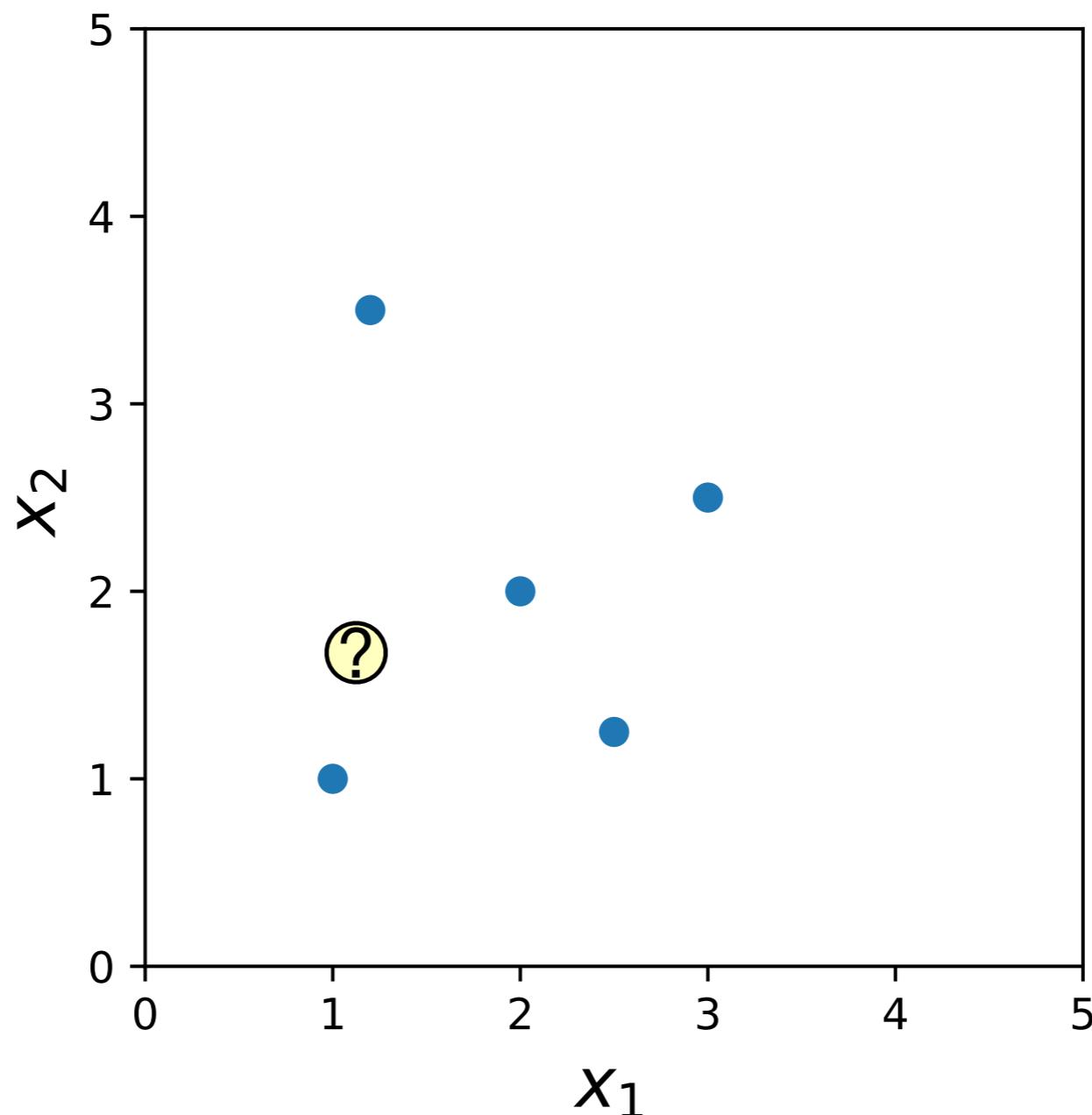


Joe Bemister-Buffington, Alex J. Wolf, Sebastian Raschka, and Leslie A. Kuhn (2020)
Machine Learning to Identify Flexibility Signatures of Class A GPCR Inhibition
Biomolecules 2020, 10, 454.

1-Nearest Neighbor

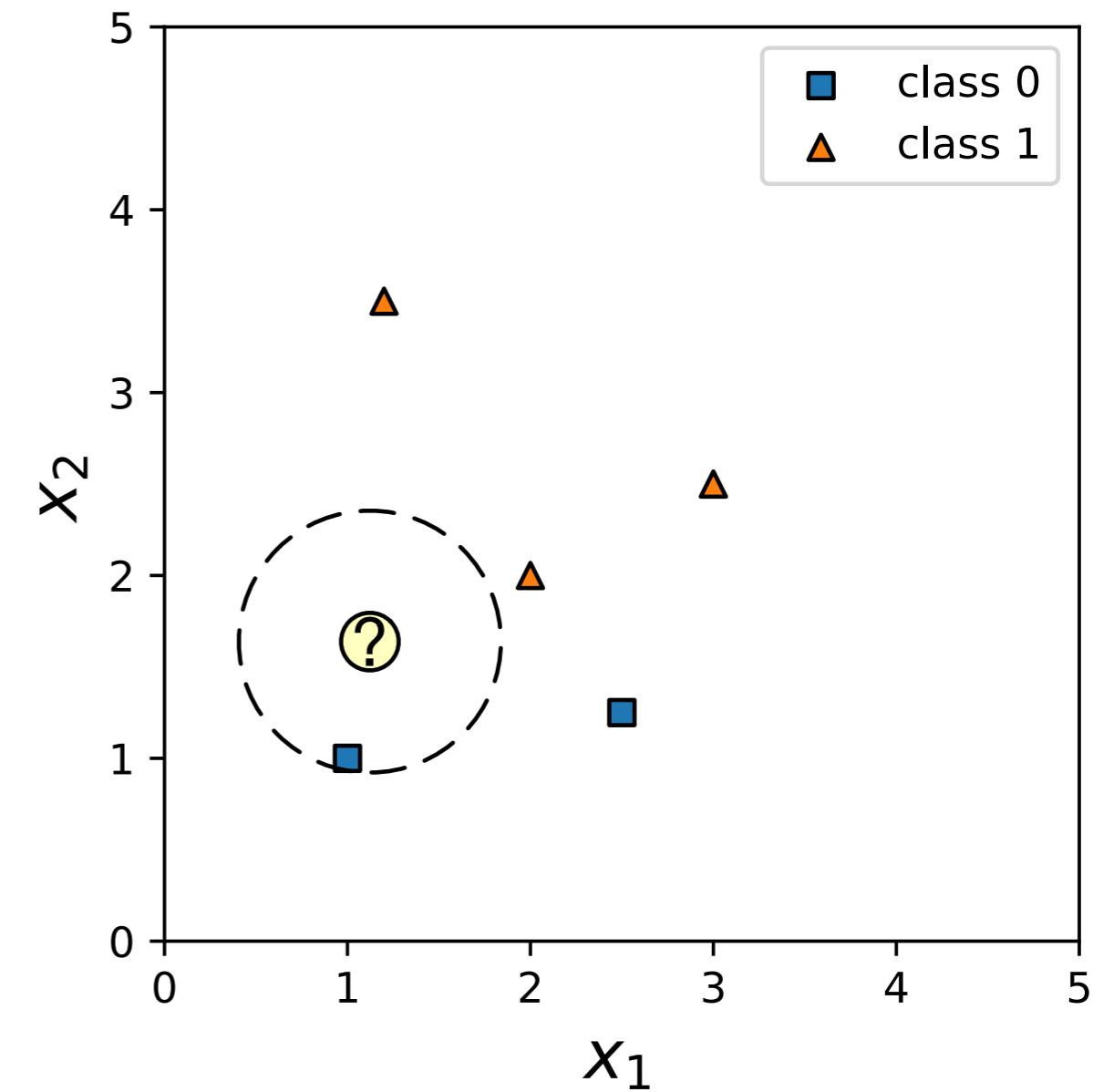
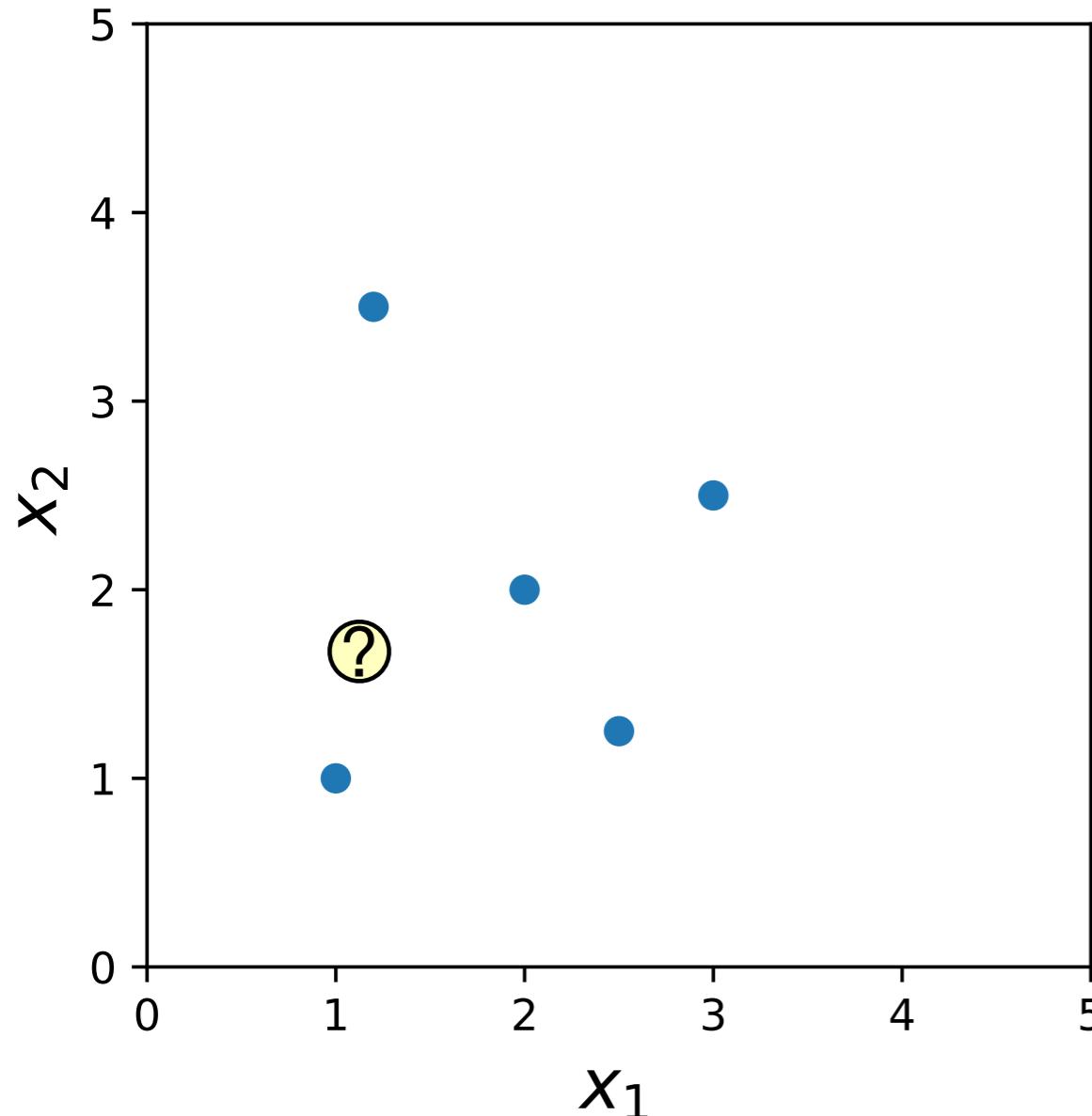
1-Nearest Neighbor

Task: predict the target / label of a new data point



1-Nearest Neighbor

Task: predict the target / label of a new data point



How? Look at most "similar" data point in training set

1-Nearest Neighbor Training Step

$$\langle \mathbf{x}^{[i]}, y^{[i]} \rangle \in \mathcal{D} \quad (|\mathcal{D}| = n)$$

How do we "train" the 1-NN model?

1-Nearest Neighbor Training Step

$$\langle \mathbf{x}^{[i]}, y^{[i]} \rangle \in \mathcal{D} \quad (|\mathcal{D}| = n)$$

To train the 1-NN model, we simply "remember" the training dataset

1-Nearest Neighbor Prediction Step

Given: $\langle \mathbf{x}^{[i]}, y^{[i]} \rangle \in \mathcal{D}$ ($|\mathcal{D}| = n$)

$\langle \mathbf{x}^{[q]}, \text{???} \rangle$

Predict: $f(\mathbf{x}^{[q]})$

closest_point := None

Algorithm:

closest_distance := ∞

query point

- for $i = 1, \dots, n$:
 - current_distance := $d(\mathbf{x}^{[i]}, \mathbf{x}^{[q]})$
 - if $\text{current_distance} < \text{closest_distance}$:
 - closest_distance := current_distance
 - closest_point := $\mathbf{x}^{[i]}$
- return $f(\text{closest_point})$

Commonly used: Euclidean Distance (L^2)

$$d(\mathbf{x}^{[a]}, \mathbf{x}^{[b]}) = \sqrt{\sum_{j=1}^m (x_j^{[a]} - x_j^{[b]})^2}$$

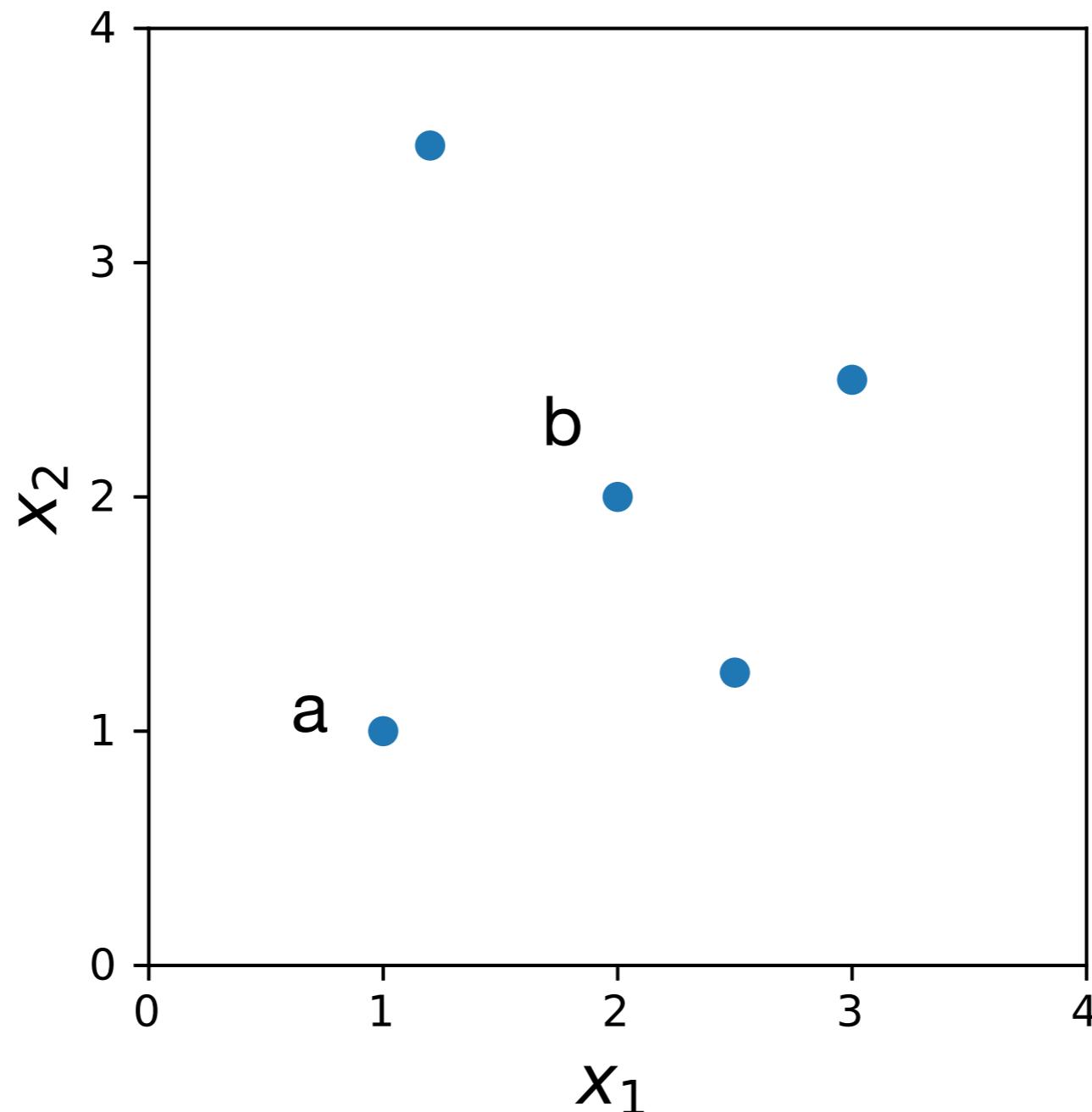
Lecture 2 (Nearest Neighbors)

Topics

1. Intro to nearest neighbor models
- 2. Nearest neighbor decision boundary**
3. K-nearest neighbors
4. Big-O & k-nearest neighbors runtime complexity
5. Improving k-nearest neighbors: modifications
and hyperparameters
6. K-nearest neighbors in Python

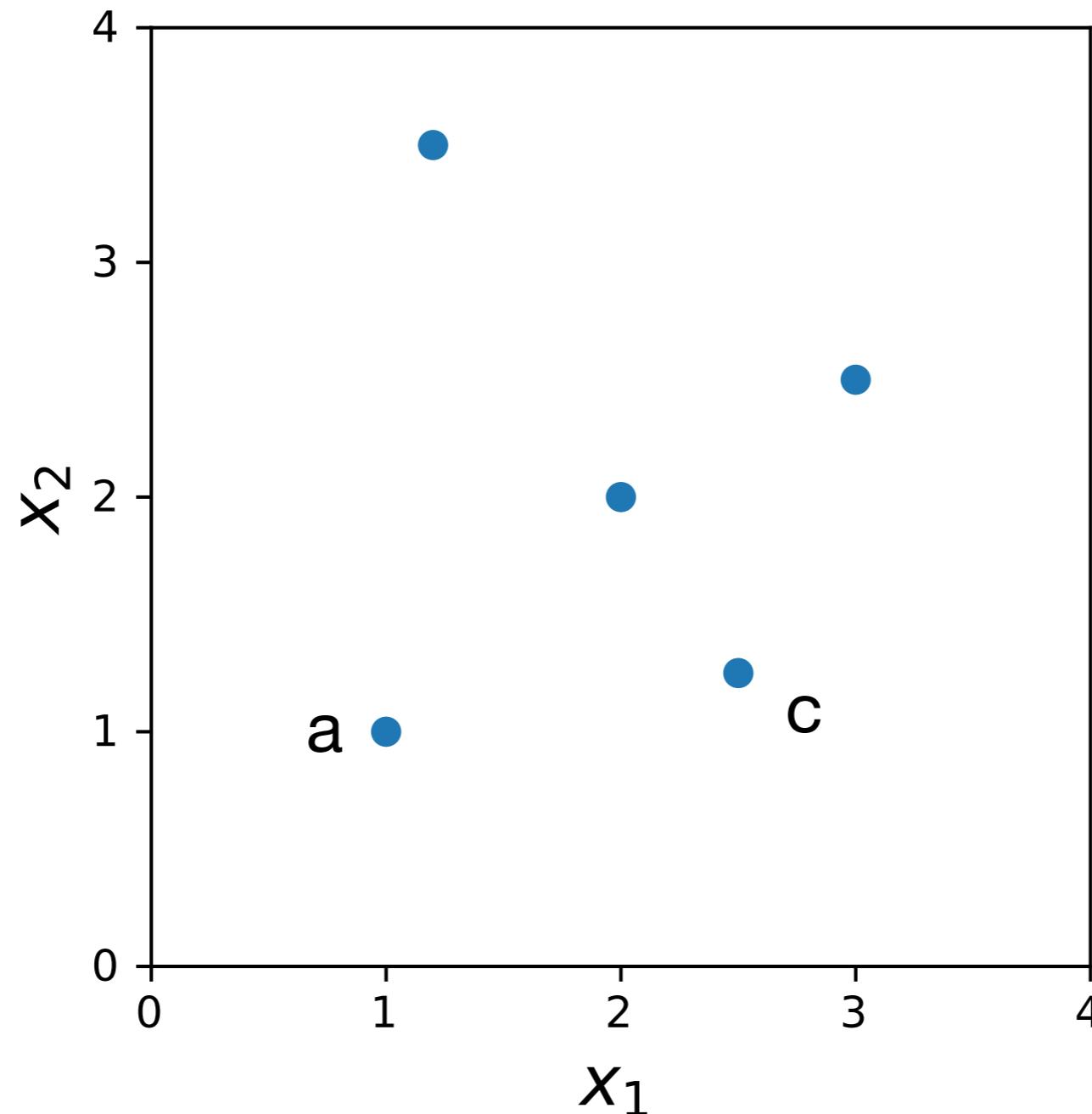
Nearest Neighbor Decision Boundary

Decision Boundary Between (a) and (b)



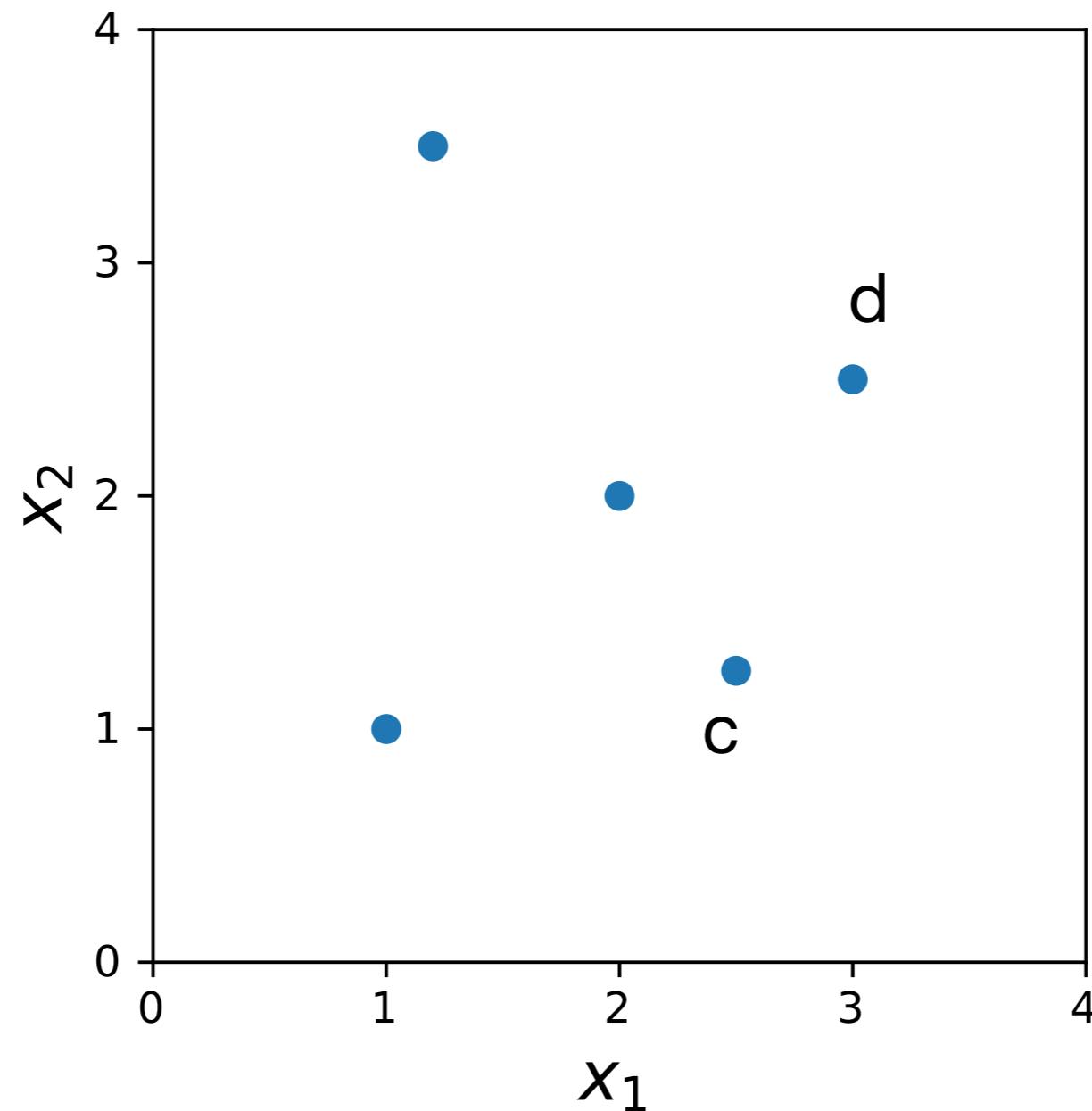
How does it look like?

Decision Boundary Between (a) and (c)

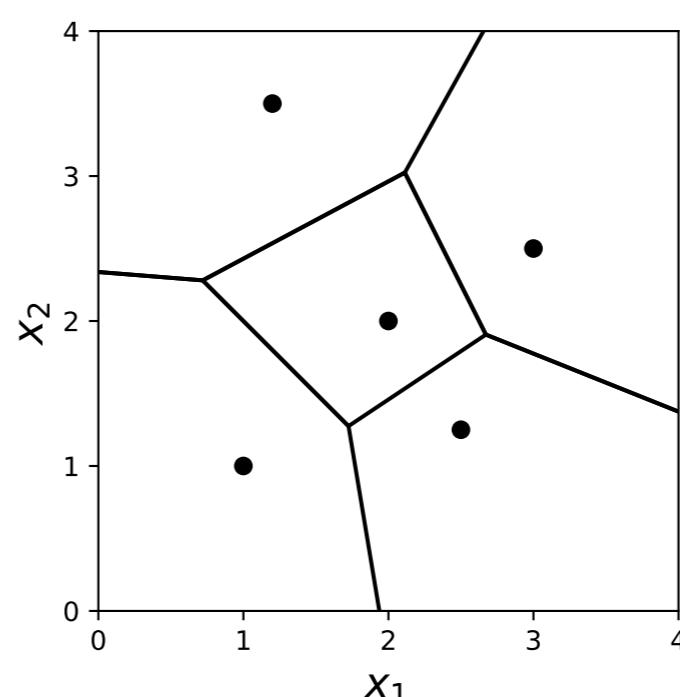
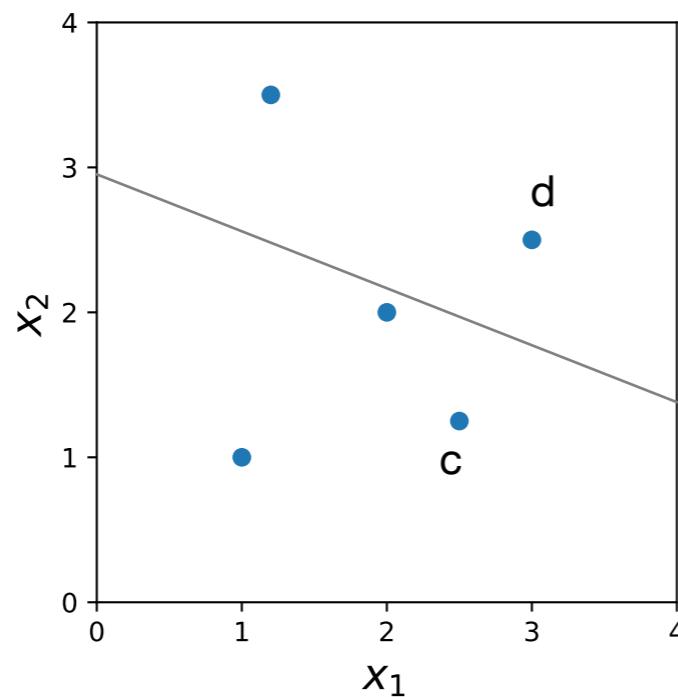
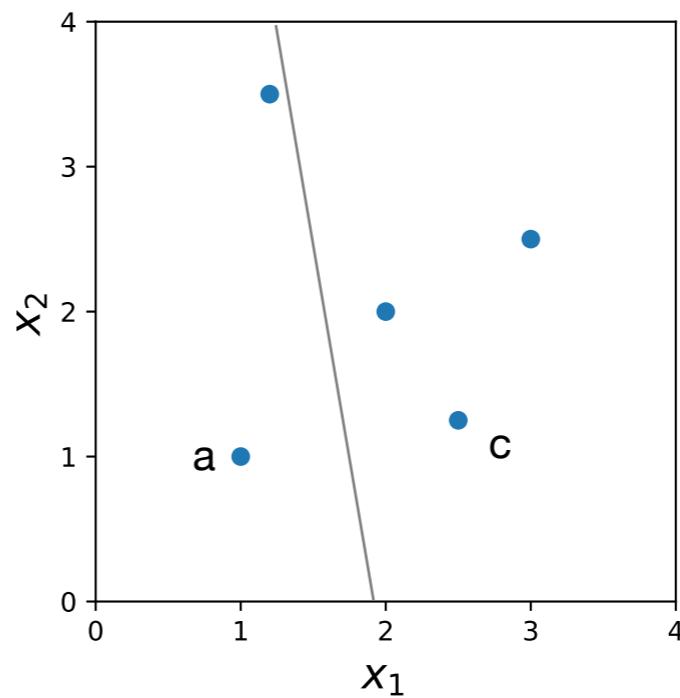
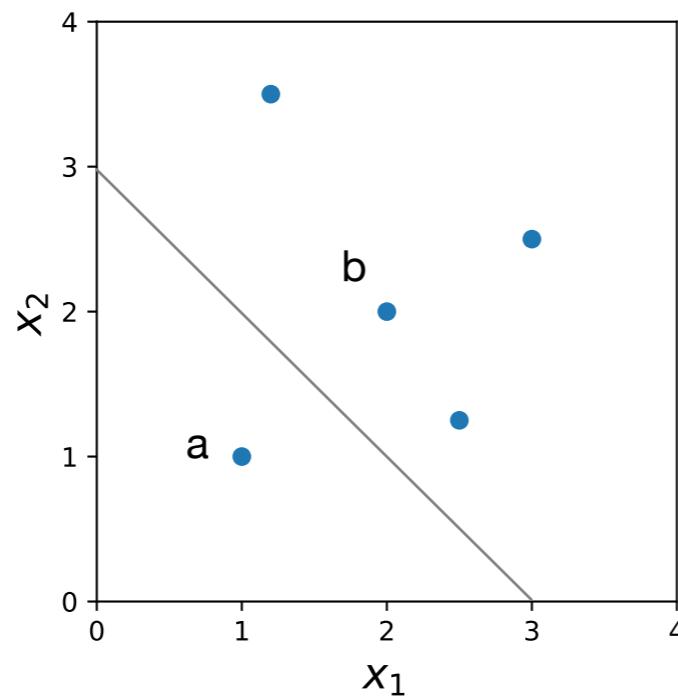


How does it look like?

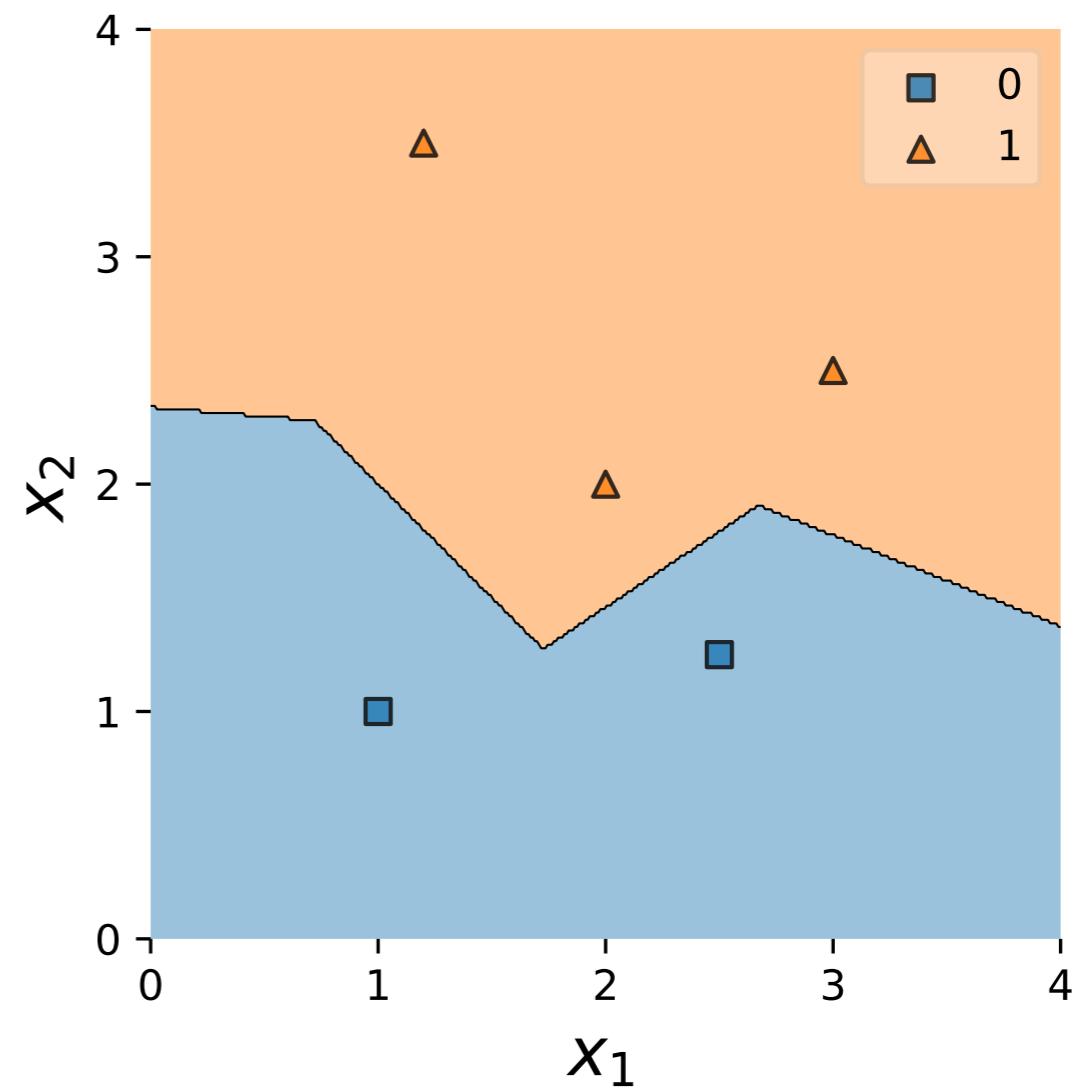
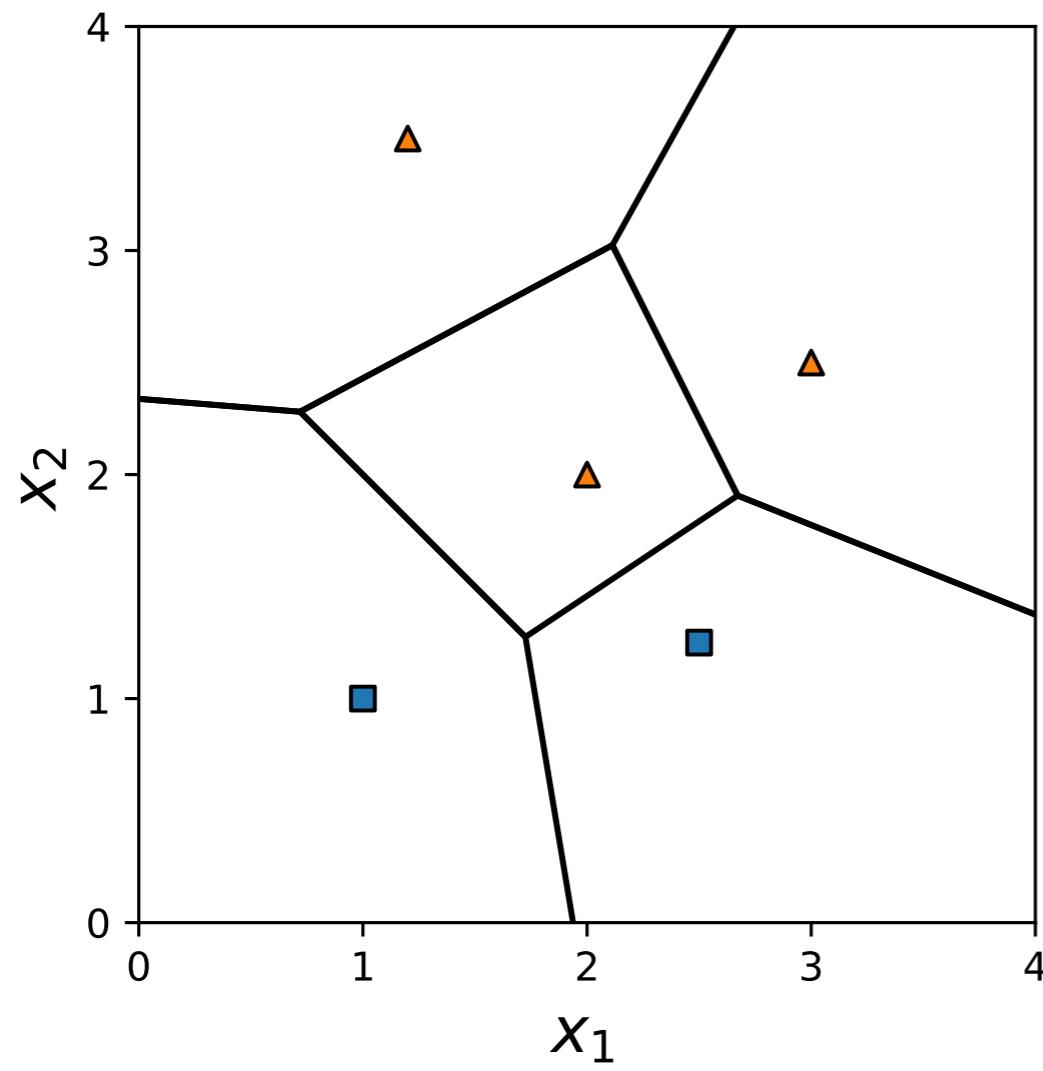
Decision Boundary Between (a) and (c)



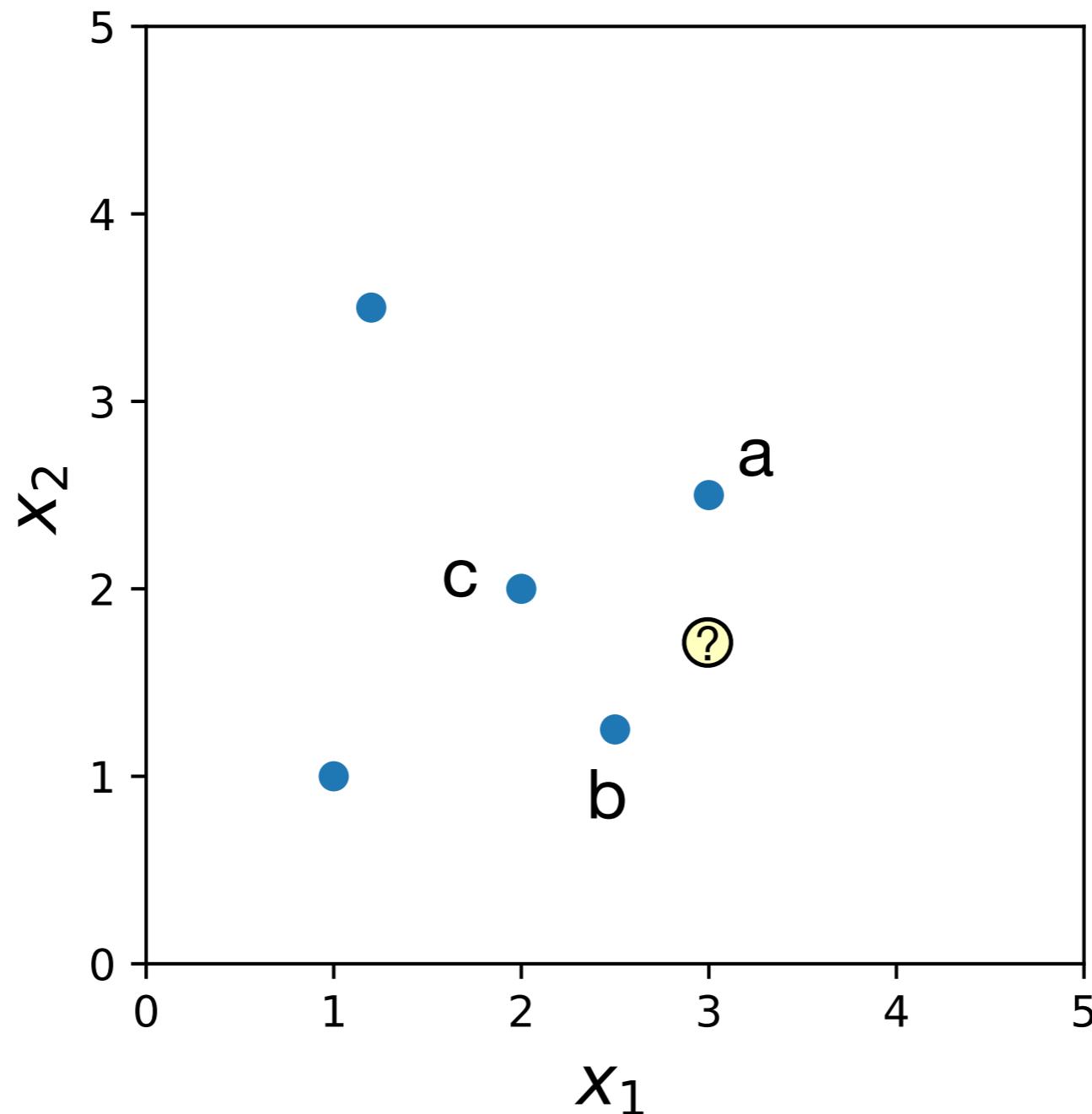
Decision Boundary of 1-NN



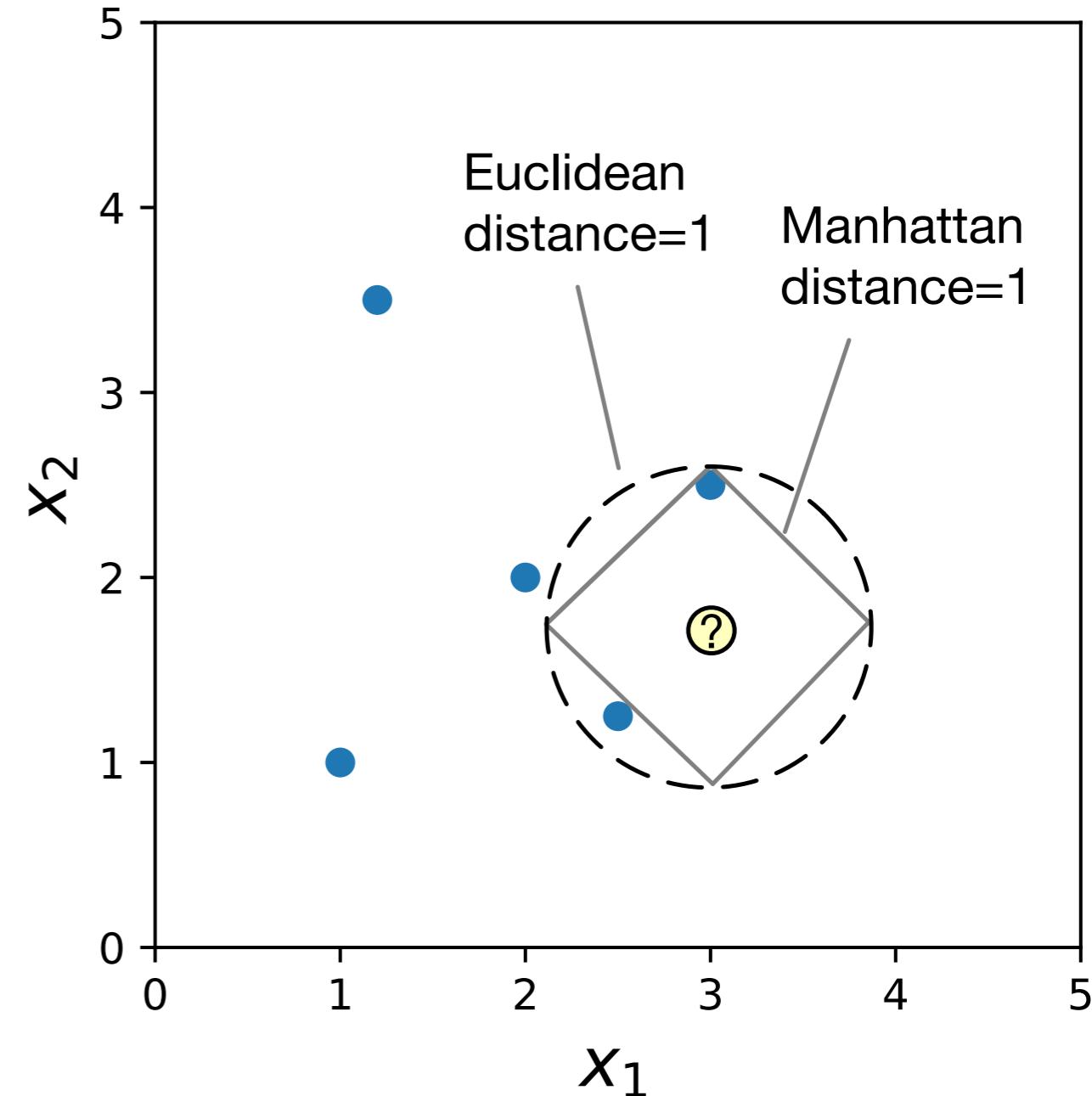
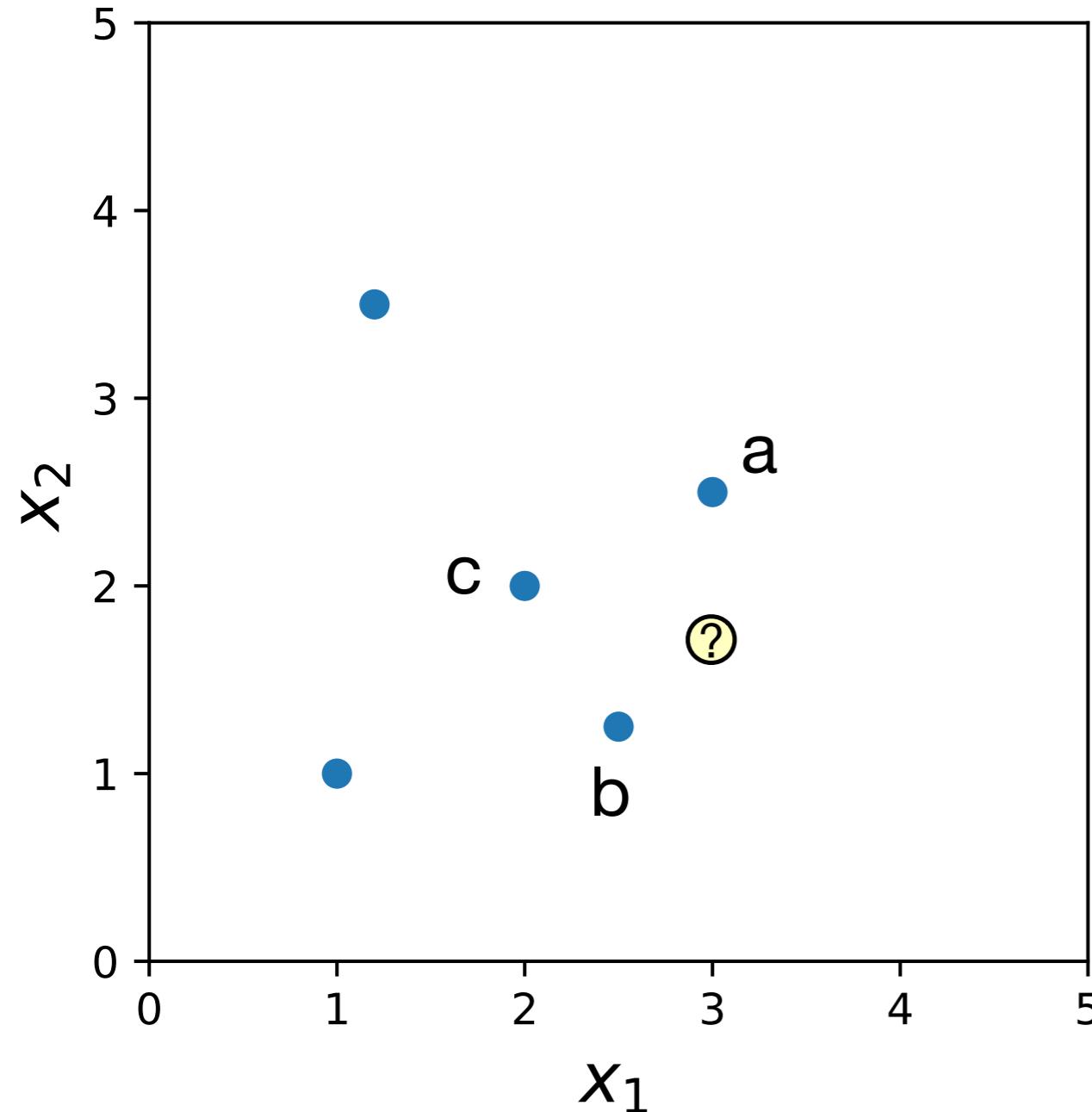
Decision Boundary of 1-NN



Which Point is Closest to ? ?



Depends on the Distance Measure!



Some Common Continuous Distance Measures

Euclidean

Manhattan

$$\text{Minkowski: } d(\mathbf{x}^{[a]}, \mathbf{x}^{[b]}) = \left[\sum_{j=1}^m \left(|x_j^{[a]} - x_j^{[b]}| \right)^p \right]^{\frac{1}{p}}$$

Mahalanobis

Cosine similarity

...

Minkowski style metrics

- euclidean
- manhattan
- chebyshev
- minkowski

Miscellaneous spatial metrics

- canberra
- braycurtis
- haversine

Normalized spatial metrics

- mahalanobis
- wminkowski
- seuclidean

Angular and correlation metrics

- cosine
- dot
- correlation
- spearmanr
- tsss
- true-angular

Probability metrics

- hellinger
- wasserstein

Metrics for binary data

- hamming
- jaccard
- dice
- russelrao
- kulsinski
- rogerstanimoto
- sokalmichener
- sokalsneath
- yule

Source: <https://github.com/lmcinnes/pynndescent>

Some Discrete Distance Measures

Hamming distance: $d(\mathbf{x}^{[a]}, \mathbf{x}^{[b]}) = \sum_{j=1}^m |x_j^{[a]} - x_j^{[b]}|$ where $x_j \in \{0,1\}$

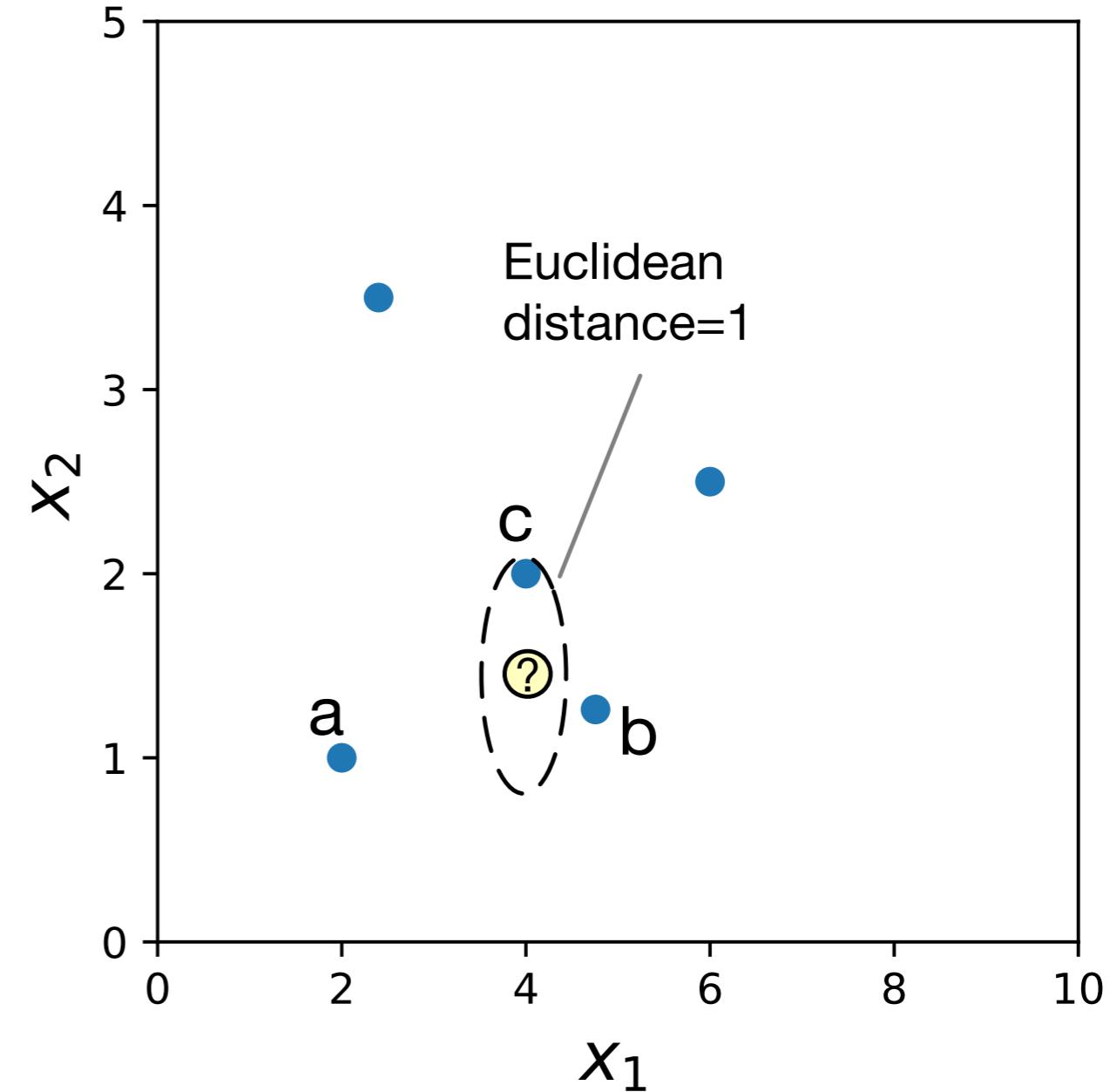
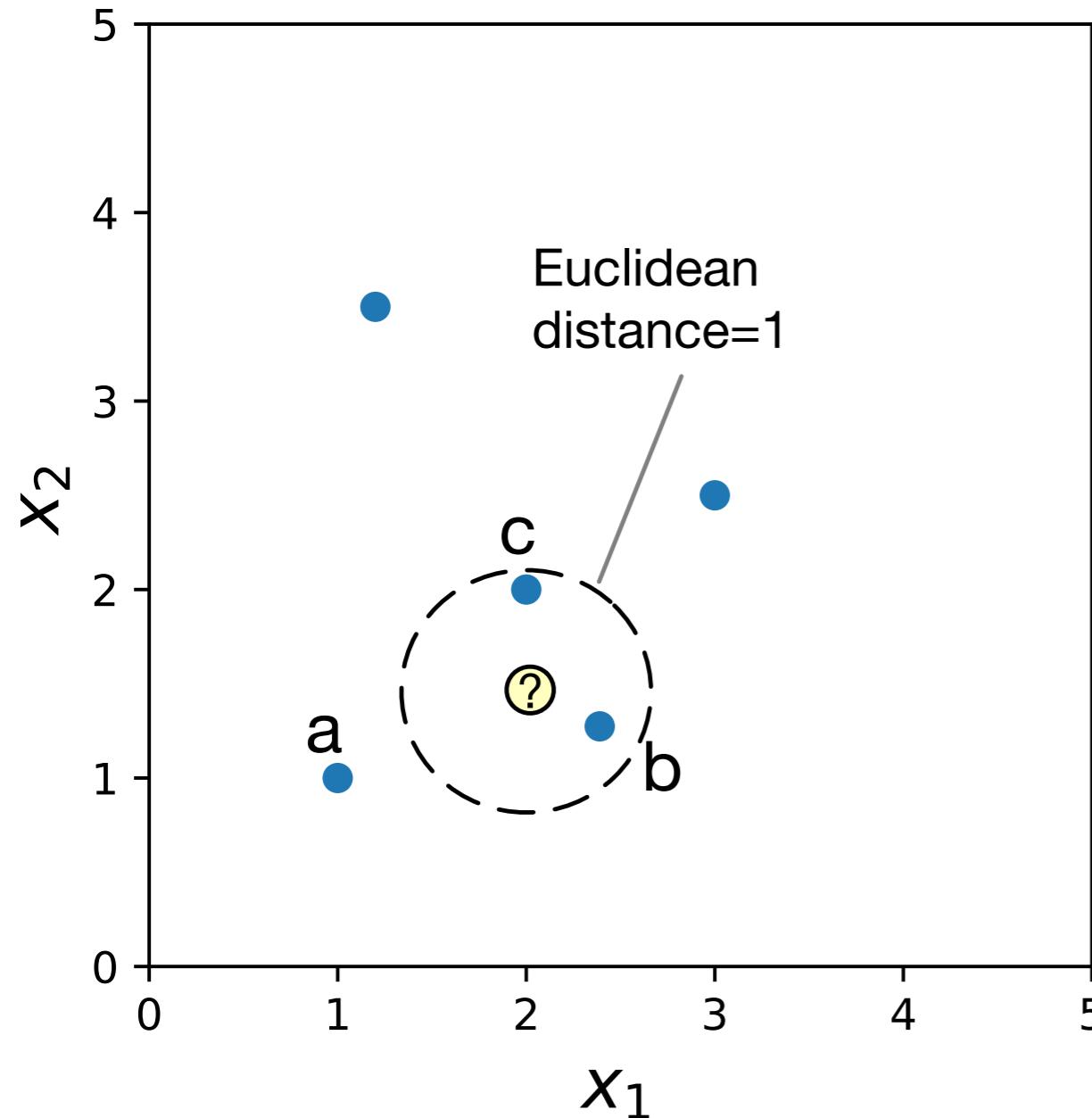
Jaccard/Tanimoto similarity:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

Dice: $D(A, B) = \frac{2|A \cap B|}{|A| + |B|}$

...

Feature Scaling

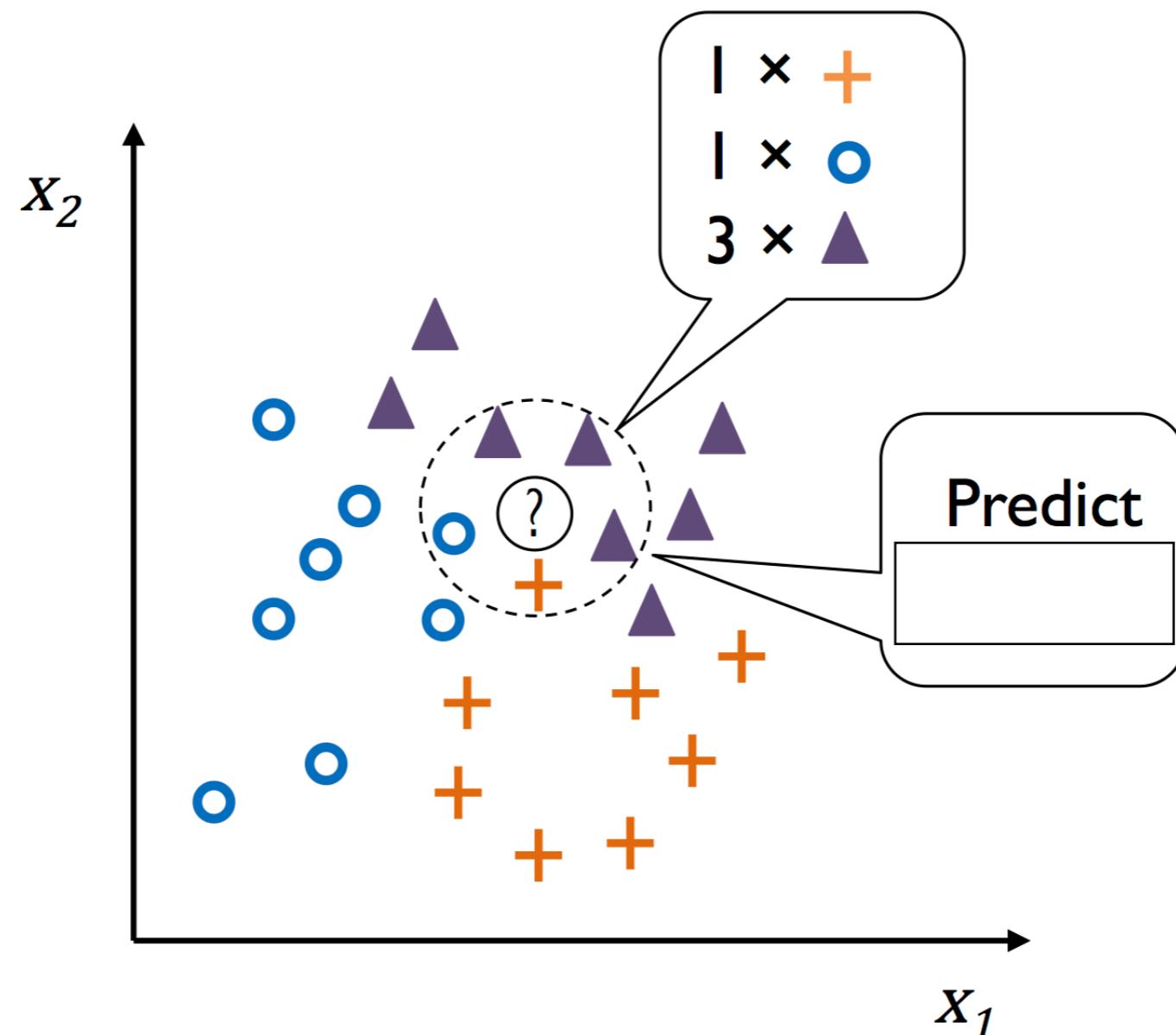


Lecture 2 (Nearest Neighbors)

Topics

1. Intro to nearest neighbor models
2. Nearest neighbor decision boundary
- 3. K-nearest neighbors**
4. Big-O & k-nearest neighbors runtime complexity
5. Improving k-nearest neighbors: modifications and hyperparameters
6. K-nearest neighbors in Python

k -Nearest Neighbors



A

y:



Majority vote:



Plurality Vote:



B

y:



Majority vote: None

Plurality Vote: 

k NN for Classification

$$\mathcal{D}_k = \{\langle \mathbf{x}^{[1]}, f(\mathbf{x}^{[1]}) \rangle, \dots, \langle \mathbf{x}^{[k]}, f(\mathbf{x}^{[k]}) \rangle\} \quad \mathcal{D}_k \subseteq \mathcal{D}$$

$$h(\mathbf{x}^{[q]}) = \arg \max_{y \in \{1, \dots, t\}} \sum_{i=1}^k \delta(y, f(\mathbf{x}^{[i]}))$$
$$\delta(a, b) = \begin{cases} 1, & \text{if } a = b, \\ 0, & \text{if } a \neq b. \end{cases}$$

$$h(\mathbf{x}^{[t]}) = \mathbf{mode}\left(\{f(\mathbf{x}^{[1]}), \dots, f(\mathbf{x}^{[k]})\}\right)$$

kNN for Regression

$$\mathcal{D}_k = \{\langle \mathbf{x}^{[1]}, f(\mathbf{x}^{[1]}) \rangle, \dots, \langle \mathbf{x}^{[k]}, f(\mathbf{x}^{[k]}) \rangle\} \quad \mathcal{D}_k \subseteq \mathcal{D}$$

$$h(\mathbf{x}^{[t]}) = \frac{1}{k} \sum_{i=1}^k f(\mathbf{x}^{[i]})$$

Lecture 2 (Nearest Neighbors)

Topics

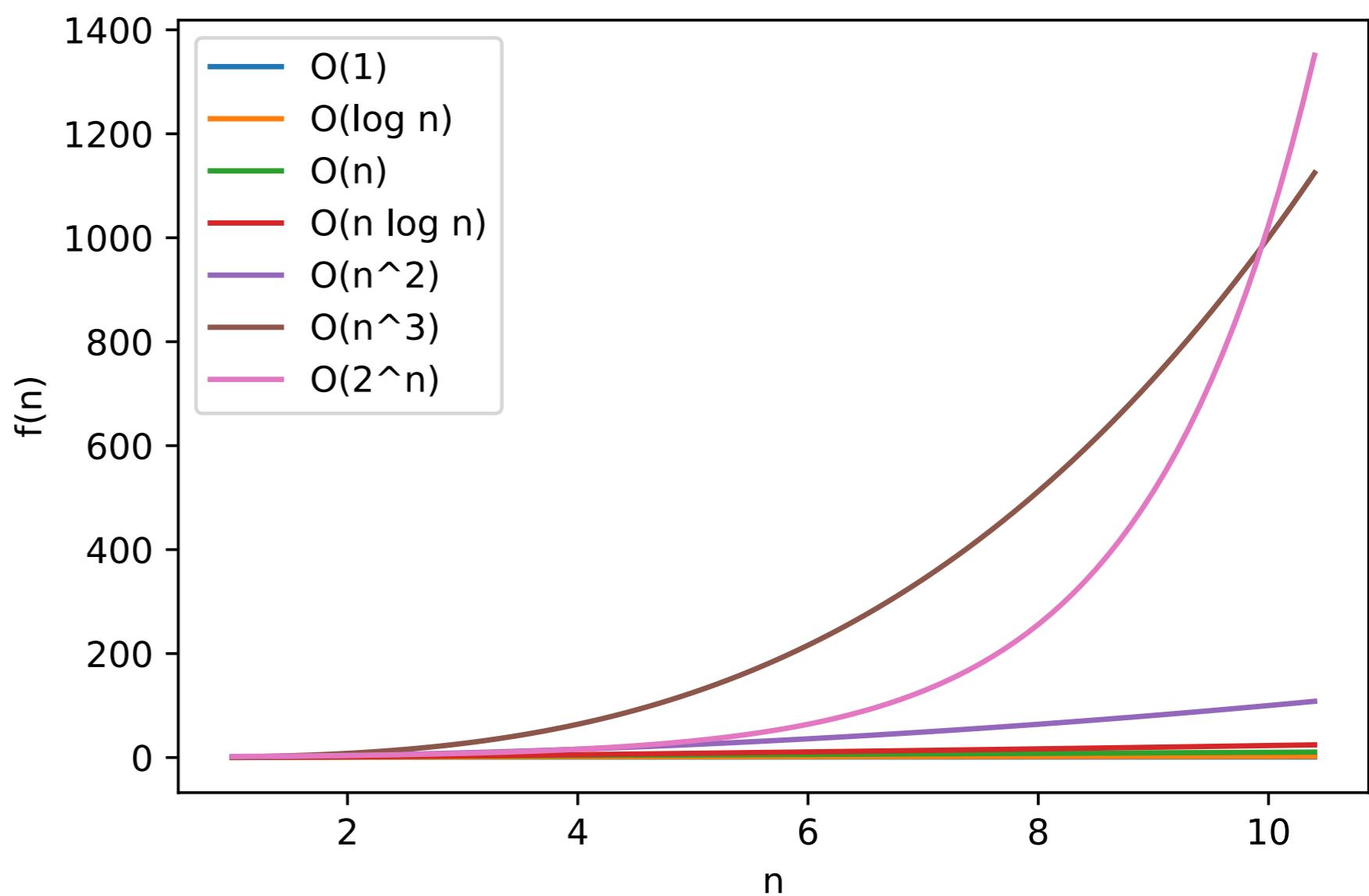
1. Intro to nearest neighbor models
2. Nearest neighbor decision boundary
3. K-nearest neighbors
4. **Big-O & k-nearest neighbors runtime complexity**
5. Improving k-nearest neighbors: modifications and hyperparameters
6. K-nearest neighbors in Python

Big-O

$f(n)$	Name
1	Constant
$\log n$	Logarithmic
n	Linear
$n \log n$	Log Linear
n^2	Quadratic
n^3	Cubic
n^c	Higher-level polynomial
2^n	Exponential

Big-O

$f(n)$	Name
1	Constant
$\log n$	Logarithmic
n	Linear
$n \log n$	Log Linear
n^2	Quadratic
n^3	Cubic
n^c	Higher-level polynomial
2^n	Exponential



Big-O Example 1

$$f(x) = 14x^2 - 10x + 25$$

$$\mathcal{O}(\quad)$$

Big-O Example 2

$$f(x) = (2x + 8)\log_2(x + 9)$$

$$\mathcal{O}(\quad)$$

Big-O Example 2

$$f(x) = (2x + 8)\log_2(x + 9)$$

Why don't we have to distinguish between different logarithms?

Big-O Example 3

```
A = [[1, 2, 3],  
     [2, 3, 4]]
```

```
B = [[5, 8],  
     [6, 9],  
     [7, 10]]
```

$$\mathcal{O}(\quad)$$

```
def matrixmultiply (A, B):  
  
    C = [[0 for row in range(len(A))]  
          for col in range(len(B[0]))]  
  
    for row_a in range(len(A)):  
        for col_b in range(len(B[0])):  
            for col_a in range(len(A[0])):  
                C[row_a][col_b] += \  
                    A[row_a][col_a] * B[col_a][col_b]  
    return C  
  
matrixmultiply(A, B)
```

Out[16]:

```
[[38, 56], [56, 83]]
```

Naive Nearest Neighbor Search

$\mathcal{D}_k := \{\}$

$\mathcal{O}(\quad)$

while $|\mathcal{D}_k| < k$:

- `closest_distance` := ∞
- for $i = 1, \dots, n$, $\forall i \notin \mathcal{D}_k$:
 - `current_distance` := $d(\mathbf{x}^{[i]}, \mathbf{x}^{[q]})$
 - if `current_distance` < `closest_distance`:
 - * `closest_distance` := `current_distance`
 - * `closest_point` := $\mathbf{x}^{[i]}$
- add `closest_point` to \mathcal{D}_k

Naive Nearest Neighbor Search

$\mathcal{D}_k := \{\}$

$\mathcal{O}(\quad)$

while $|\mathcal{D}_k| < k$:

How can you make this more efficient?

- `closest_distance := infinity`
- for $i = 1, \dots, n$, $\forall i \notin \mathcal{D}_k$:
 - `current_distance := $d(\mathbf{x}^{[i]}, \mathbf{x}^{[q]})$`
 - if `current_distance < closest_distance`:
 - * `closest_distance := current_distance`
 - * `closest_point := $\mathbf{x}^{[i]}$`
- add `closest_point` to \mathcal{D}_k

Lecture 2 (Nearest Neighbors)

Topics

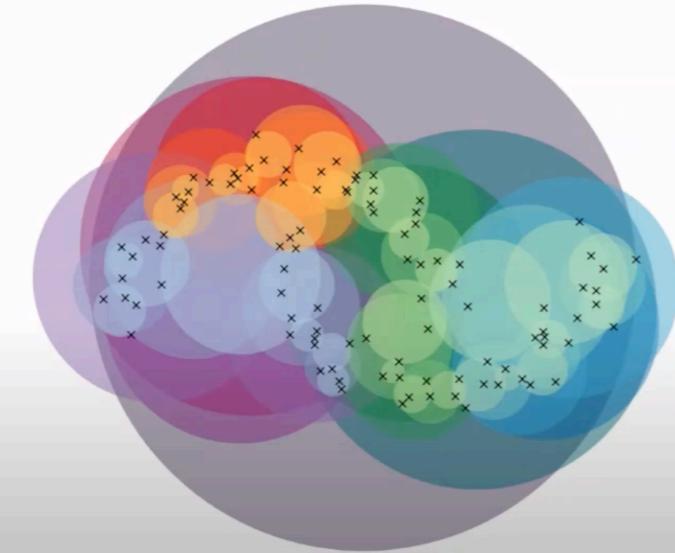
1. Intro to nearest neighbor models
2. Nearest neighbor decision boundary
3. K-nearest neighbors
4. Big-O & k-nearest neighbors runtime complexity
- 5. Improving k-nearest neighbors: modifications and hyperparameters**
6. K-nearest neighbors in Python

Improving Computational Performance

Data Structures



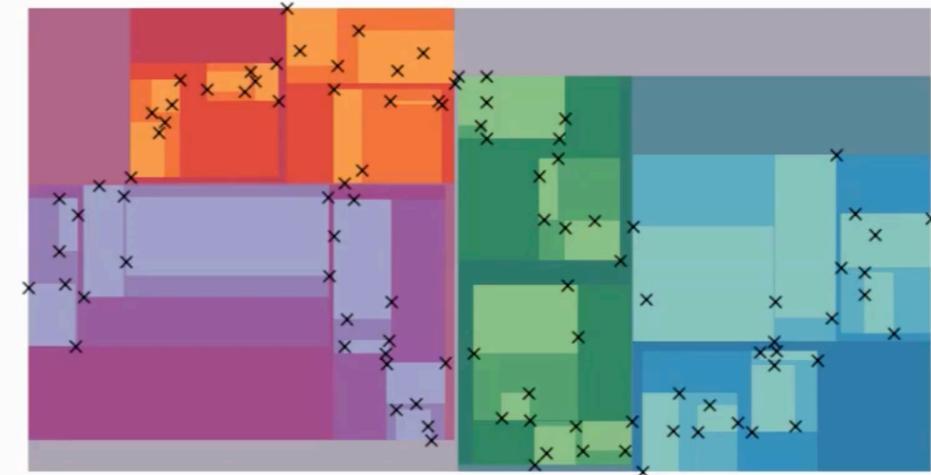
Ball Tree



II ▶ 3:54 / 27:42



KD-Tree



II ▶ 3:54 / 27:42



Random Projection Tree



II ▶ 4:20 / 27:42

Source: PyNNDescent: Fast approximate neighbor search

- https://www.youtube.com/watch?v=xPadY4_kt3o

Improving Computational Performance

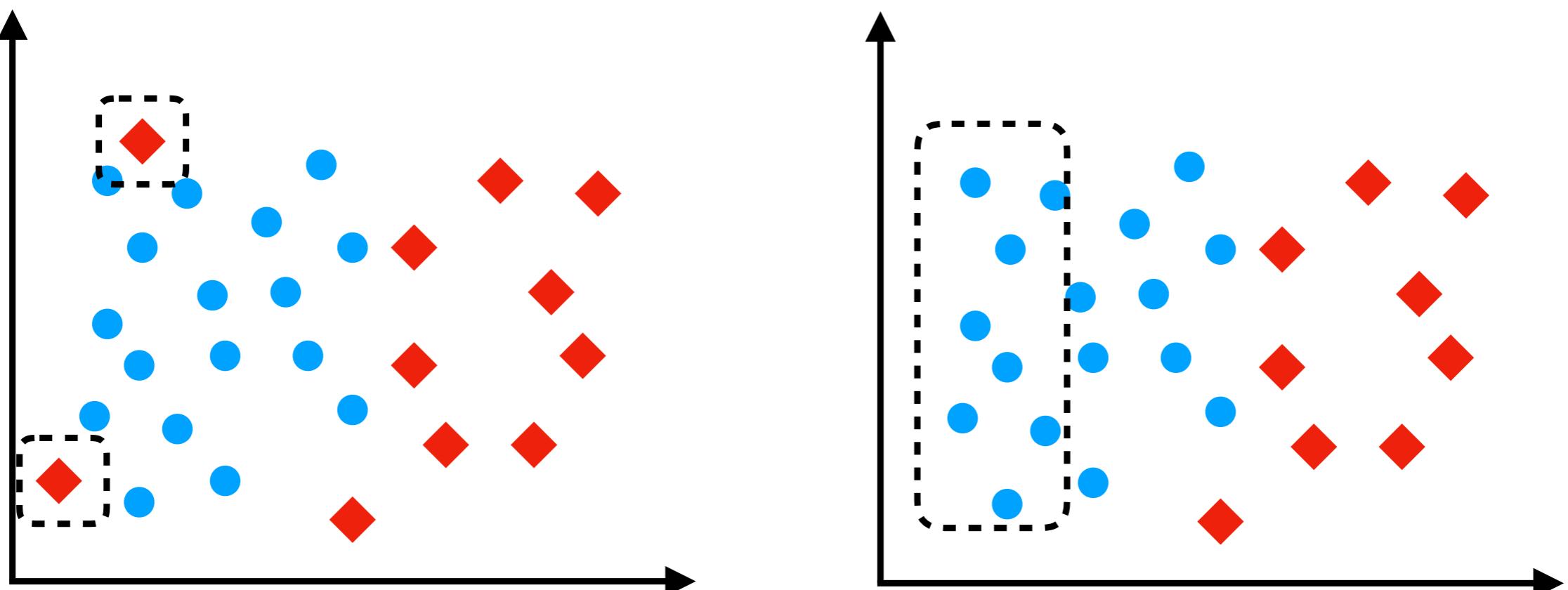
Dimensionality Reduction

Improving Computational Performance

Editing / "Pruning"

Improving Computational Performance

Editing / "Pruning"



Improving Computational Performance

Prototypes

Efficient Nearest Neighbor Language Models

Junxian He, Graham Neubig, Taylor Berg-Kirkpatrick

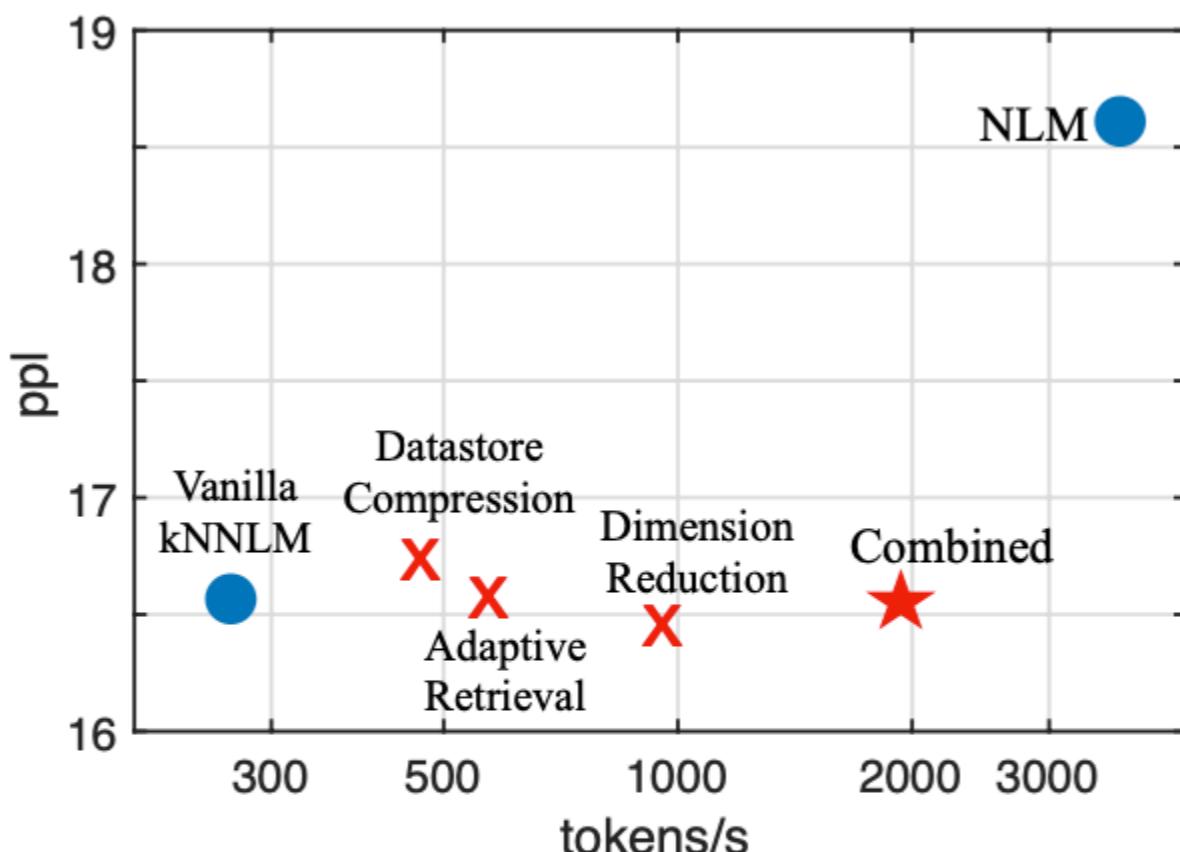


Figure 1: Perplexity (ppl) and evaluation speed for different models. Circled points represent the neural language model (NLM) and k -nearest neighbors language model (k NN-LM) baselines respectively, while others are the methods that we propose and explore in this paper.

<https://arxiv.org/abs/2109.04212>

Improving Predictive Performance

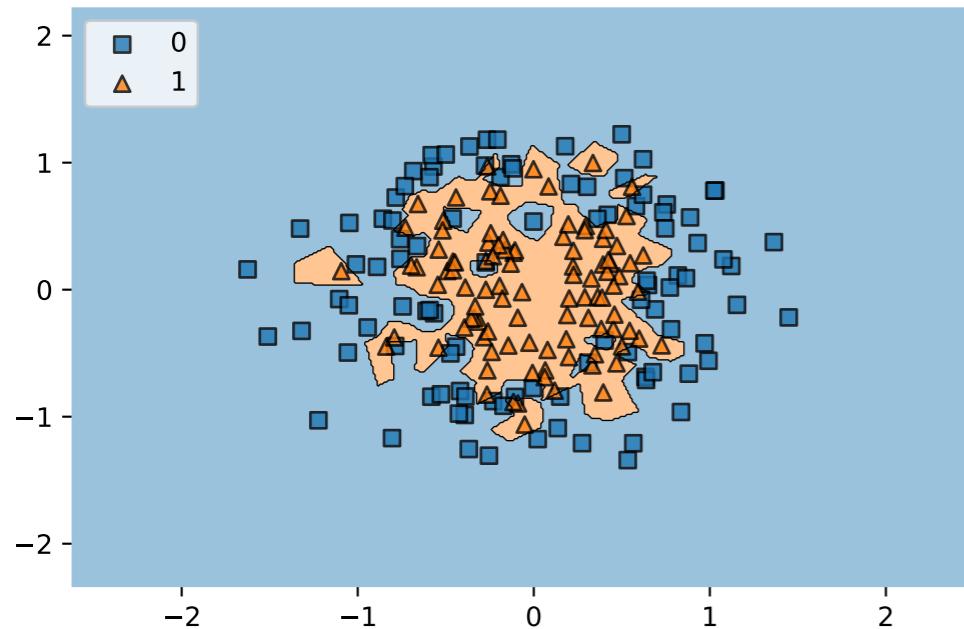
Hyperparameters

Hyperparameters

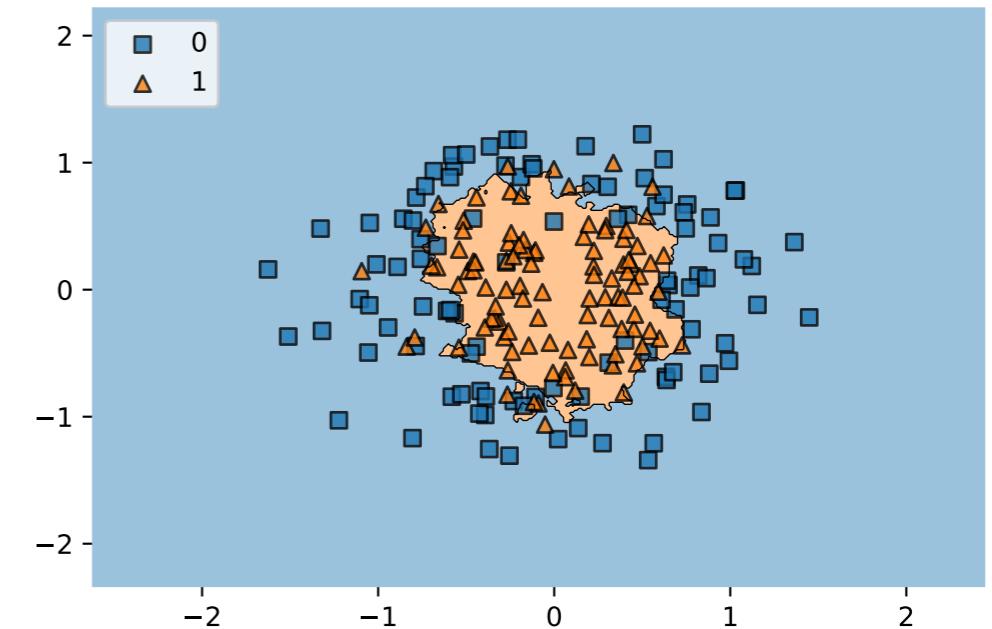
- Value of k
- Scaling of the feature axes
- Distance measure
- Weighting of the distance measure

$$k \in \{1, 3, 7\}$$

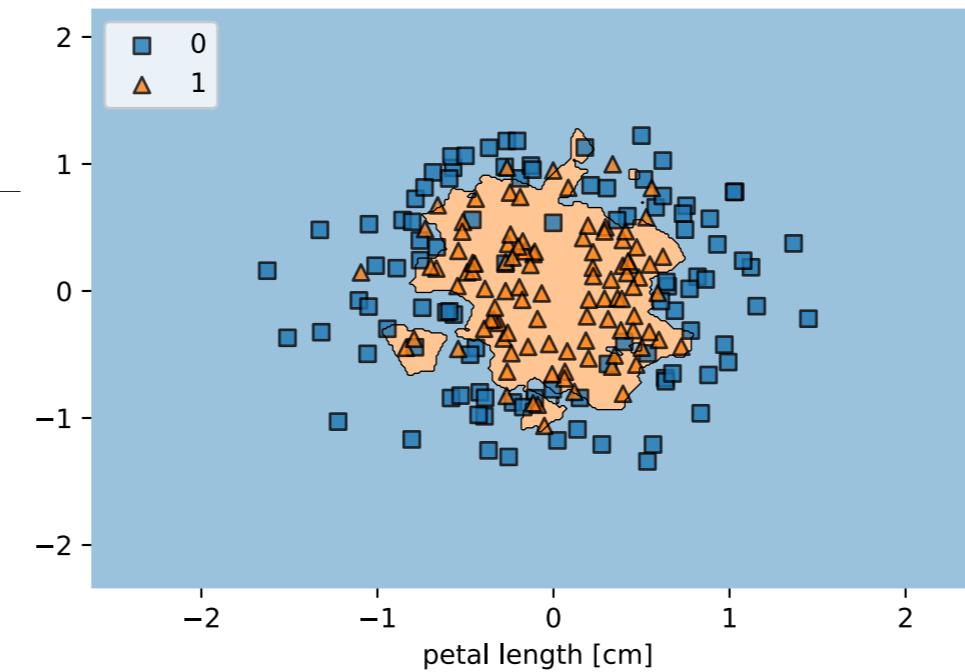
$k = \underline{\hspace{1cm}}$



$k = \underline{\hspace{1cm}}$



$k = \underline{\hspace{1cm}}$



Feature-Weighting via Euclidean Distance

$$d_w(\mathbf{x}^{[a]}, \mathbf{x}^{[b]}) = \sqrt{\sum_{j=1}^m w_j (x_j^{[a]} - x_j^{[b]})^2}$$

Distance-weighted k NN

$$h(\mathbf{x}^{[t]}) = \arg \max_{j \in \{1, \dots, p\}} \sum_{i=1}^k w^{[i]} \delta(j, f(\mathbf{x}^{[i]}))$$

$$w^{[i]} = \frac{1}{d(\mathbf{x}^{[i]}, \mathbf{x}^{[t]})^2 + \epsilon}$$

Small constant to avoid zero division

Lecture 2 (Nearest Neighbors)

Topics

1. Intro to nearest neighbor models
2. Nearest neighbor decision boundary
3. K-nearest neighbors
4. Big-O & k-nearest neighbors runtime complexity
5. Improving k-nearest neighbors: modifications and hyperparameters
- 6. K-nearest neighbors in Python**

***kNN* in Python**

DEMO

More Advanced Material for those who are interested

Tree algorithms explained: Ball Tree Algorithm vs. KD Tree vs. Brute Force

<https://towardsdatascience.com/tree-algorithms-explained-ball-tree-algorithm-vs-kd-tree-vs-brute-force-9746debc940>

PyNNDescent: Fast approximate neighbor search

- https://www.youtube.com/watch?v=xPadY4_kt3o
- based on graphs instead of space-tree structures like KD-Tree and Ball Tree
 - more on graphs: Dong, Moses, Li 2011; Malkov and Yashumin, 2018; Iwasaki and Miyazaki, 2018

Efficient Nearest Neighbor Language Models (2021) Junxian He, Graham Neubig, Taylor Berg-Kirkpatrick

<https://arxiv.org/abs/2109.04212>

Urvashi Khandelwal, Omer Levy, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. 2019. Generalization through memorization: Nearest neighbor language models. In Proceedings of ICLR.

<https://arxiv.org/abs/1911.00172>