HPE DSI 311
Introduction to Machine Learning

Spring 2023

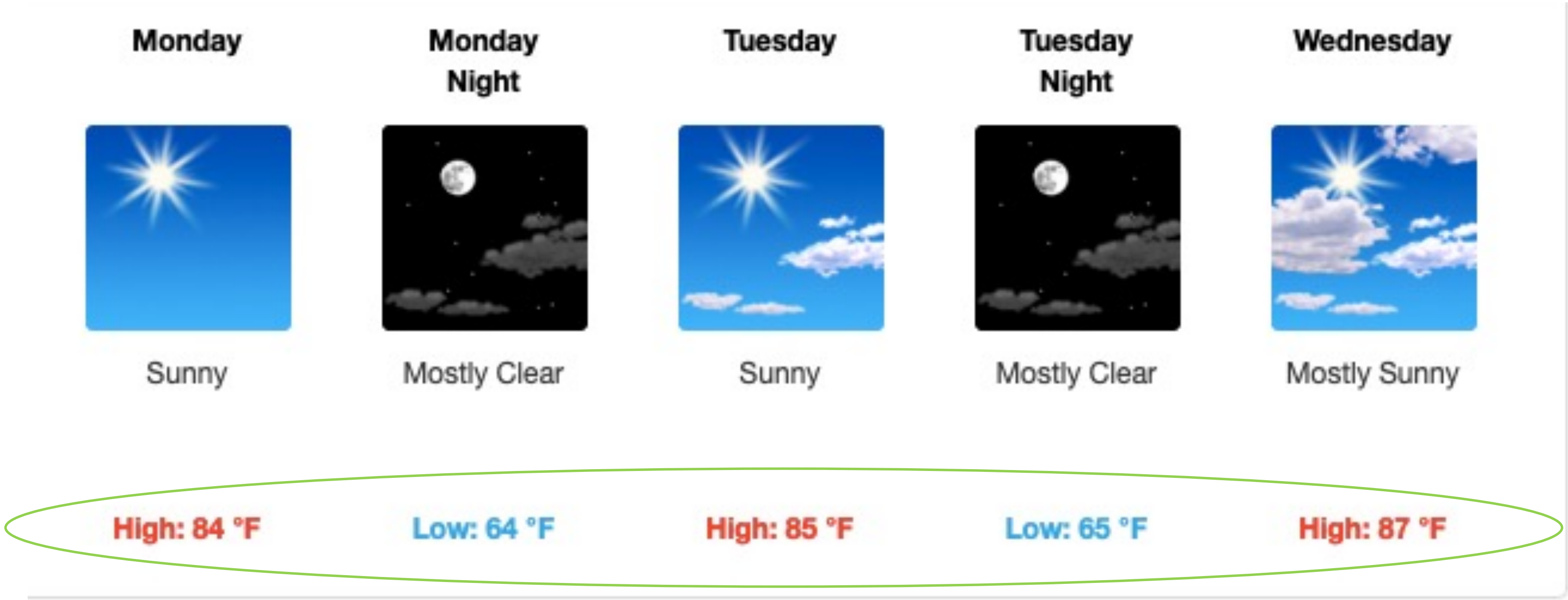
Instructor: Ioannis Konstantinidis

Overview

- Simple models to get started:
  - K nearest neighbors
  - Logistic Regression
- Hands-on examples
  - Jupyter Notebooks
  - Data Summarization

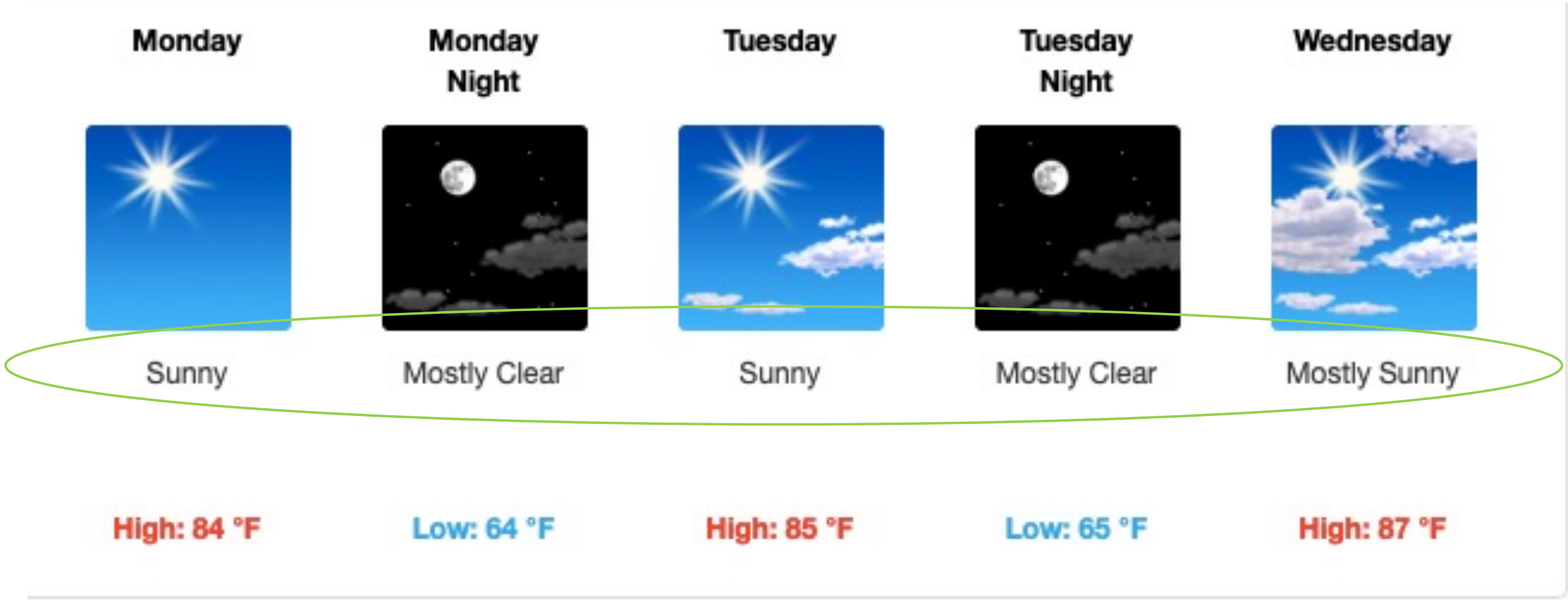
ML techniques: How do they differ?

Supervised ML



Regression:
predict a number

Supervised ML



Classification: predict a label

Regression: predict a number

ML techniques by data type

Supervised: works with data that is already classified to tailor rules for classifying new (and as yet unclassified) individuals

- Predict: What would the data point x do?
- Examples: Regression, Classification

Unsupervised: aims to uncover groups of observations from initially unclassified data

- Analyze: How is the data set X structured?
- Examples: Clustering, Anomaly Detection

Hewlett Packard Enterprise
Data Science Institute
UNIVERSITY OF HOUSTON

Classification using K Nearest Neighbors

K-Nearest Neighbors: algorithm

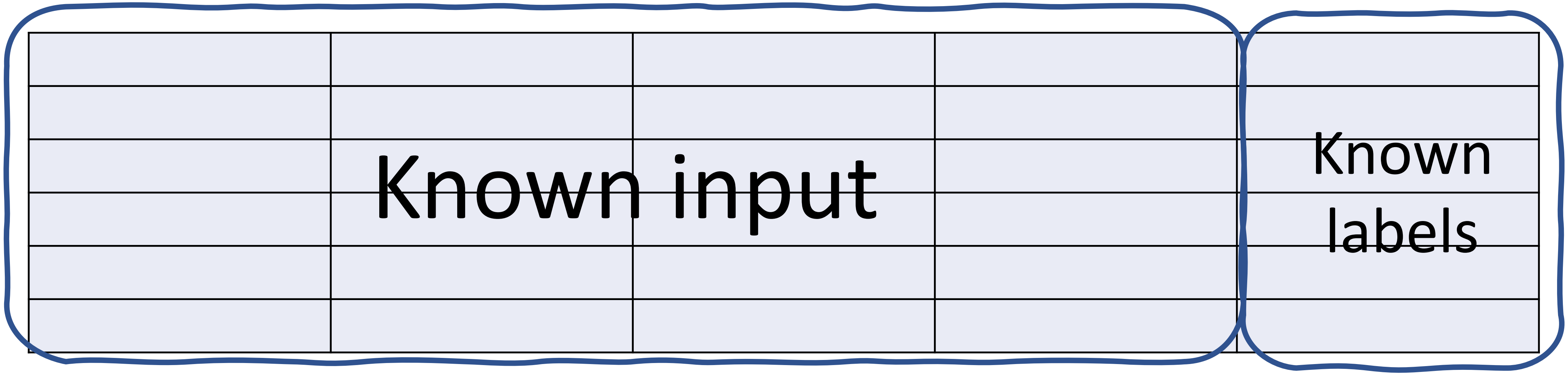
Training algorithm

- All training example points (*x_train, y_train*) go into a reference list

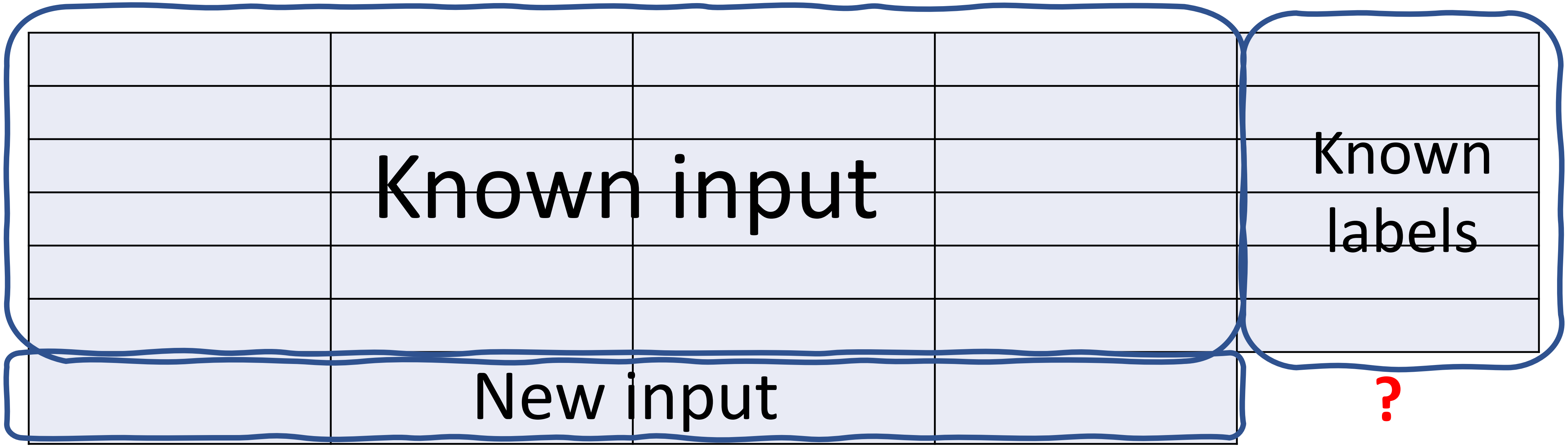
Classification algorithm (for fixed k)

- Given a query instance *x_test* to be classified, find the **nearest** point *x_train* in the reference list
- Repeat until the **k nearest points** are identified from the list
- Calculate *y_predict* based on the values of *y_train* for these neighbors, i.e., the k nearest points

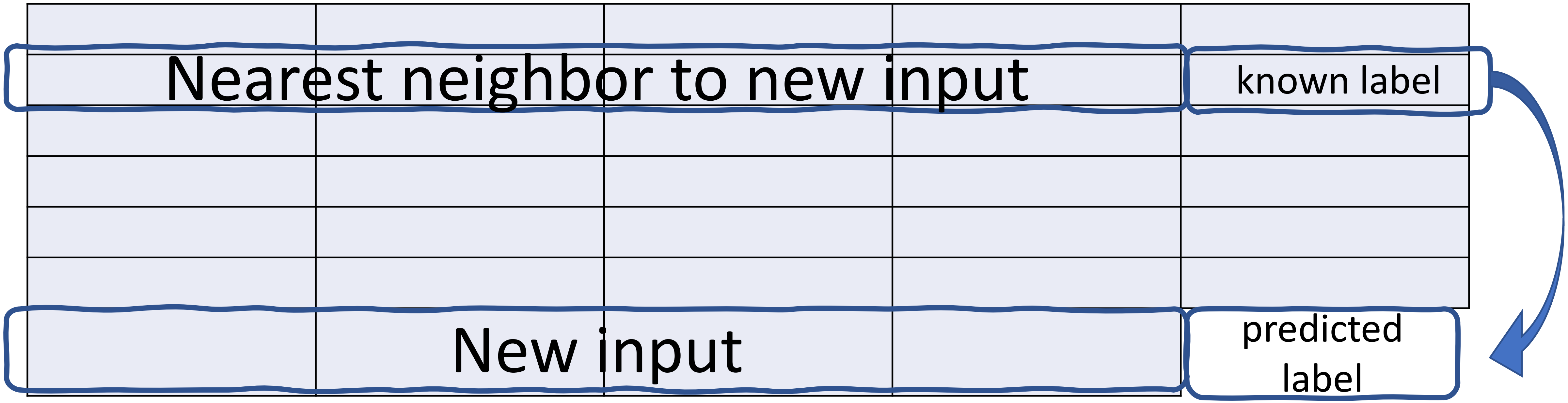
In practical terms: training

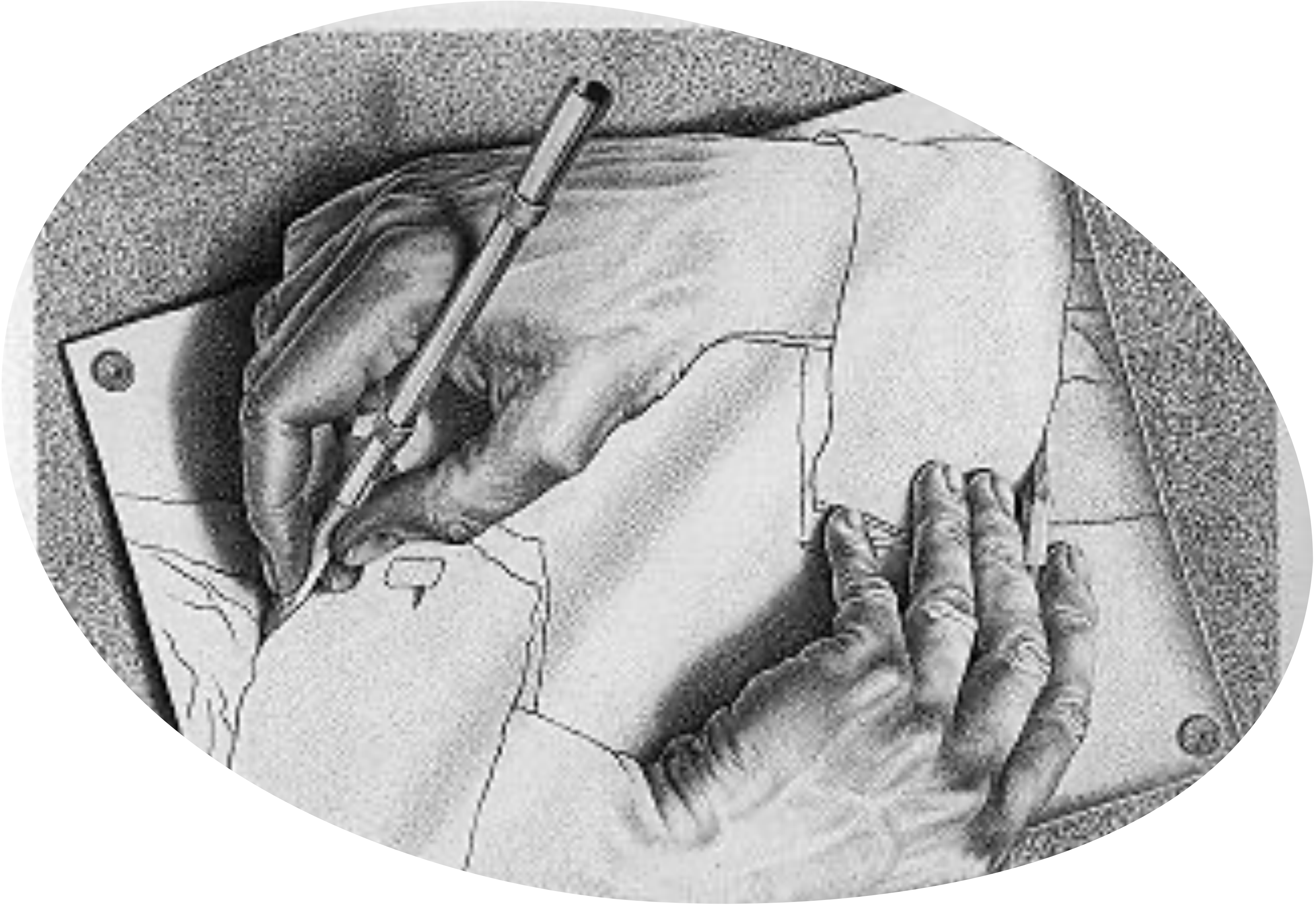


In practical terms: prediction



In practical terms: prediction





Hands-on Example:

k-NN

K-Nearest Neighbors: sklearn implementation

KNeighborsClassifier(*n_neighbors=5,*
weights='uniform',
algorithm='auto',
leaf_size=30,
p=2,
metric='minkowski',
metric_params=None,
n_jobs=None,
***kwargs*)

https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

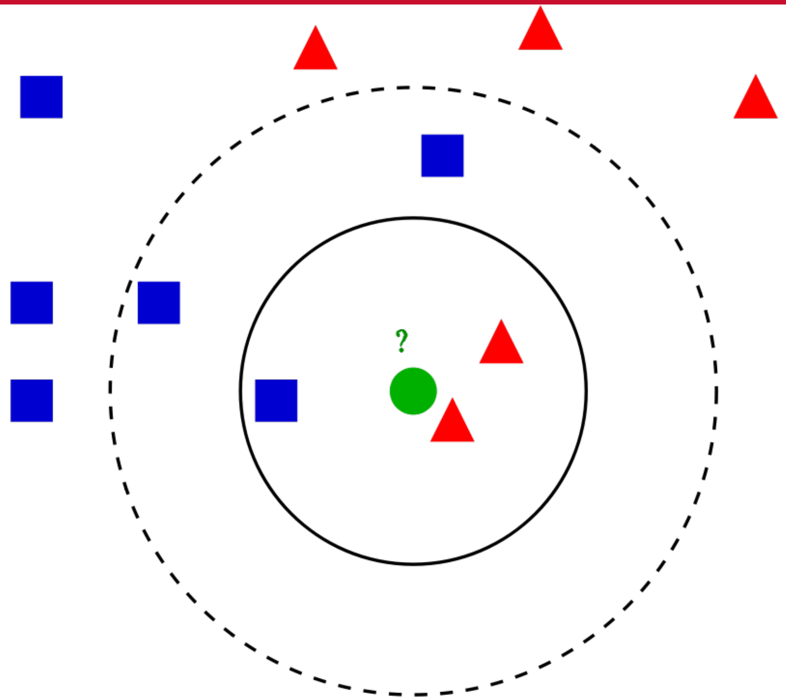
K-Nearest Neighbors: choice of K

KNeighborsClassifier(***n_neighbors***=5,
weights='uniform',
algorithm='auto',
leaf_size=30,
p=2,
metric='minkowski',
metric_params=None,
n_jobs=None,
***kwargs*)

K-Nearest Neighbors: choice of K

Who are the neighbors of the new sample (green circle)?

Blue squares or red triangles?



$$g(\mathbf{x}) = \sum_{i \in \mathrm{kNN}(\mathbf{x})} y_i$$

- k = 1: a RED TRIANGLE is the nearest neighbor, so the guess would be RED TRIANGLE
- k = 3 (solid line circle): 2 red triangles and only 1 blue square in the neighborhood, so the guess would be RED TRIANGLE
- k = 5 (dotted line circle): 3 blue squares and only 2 red triangles in the neighborhood, so the guess would be BLUE SQUARE

K-Nearest Neighbors: choice of weight

KNeighborsClassifier(*n_neighbors=5,*
weights*='uniform',*
algorithm='auto',
leaf_size=30,
p=2,
metric='minkowski',
metric_params=None,
n_jobs=None,
***kwargs*)

K-Nearest Neighbors: choice of weight

Weights default='uniform'

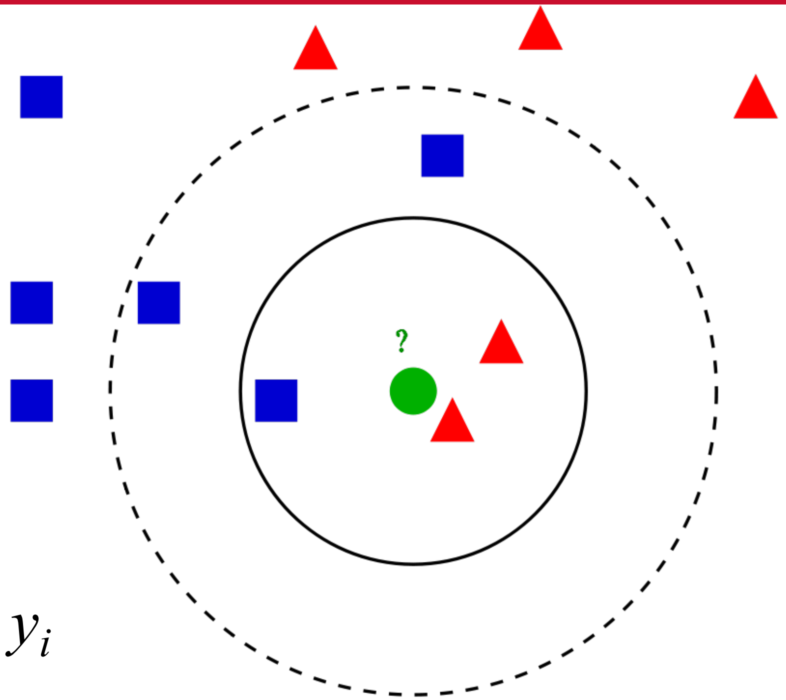
weight function used in prediction. Possible values:

- 'uniform' : uniform weights. All points in each neighborhood are weighted equally.
- 'distance' : weight points by the inverse of their distance. In this case, closer neighbors of a query point will have a greater influence than neighbors which are further away.
- [callable] : a user-defined function which accepts an array of distances, and returns an array of the same shape containing the weights.

K-Nearest Neighbors: choice of weight

Who are the neighbors of the new sample (green circle)?

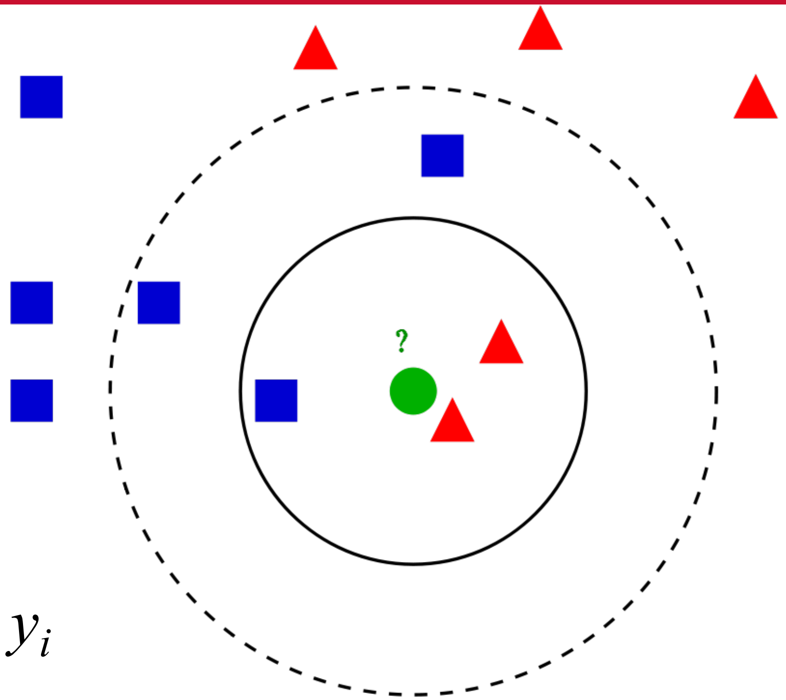
Blue squares or red triangles?



$$g(\mathbf{x}) = \sum_{i \in \mathrm{kNN}(\mathbf{x})} weight(\mathbf{x}_i, \mathbf{x})\, y_i$$

Default option for $weight(\mathbf{x}_i, \mathbf{x})$ is uniform (every weight = 1)

K-Nearest Neighbors: choice of weight



Who are the neighbors of the new sample (green circle)?

Blue squares or red triangles?

$$g(\mathbf{x}) = \sum_{i \in \mathrm{kNN}(\mathbf{x})} weight(\mathbf{x}_i, \mathbf{x})\, y_i$$

k = 5:

3 distant blue squares and 2 close red triangles in the neighborhood

- Uniform weights: the guess would be BLUE SQUARE
- Distance weights: the guess would be RED TRIANGLE

K-Nearest Neighbors: choice of metric

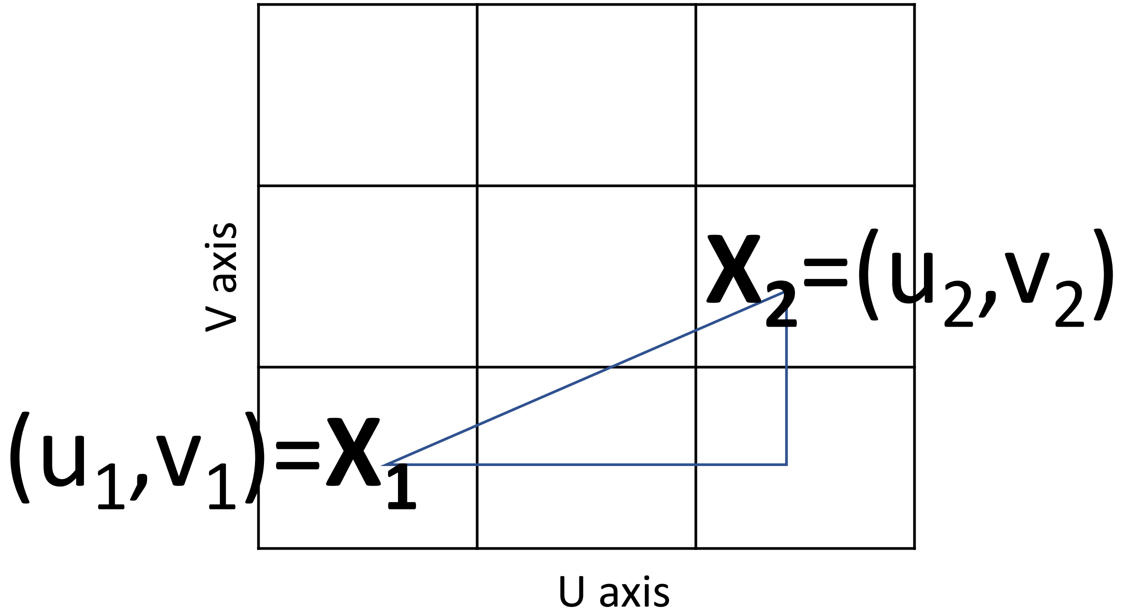
KNeighborsClassifier(*n_neighbors=5,*
weights='uniform',
algorithm='auto',
leaf_size=30,
p=2,
metric='minkowski',
metric_params=None,
n_jobs=None,
***kwargs*)

K-Nearest Neighbors: choice of metric

Dimension(X) = 2

Pythagorean theorem:

$$\{\text{distance}(\mathbf{X_1}, \mathbf{X_2})\}^2$$
$$=$$
$$\{\text{distance\_along\_u\_axis}(\mathbf{X_1}, \mathbf{X_2})\}^2$$
$$+$$
$$\{\text{distance\_along\_v\_axis}(\mathbf{X_1}, \mathbf{X_2})\}^2$$



$$\{\text{distance}(\mathbf{X_1}, \mathbf{X_2})\}^2 = (\mathbf{u_1} - \mathbf{u_2})^2 + (\mathbf{v_1} - \mathbf{v_2})^2$$

Euclidean: As the crow flies

Dimension(X) = 2

Example:

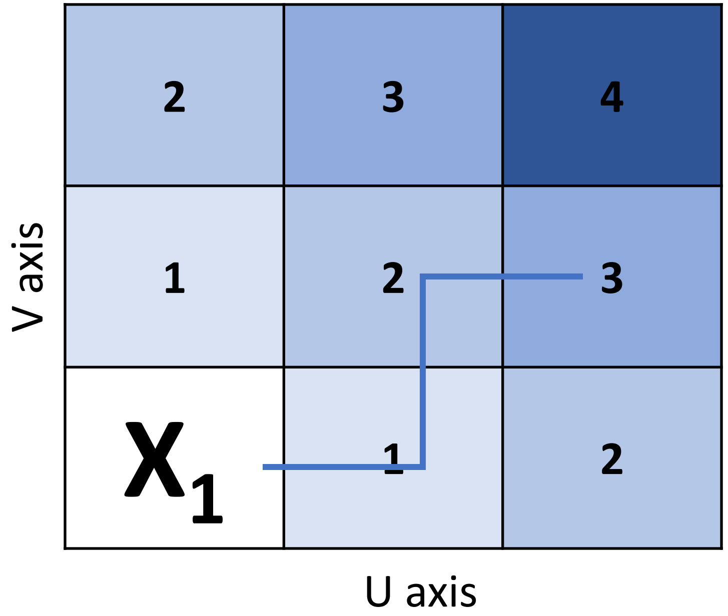
V axis			
	2	2.236	2.828
	1	1.414	2.236
	X_1	1	2

U axis

Manhattan: As your rideshare drives

Dimension(X) = 2

distance($\mathbf{X}_1$, $\mathbf{X}_2$)
=
distance_along_u_axis($\mathbf{X}_1$, $\mathbf{X}_2$)
+
distance_along_v_axis($\mathbf{X}_1$, $\mathbf{X}_2$)



$$\text{distance}(\mathbf{X}_1 , \mathbf{X}_2) = |\mathbf{u}_1 - \mathbf{u}_2| + |\mathbf{v}_1 - \mathbf{v}_2|$$

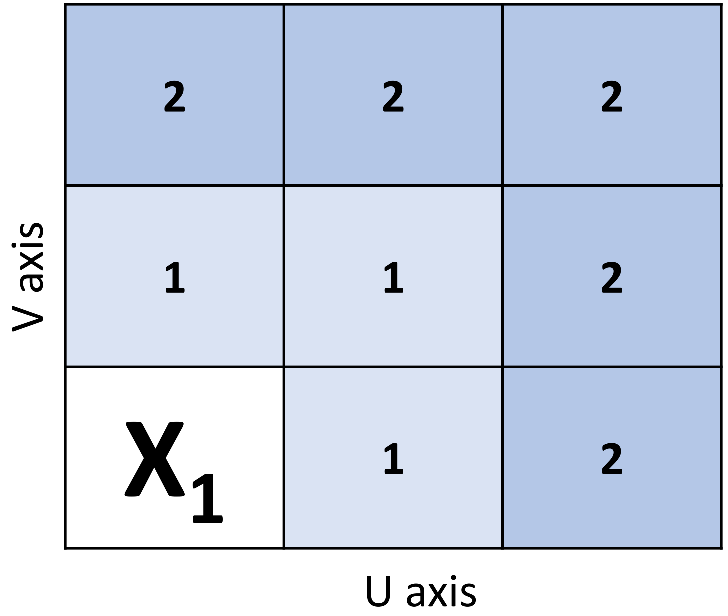
Maximum Distance: close in every way

Dimension(X) = 2

distance($\mathbf{X_1}$, $\mathbf{X_2}$)

=

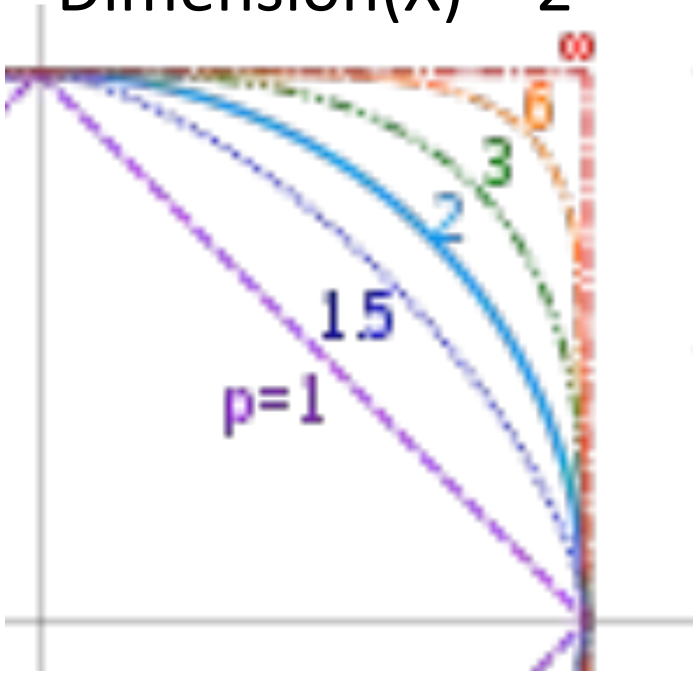
Max{ distance_along_u_axis($\mathbf{X_1}$, $\mathbf{X_2}$),
distance_along_v_axis($\mathbf{X_1}$, $\mathbf{X_2}$) }



Minkowski: The L^p metric

Dimension(X) = 2

$$\{\, \text{distance}(\, \mathbf{X}_1 , \mathbf{X}_2 \,) \,\}^p$$
$$=$$
$$\{\, \text{distance\_along\_u\_axis}(\, \mathbf{X}_1 , \mathbf{X}_2 \,) \,\}^p$$
$$+$$
$$\{\, \text{distance\_along\_v\_axis}(\, \mathbf{X}_1 , \mathbf{X}_2 \,) \,\}^p$$



p = 2: Euclidean – Each point on blue arc is same distance from LL corner
p = 1: Manhattan – Each point on violet diagonal is same distance from LL corner
p = ∞ : Maximum – Each point on red sides is same distance from LL corner

Minkowski: The L^p metric

Metric is the choice of distance for finding the nearest neighbors. The default metric is minkowski with p=2, which is equivalent to the standard Euclidean metric.

P is the power parameter for the Minkowski metric.

- When p = 1, this is equivalent to using manhattan_distance (l1)
- When p=2, this is euclidean_distance (l2)
- For arbitrary p, it is minkowski_distance (l_p)

K-Nearest Neighbors: choice of algorithm

KNeighborsClassifier(*n_neighbors=5,*
weights='uniform',
algorithm='auto',
leaf_size=30,
p=2,
metric='minkowski',
metric_params=None,
n_jobs=None,
***kwargs*)

K-Nearest Neighbors: choice of algorithm

Algorithm default='auto'

Algorithm used to find the nearest neighbors:

- 'ball_tree' will use a BallTree algorithm
- 'kd_tree' will use a KDTree algorithm
- 'brute' will use a brute-force search
- 'auto' will attempt to decide the most appropriate algorithm based on the values passed to fit() method

K-Nearest Neighbors: AKA lazy, instance-based learning

Lazy: No training process

Instance-based: Construct only local approximation to the target function that differs based on the neighborhood of each new query instance

Are there any disadvantages?

Cost of classifying new instances can be high:

- Nearly all computation takes place at classification time rather than learning time
- Number of points needed for good coverage of feature space scales exponentially with number of dimensions

What are some other ML methods?

Logistic Regression

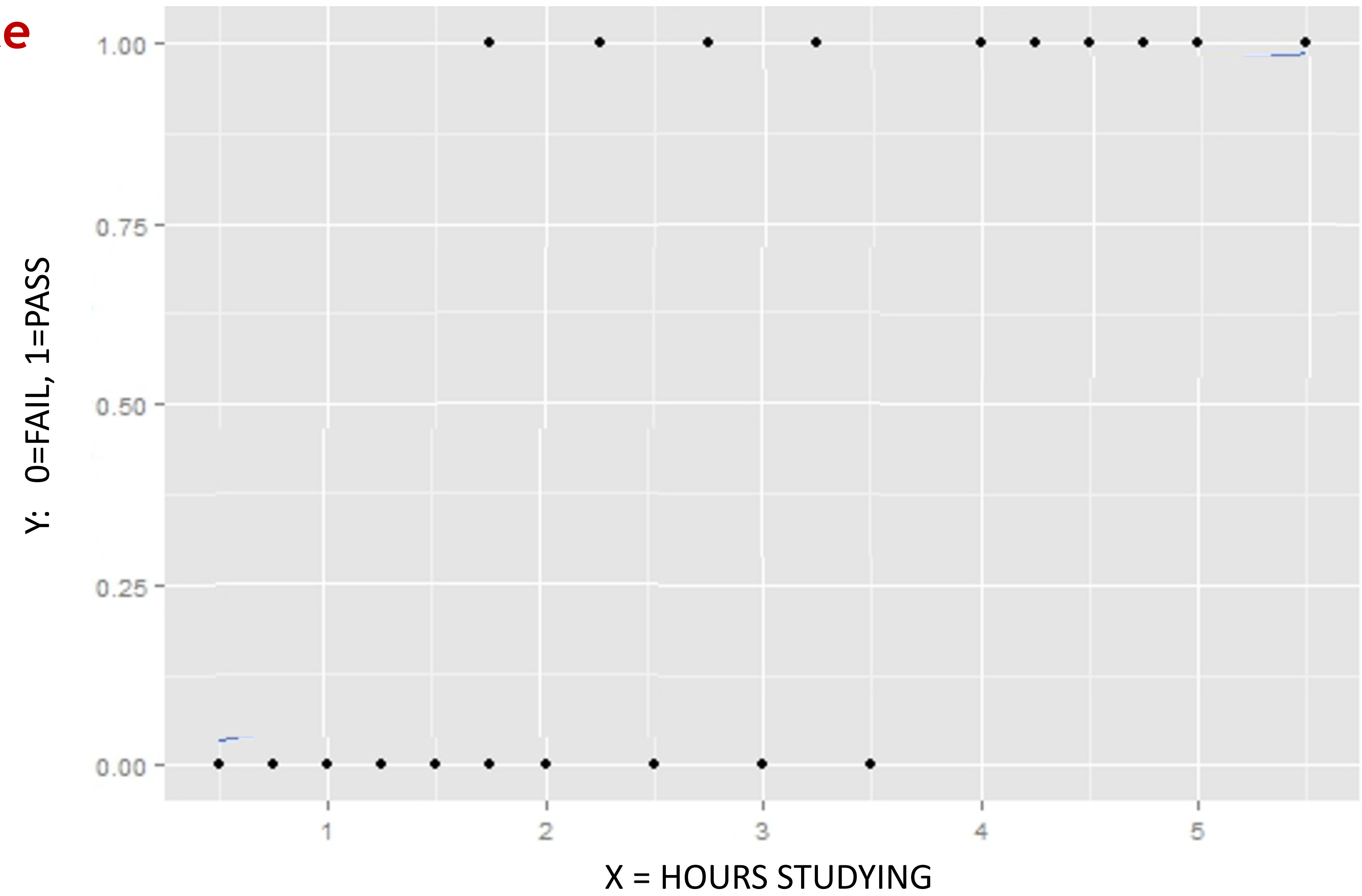
VERY simple example

HOURS STUDYING	PASSED EXAM
4	YES
1	NO
3.5	NO
2.25	YES
0.25	NO

Known input

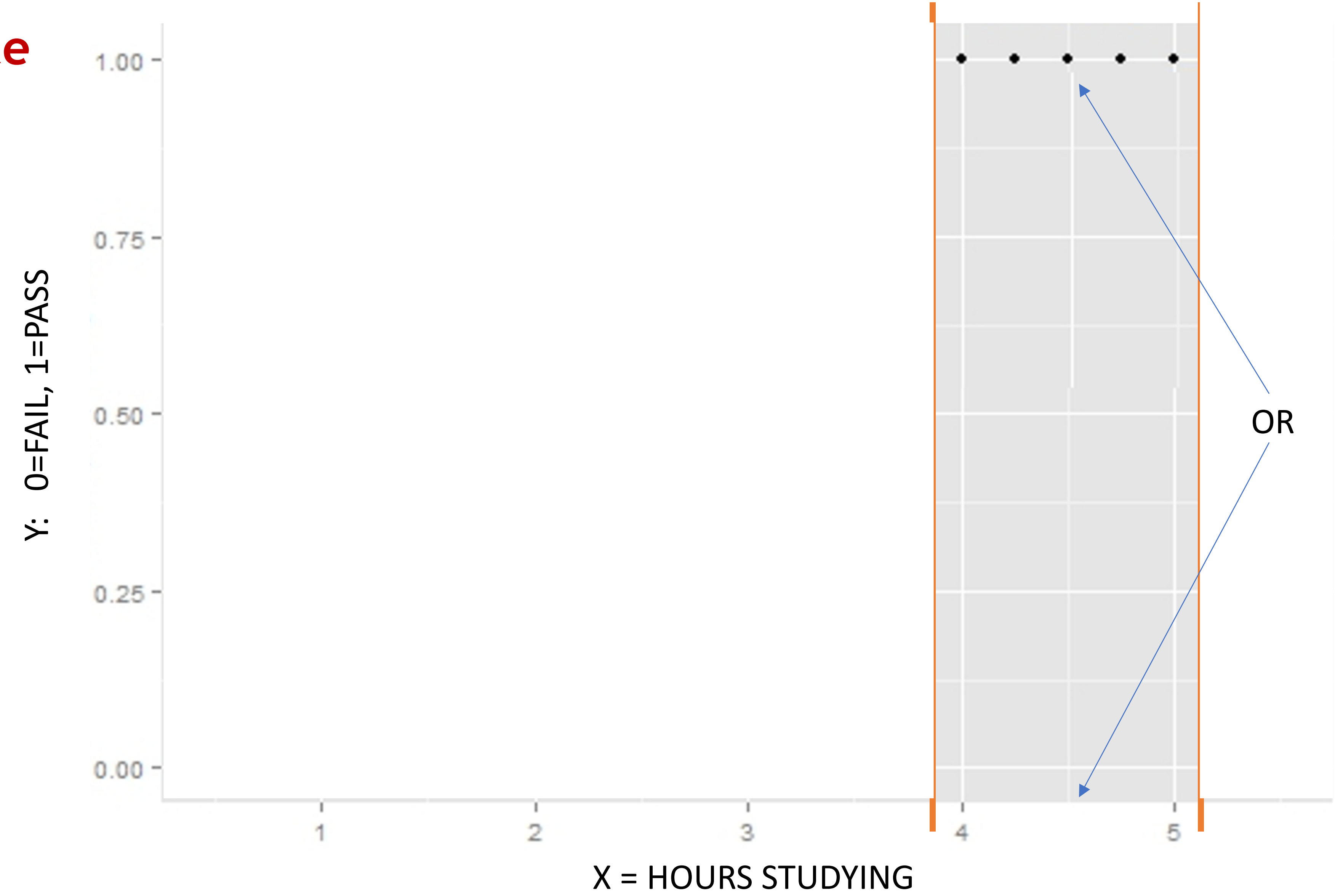
Known labels

VERY simple example



VERY simple example

New point: X=4.5

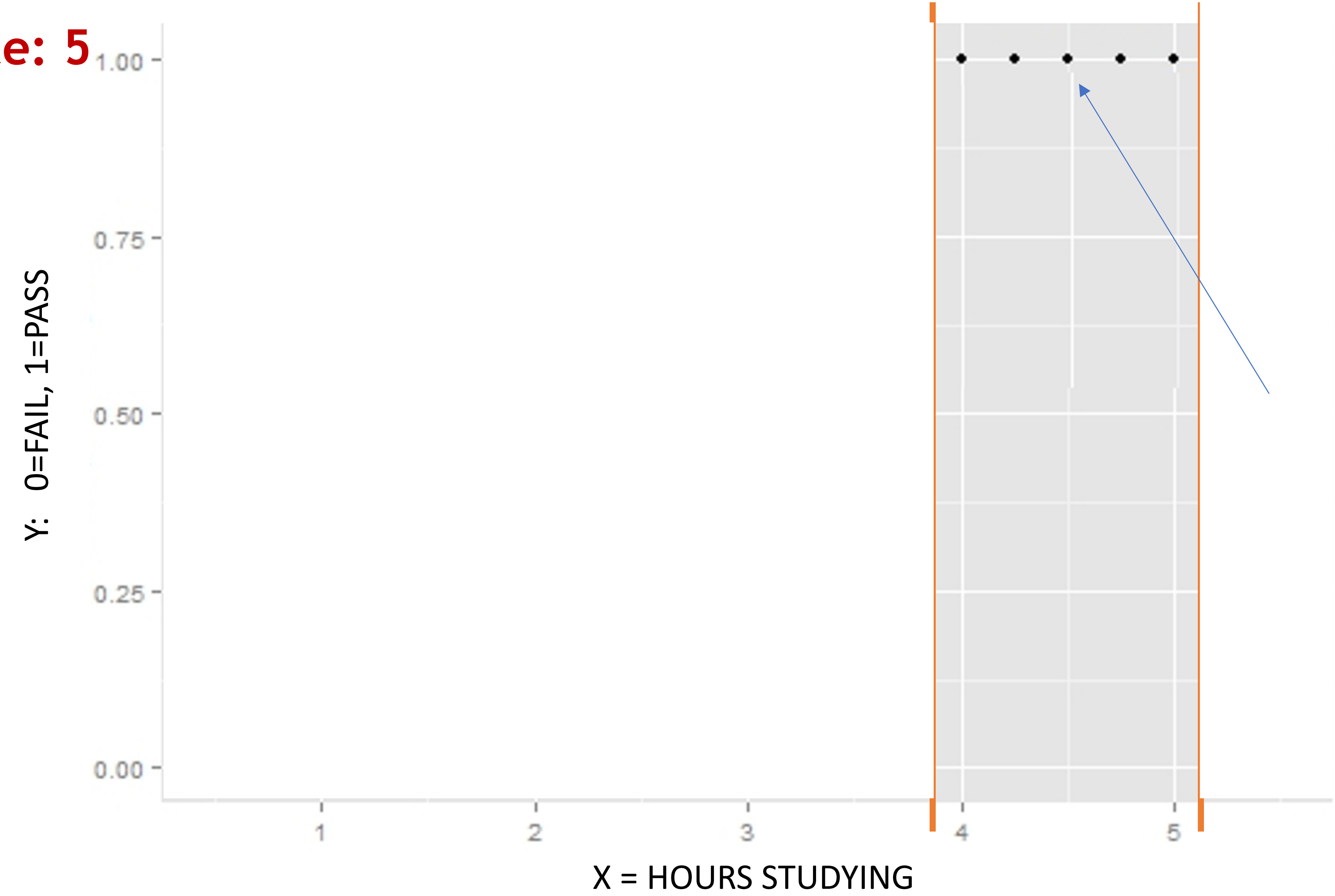


VERY simple example: 5

New point: X=4.5

All five neighbors
are labeled PASS

Predicted label = PASS



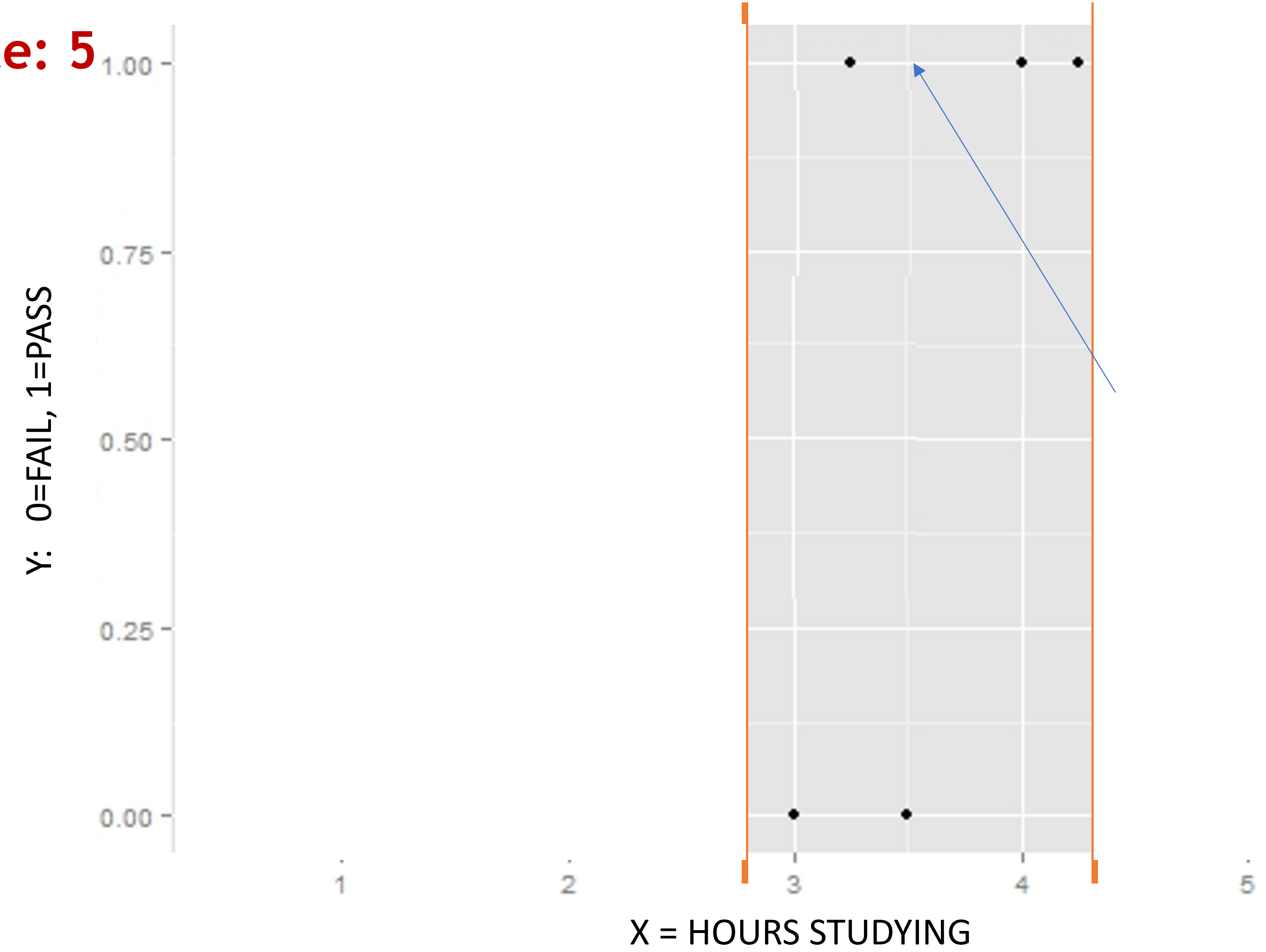
VERY simple example: 5

New point: X=3.5

three neighbors
are labeled PASS,
and two neighbors
are labeled FAIL

Predicted label = PASS

with probability 60%



VERY simple example: 5

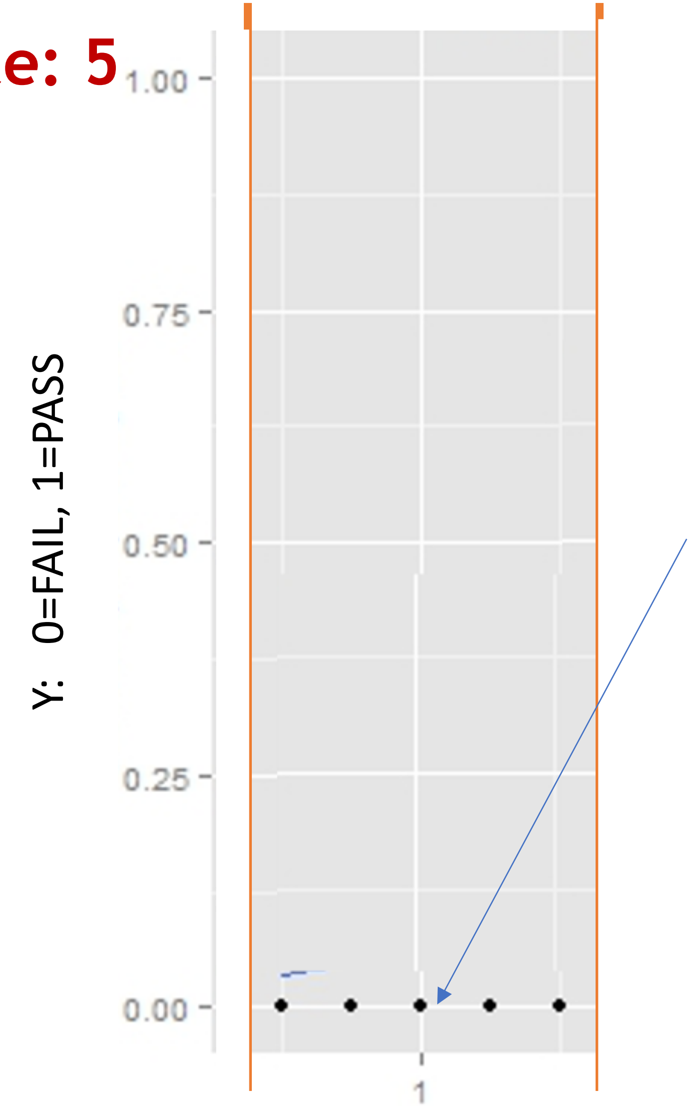
New point: X=1

All five neighbors

are labeled FAIL

Predicted label = PASS

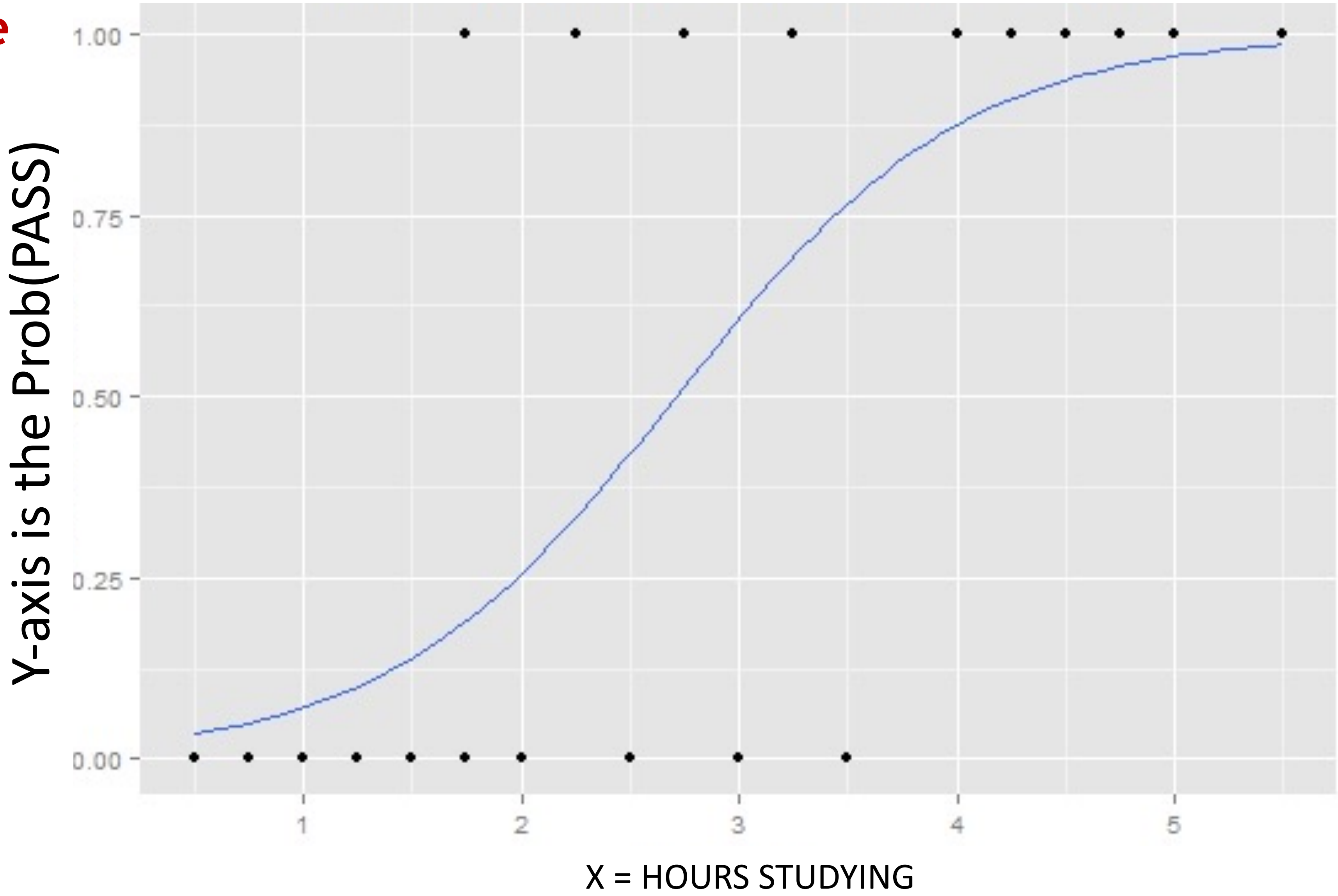
with probability 0%



X = HOURS STUDYING

VERY simple example

Blue line is the probability of PASS being the correct label for a new point X



CAUTION!

Despite its name, Logistic Regression is a method for **classification** tasks

Prob(y) = Proportion of “success” among neighbors

$$Prob(y) = \frac{\sum y_i}{n} = \frac{\#\text{ of 1's}}{\#\text{ of trials}} = \text{Proportion of "success"}$$

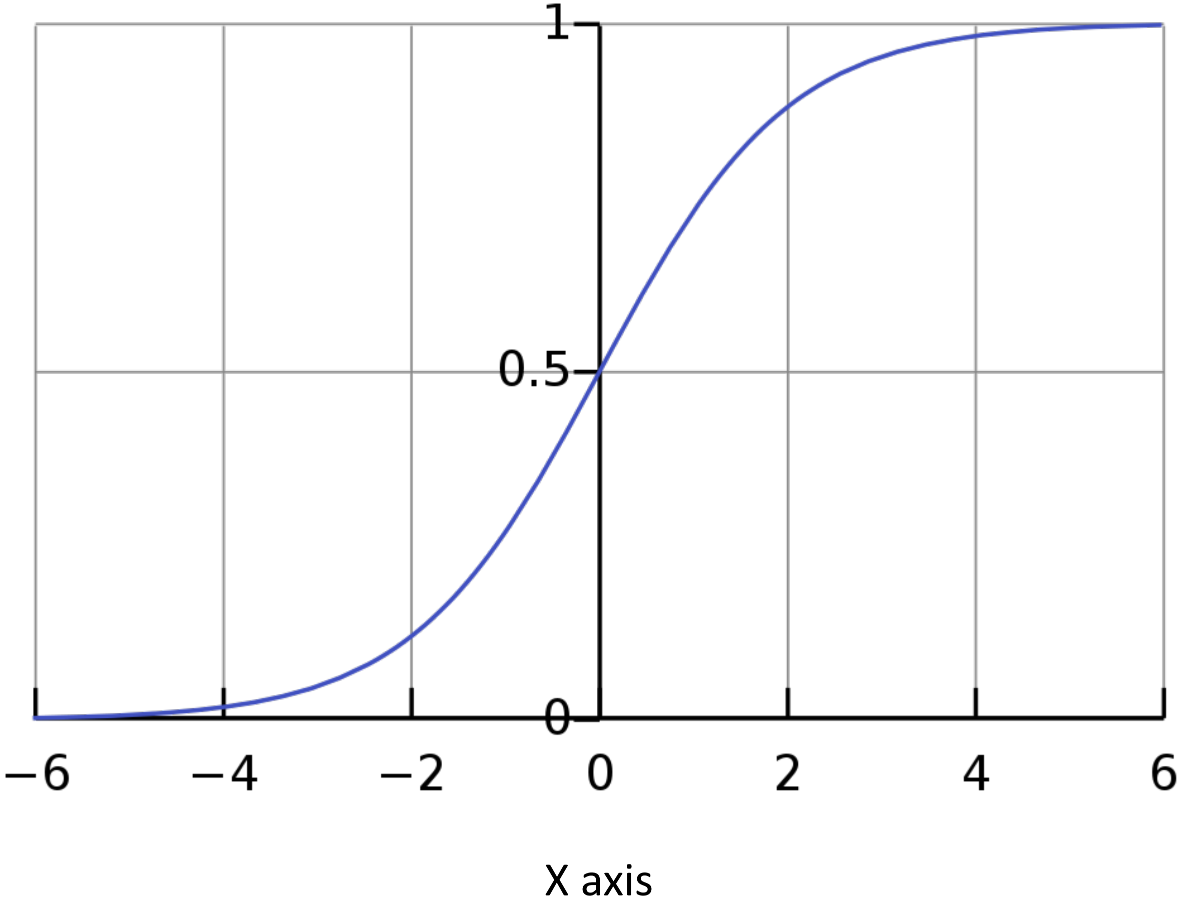
Goal of logistic regression: Predict the “true” proportion of success prob(y) at any value of the predictor variable X

Approximated by maximizing conditional log-likelihood: $\sum \log prob(y|x)$

Model for Prob(y)

The logistic function

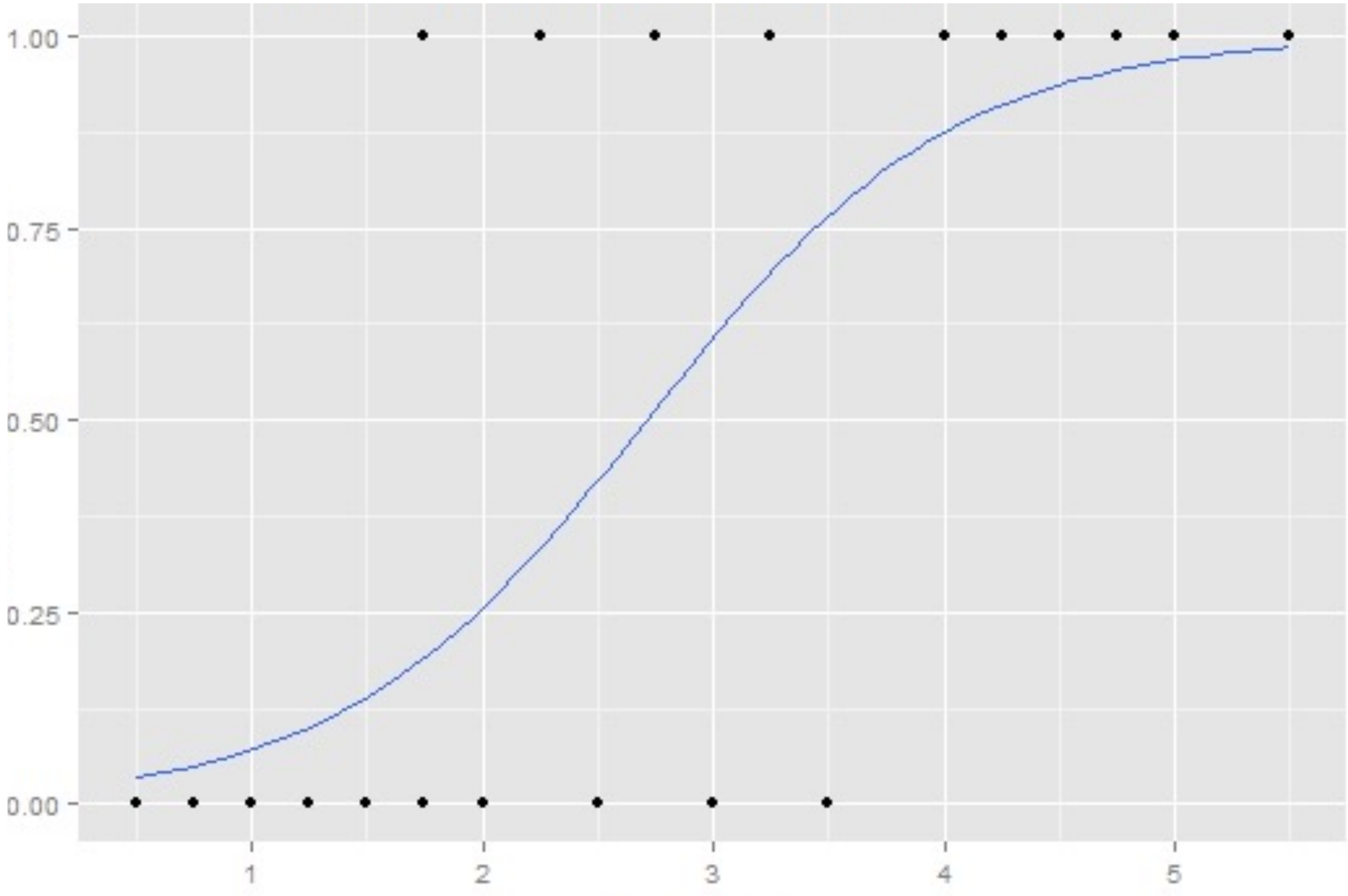
$$\text{Probability(y)} = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1}$$



VERY simple example

Probability(PASS) =

$$\frac{1}{1 + \exp(-(1.5046 \cdot \text{Hours} - 4.0777))}$$



X = HOURS STUDYING

In practical terms

HOURS STUDYING	PASSED EXAM
4	YES
1	NO
3.5	NO
2.25	YES
0.25	NO

Known input

Known labels

Replace

with

HOURS STUDYING	PROB. PASS
4	%
1	%
3.5	%
2.25	%
0.25	%

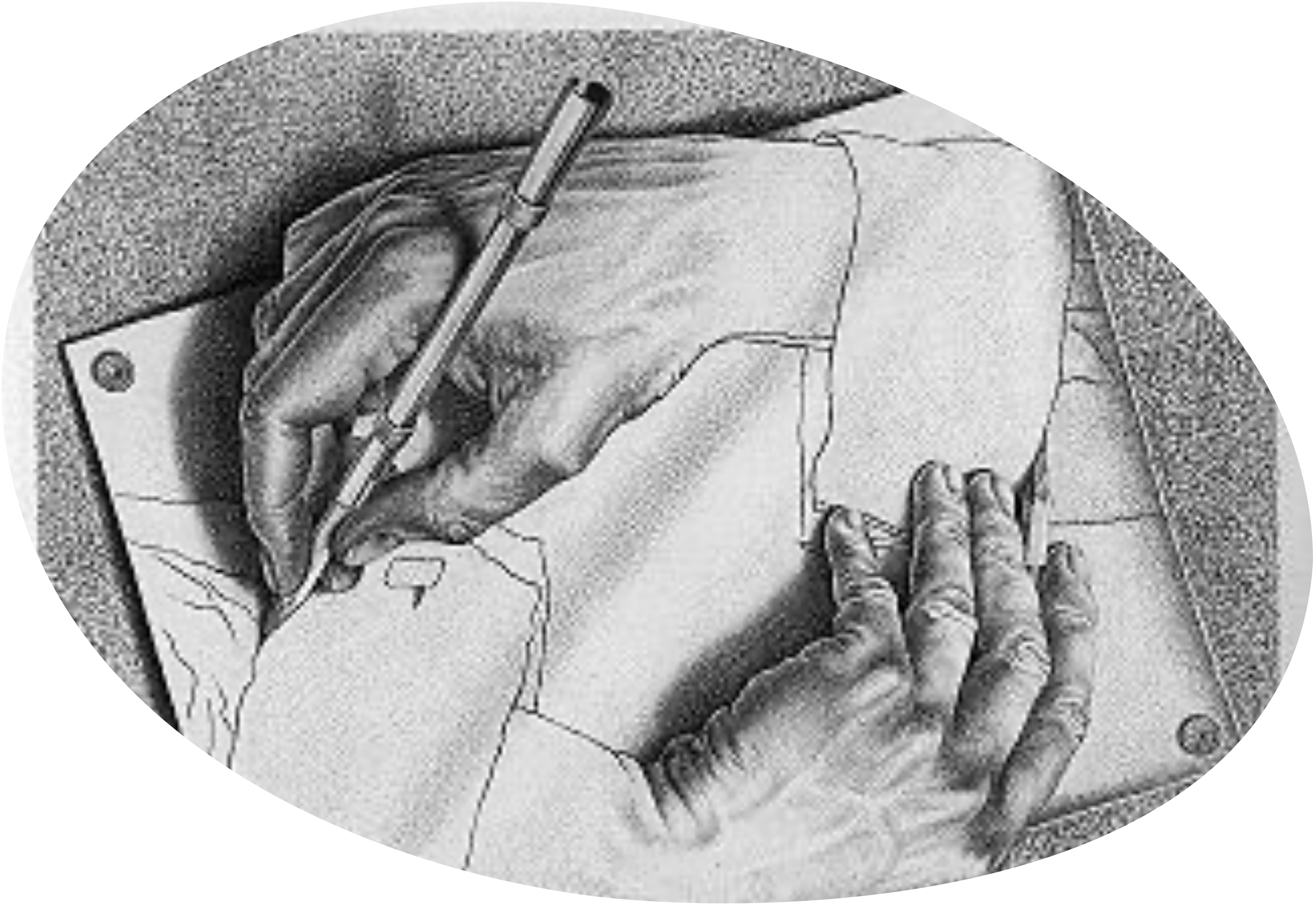
Compute formula based on logistic function

The logistic function in more dimensions

$$Probability(Y) = \frac{e^{u}}{1+e^{u}} = \frac{1}{1+e^{-u}}$$

Where u is the regular linear regression equation on M variables:

$$u = A + B_1 X_1 + B_2 X_2 + \ldots + B_M X_M$$



Hands-on Example:

Logistic regression

Logistic Regression Classifier

LogisticRegression(*penalty='l2',*
dual=False,
tol=0.0001,
C=1.0,
fit_intercept=True,
intercept_scaling=1,
class_weight=None,
random_state=None,
solver='lbfgs',
max_iter=100,
multi_class='auto',
verbose=0,
warm_start=False,
n_jobs=None,
l1_ratio=None)

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

Measuring how closely the formula fits the data

- **Penalty** {‘l1’, ‘l2’, ‘elasticnet’, ‘none’}, default=’l2’ Used to specify the norm used in the penalization. If ‘none’ (not supported by the liblinear solver), no regularization is applied.

- **C** float, default=1.0 Inverse of regularization strength; must be a positive float. Smaller values specify stronger regularization.

- **l1_ratio** float, default=None The Elastic-Net mixing parameter, with 0 <= l1_ratio <= 1. Only used if penalty='elasticnet'. Setting l1_ratio=0 is equivalent to using penalty='l2', while setting l1_ratio=1 is equivalent to using penalty='l1'. For 0 < l1_ratio <1, the penalty is a combination of L1 and L2

Choosing a method for solving the fitting problem

- **solver**{‘newton-cg’, ‘lbfgs’, ‘liblinear’, ‘sag’, ‘saga’}, default=’lbfgs’ Algorithm to use in the optimization problem. Not every solver choice will work with every penalty choice.

- **max_iter** int, default=100 Maximum number of iterations taken for the solvers to converge.