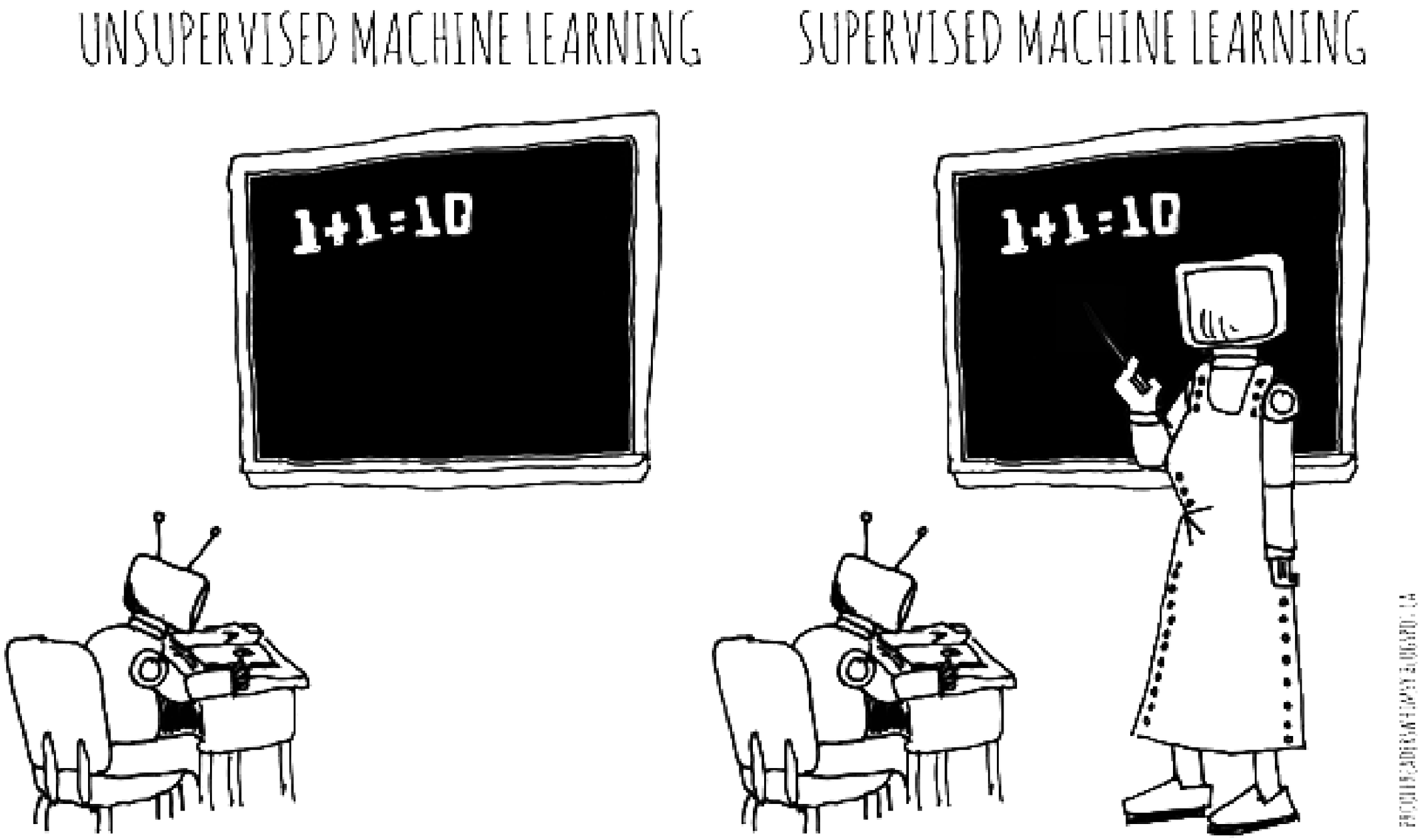
# Data Analysis
# using
# Machine Learning

Myroslav Kavatsyuk, CMST

# Unsupervised versus supervised



# Unsupervised

Data starts without labels, everything is measured

Trying to find patterns that then become labels

e.g. here are a bunch of fruit, are there groups ("clusters") of fruit that are more similar to each other than they are to the rest?

If so, maybe -- *maybe* -- they are different kinds of fruit.

**EXPLORATORY Data Analysis**

# Clustering algorithms

**Essence: *Flat* versus *hierarchical* clustering**

- **Centroid-based:** K-Means, …
- **Density-based:** MeanShift, …
- **Graph-based:** Spectral, …


***"Use the right tool for the right job"***

# Supervised

Data starts with labels AND measurements ("features")

Trying to find patterns in the measurement that are associated with labels, so that if there are new instances and features, we can predict the new label.

(e.g. a new fruit of a certain color, elongatedness, weight, acidity and sweetness -- what kind of fruit is it?)

**PREDICTIVE data analysis**

# Supervised = *trained* ML

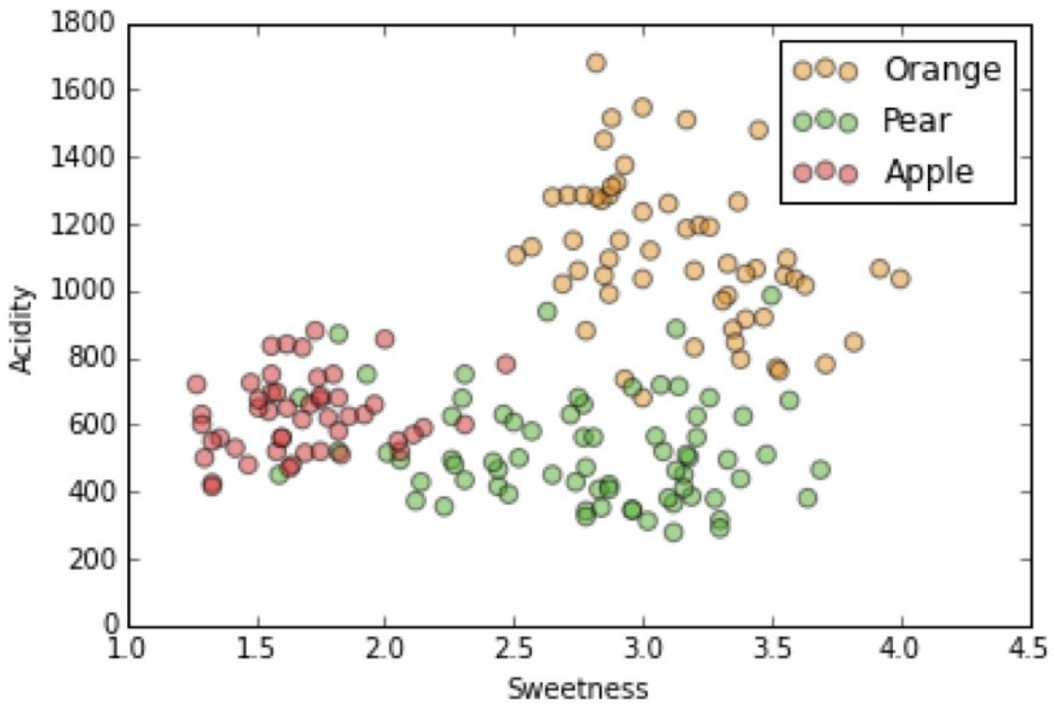
Python: ***sklearn.<alg>***
Mathematica: ***Classify[…]***

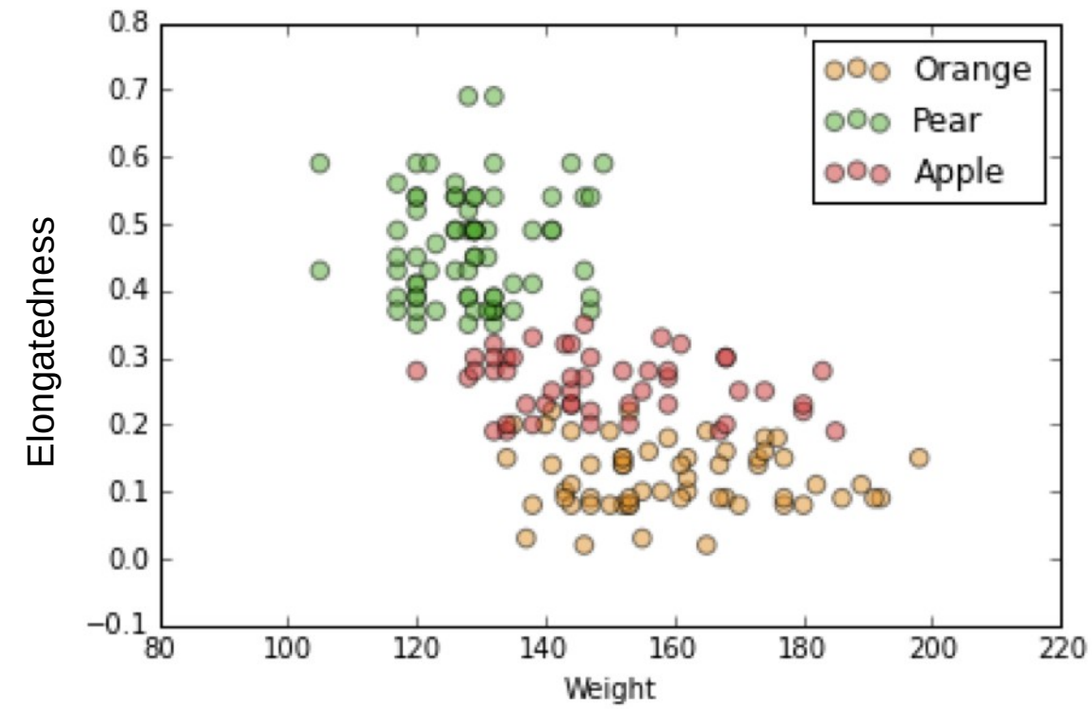
# fruit dataset + labels

| | fruit_id | fruit_name | color_id | color_name | elongatedness | weight | sweetness | acidity |
|---|---|---|---|---|---|---|---|---|
| **169** | 1 | orange | 4 | orange | 0.08 | 144 | 3.58 | 1290 |
| **170** | 1 | orange | 5 | red | 0.11 | 182 | 3.58 | 1295 |
| **171** | 1 | orange | 4 | orange | 0.11 | 144 | 3.59 | 1035 |
| **172** | 1 | orange | 4 | orange | 0.09 | 143 | 3.63 | 1015 |
| **173** | 2 | pear | 6 | yellow | 0.47 | 123 | 3.64 | 380 |
| **174** | 2 | pear | 6 | yellow | 0.56 | 126 | 3.69 | 465 |
| **175** | 1 | orange | 5 | red | 0.11 | 189 | 3.71 | 780 |
| **176** | 1 | orange | 4 | orange | 0.19 | 144 | 3.82 | 845 |
| **177** | 1 | orange | 5 | red | 0.09 | 191 | 3.92 | 1065 |
| **178** | 1 | orange | 2 | brown | 0.15 | 152 | 4.00 | 1035 |

We have three fruits: orange, apple and pear.

We know their acidity, sweetness, weight, elongatedness and color.




| | | color name | | | | | |
|---|---|---|---|---|---|---|---|
| | | blue | brown | green | orange | red | yellow |
| fruit name | apple | 3 | 1 | 15 | 0 | 16 | 14 |
| | orange | 0 | 8 | 1 | 37 | 13 | 0 |
| | pear | 2 | 12 | 9 | 3 | 2 | 43 |

Remember, some of our feature assignment was done by colorblind people, adding "noise".

A label is just a column, or feature,
or variable like all the others.

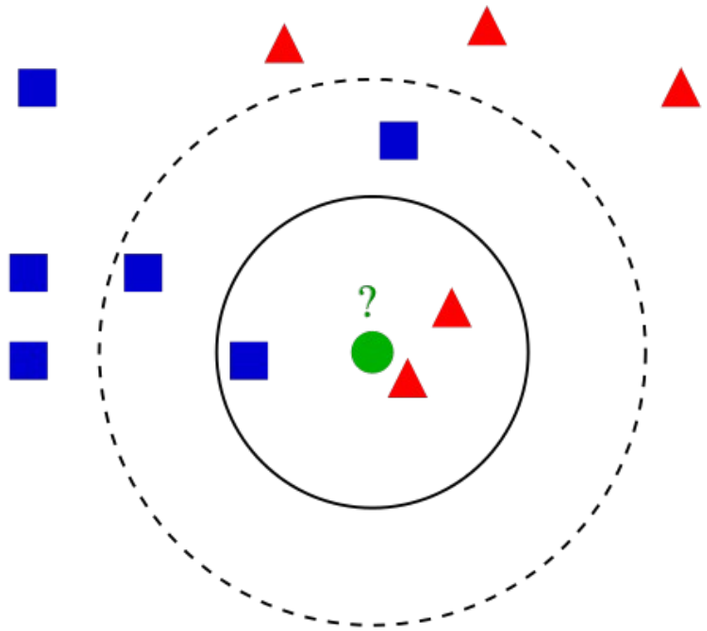
The only difference is the variable is unknown for new instances; it must be predicted

If the label is a category (e.g. fruit, or even color), we call Supervised Learning "classification".

If it is continuous and numeric (e.g. if we wanted to predict how sweet a new fruit will be), we call it "regression".

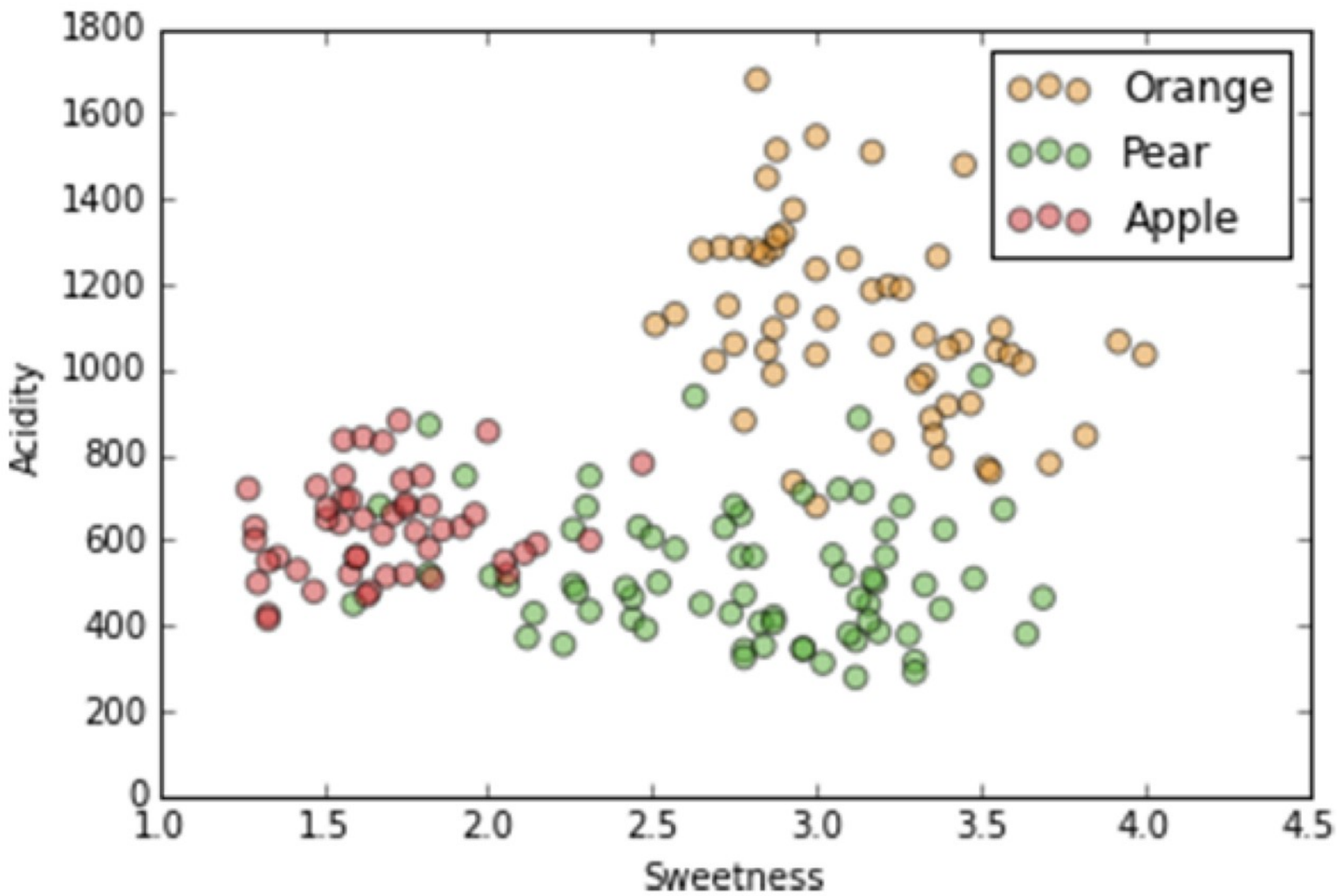
# Classifier No. 1
# -
# “*k*-nearest neighbours (*k*NN)”

- Easy to understand, use, and implement
- The classification algorithm = training data (instance-based or lazy learning)
- Falls in the category of “density estimators”



# Again, we'll use two dimensions of our dataset for ease of visualization.

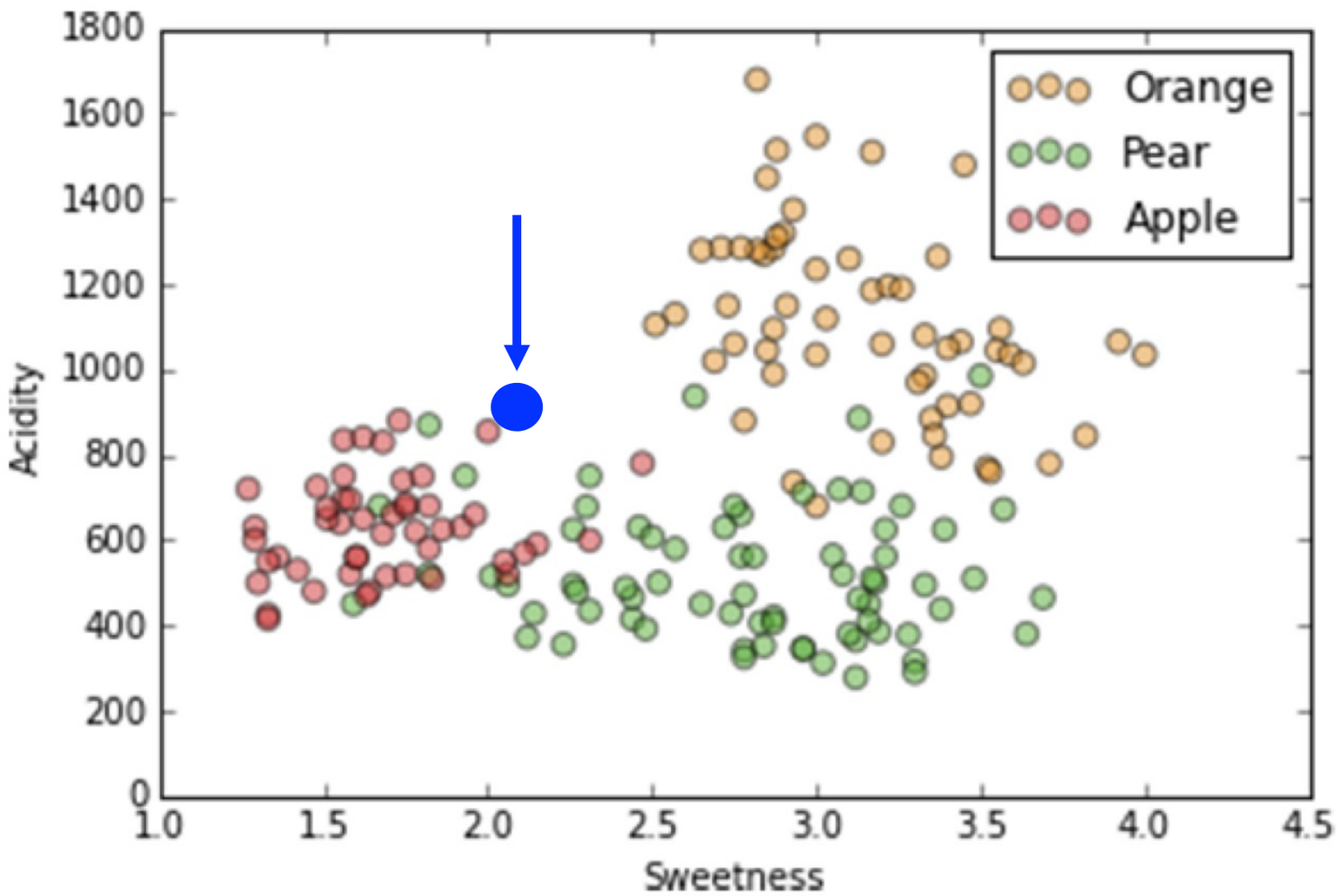
Same two dimensions, acidity and sweetness, but with axes flipped so we "forget" our clusters.




We have this data with these labels...

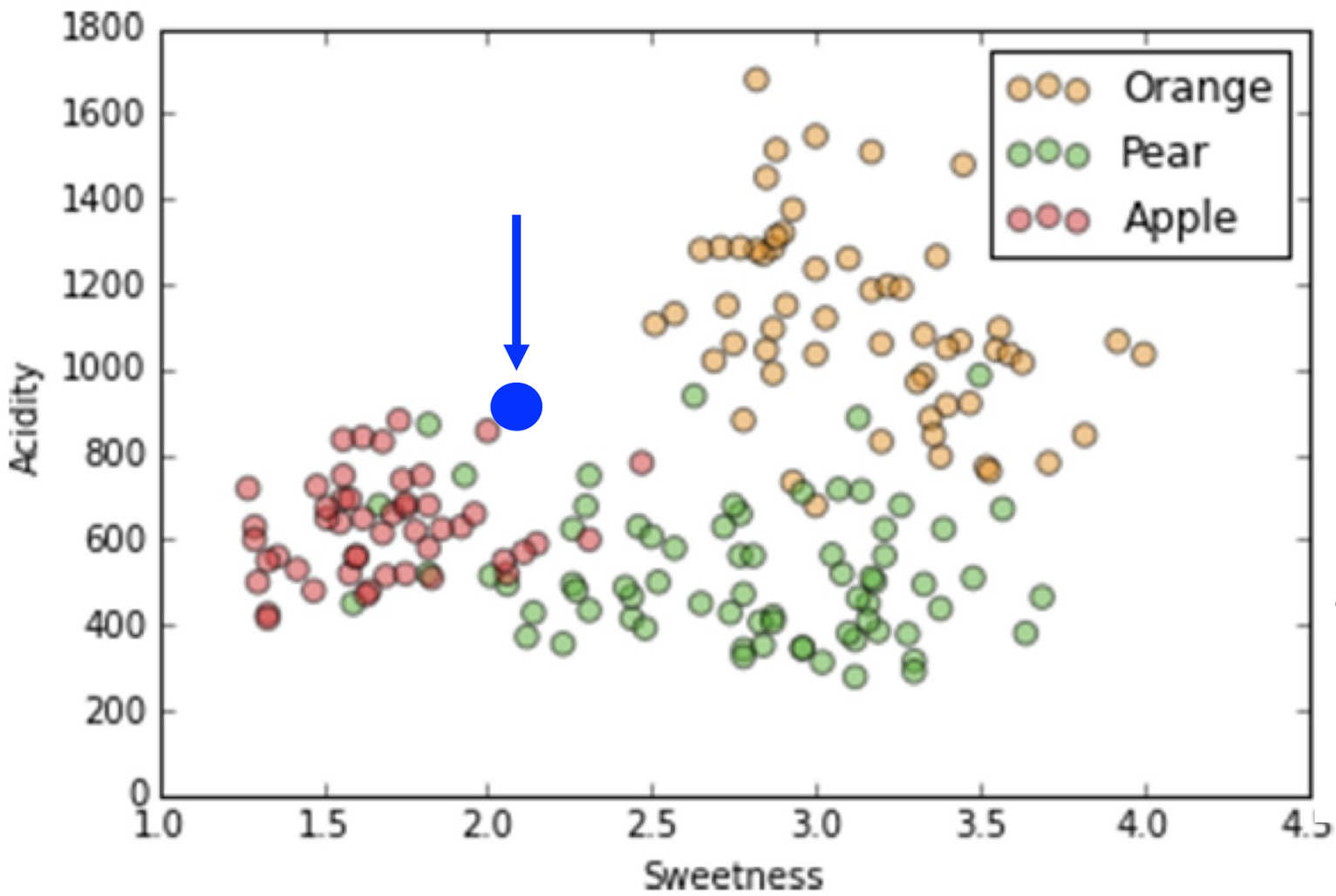
# Again, we'll use two dimensions of our dataset for ease of visualization.

Same two dimensions, acidity and sweetness, but with axes flipped so we "forget" our clusters.




If we have a new data point (in blue), what label should we assign it?

There are many classification algorithms; often it doesn't matter which one you use. Except when it does matter.



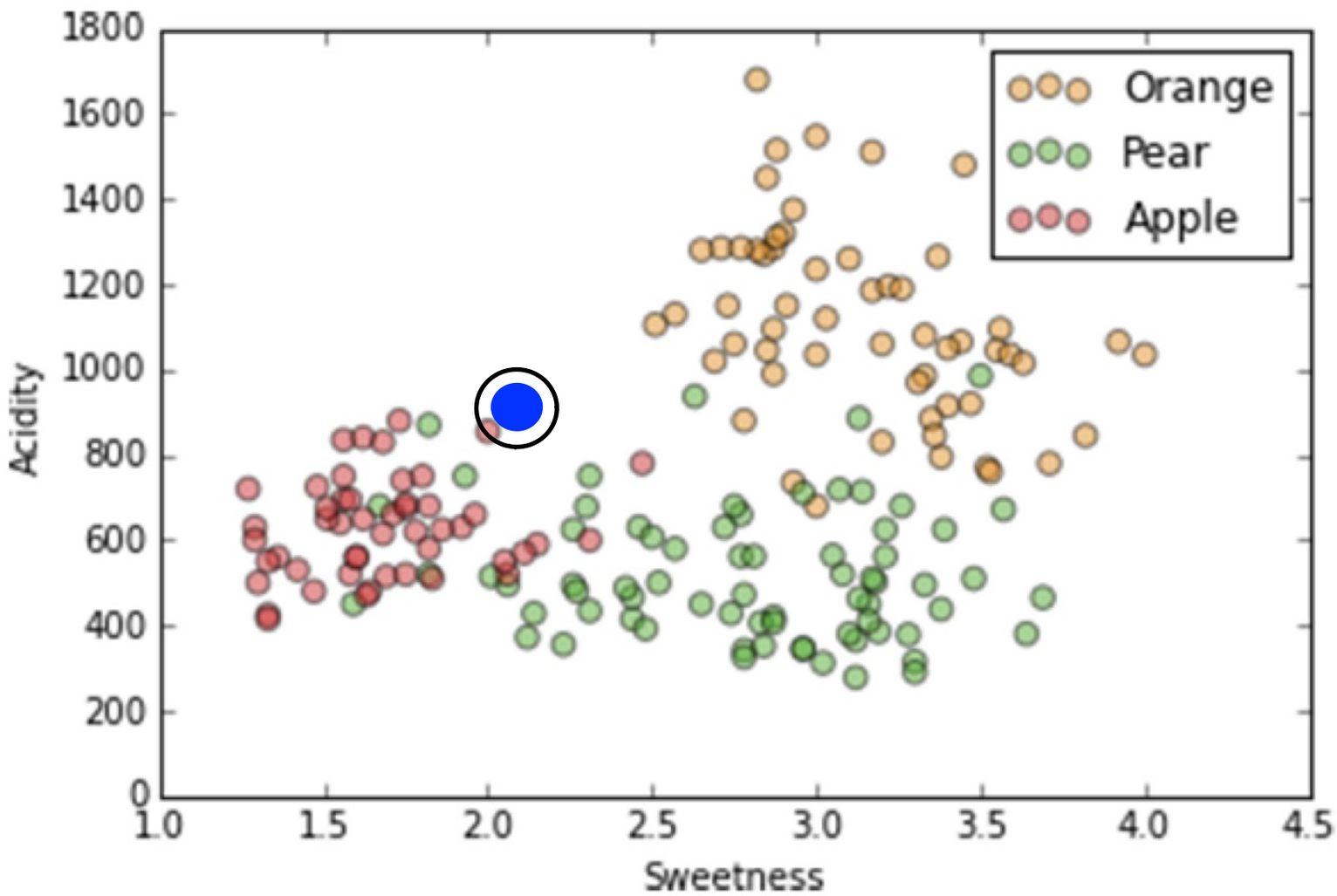

Conceptually the easiest one:

## K-Nearest Neighbor

**(aka “kNN”)**

*Note: the "k" in k-nearest neighbor stands for something different than the "k" in k-means; they're both an integer parameter of the algorithm, however.*

# kNN in action…

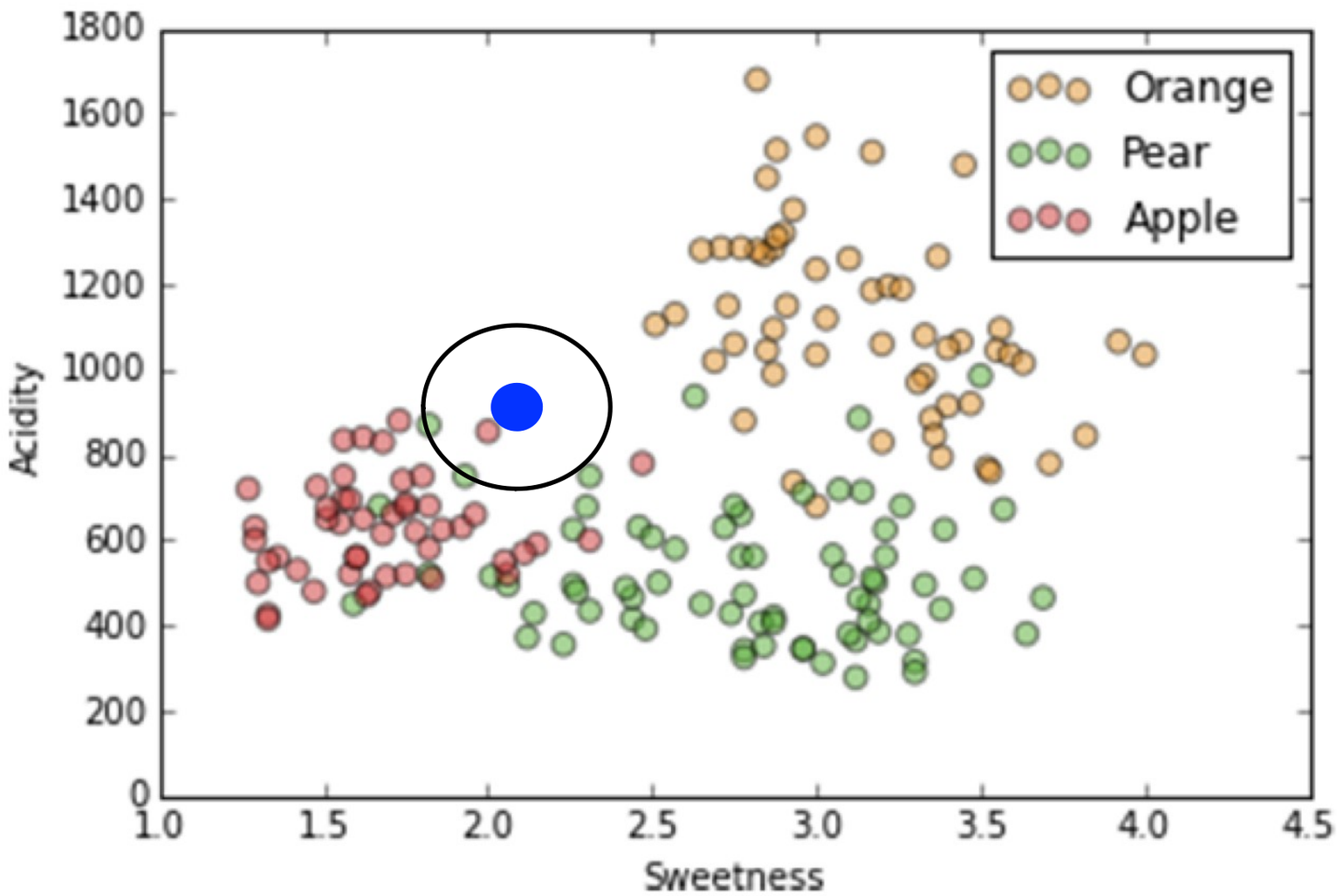



k=1

**Label is determined by the single closest neighbor:**

**Apple**

# kNN in action…




k=3

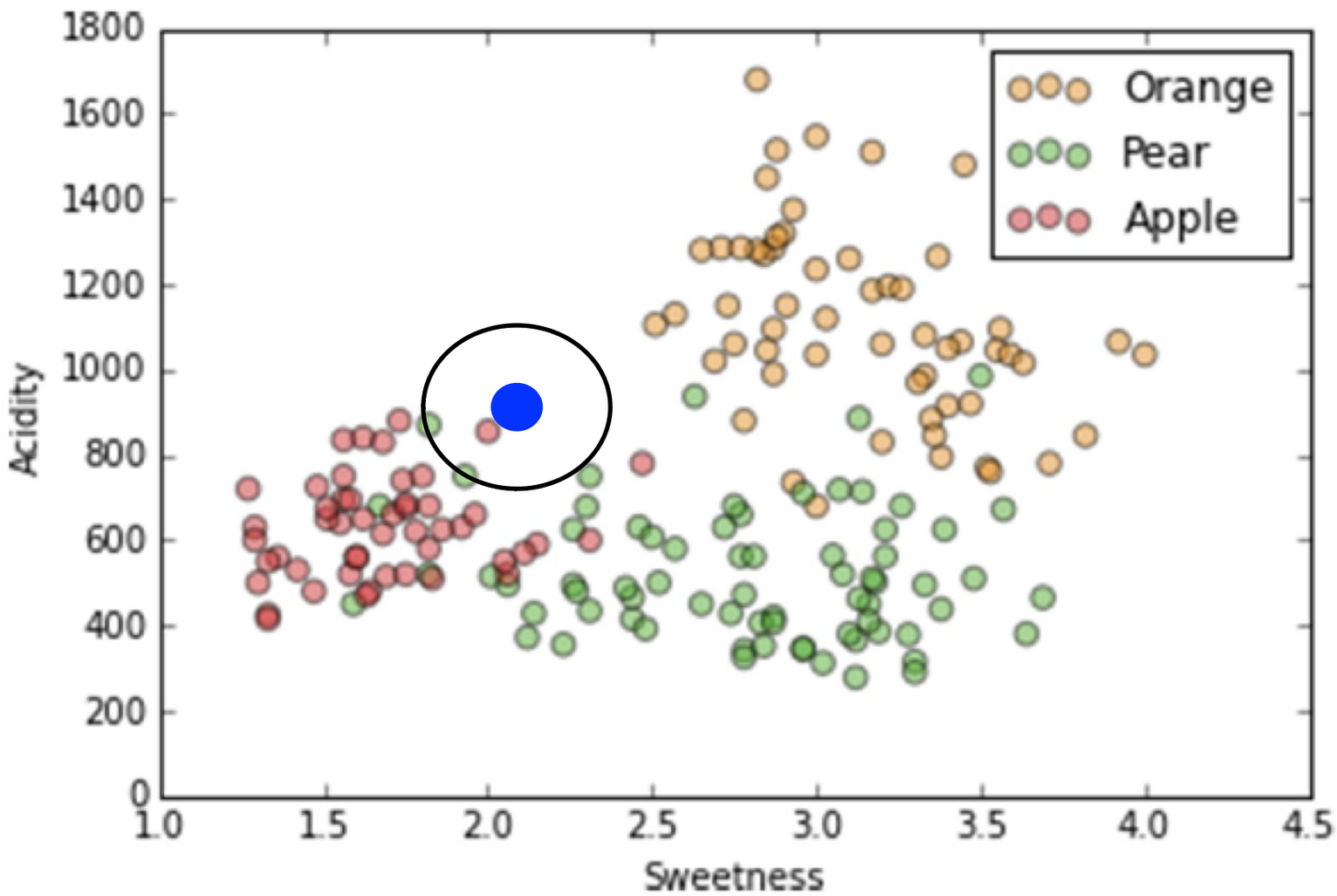
**Label is determined by majority vote:**

**Pear**

(2 pears+1 apple)

(It's usually a good idea for k to be an odd number to minimize ties)

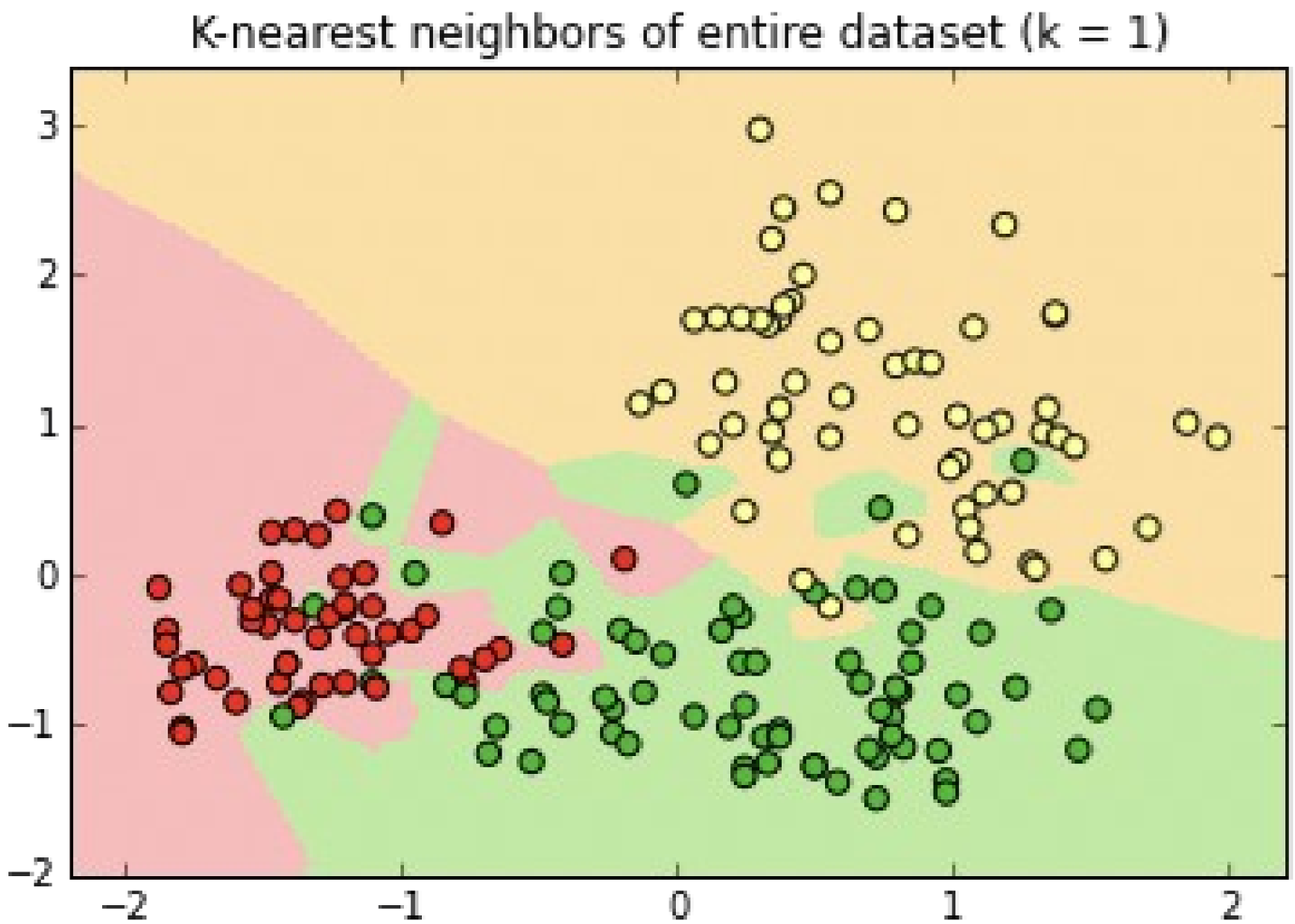
# kNN in action…




We now have the exact same problem we had with K-Means:

What's the best value of k?

# kNN in action…



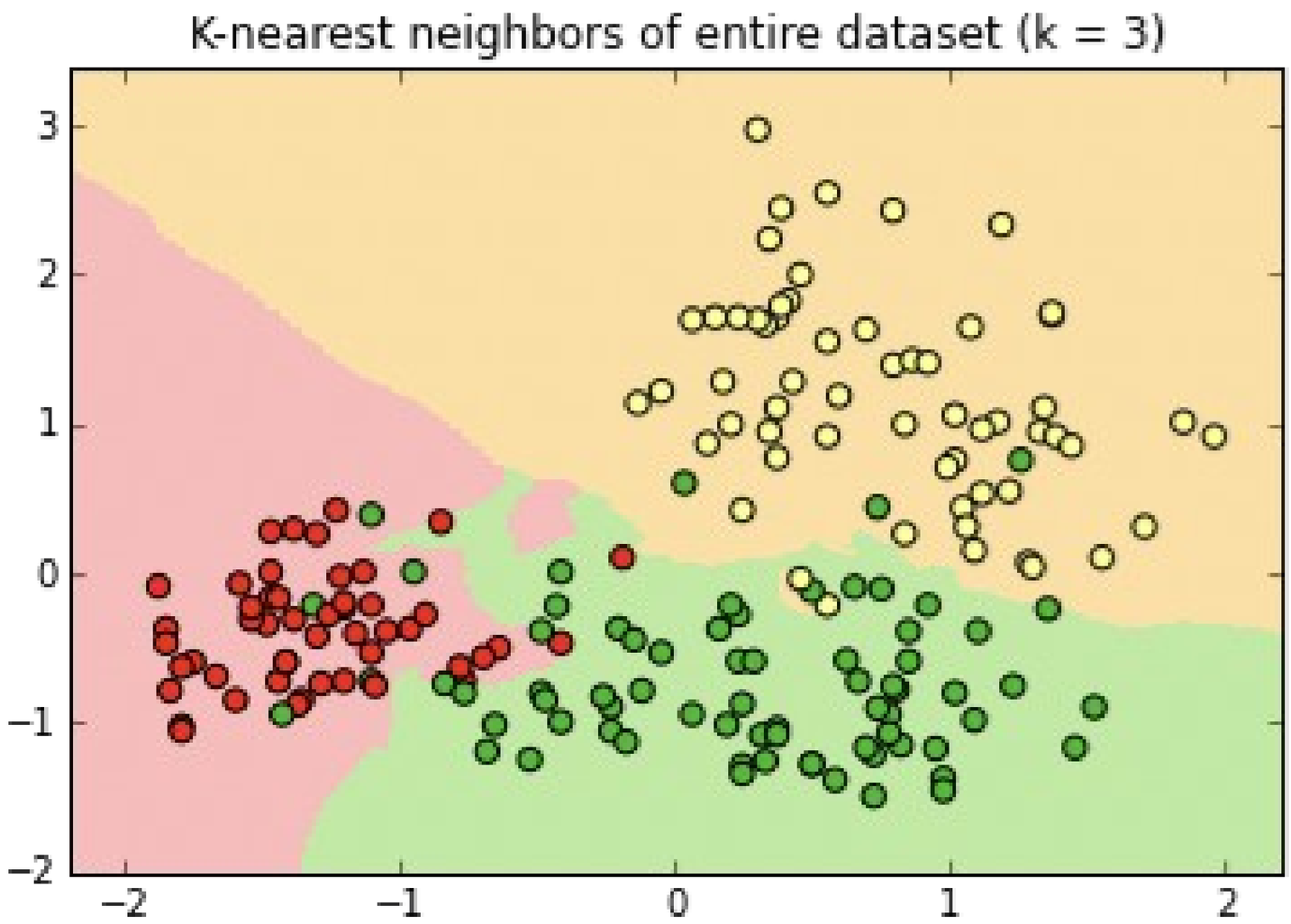

The classifier produces “decision surfaces” that determine the label of new data.

Any new point in, e.g. the red region, would be classified an apple.

If k=1, the single closest point determines the new label.

Every point has a "halo" around it.

# kNN in action…

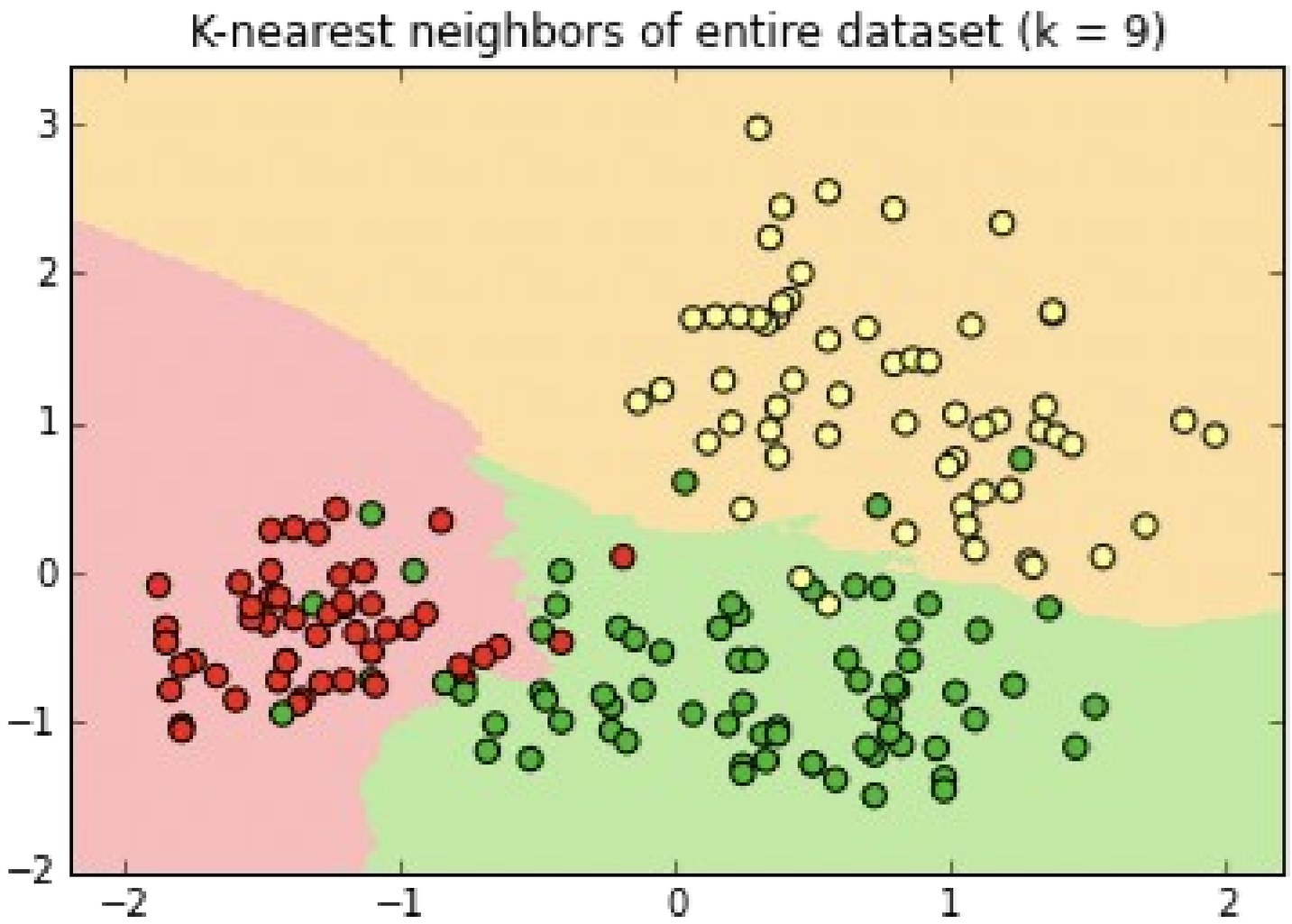



The classifier produces “decision surfaces” that determine the label of new data.

Any new point in, e.g. the red region, would be classified an apple.

If k=3, the new label is determined by a two-out-of-three "majority vote" of nearest neighbors.
The decision surface is simpler; some points do not have "halos" around them

# kNN in action…

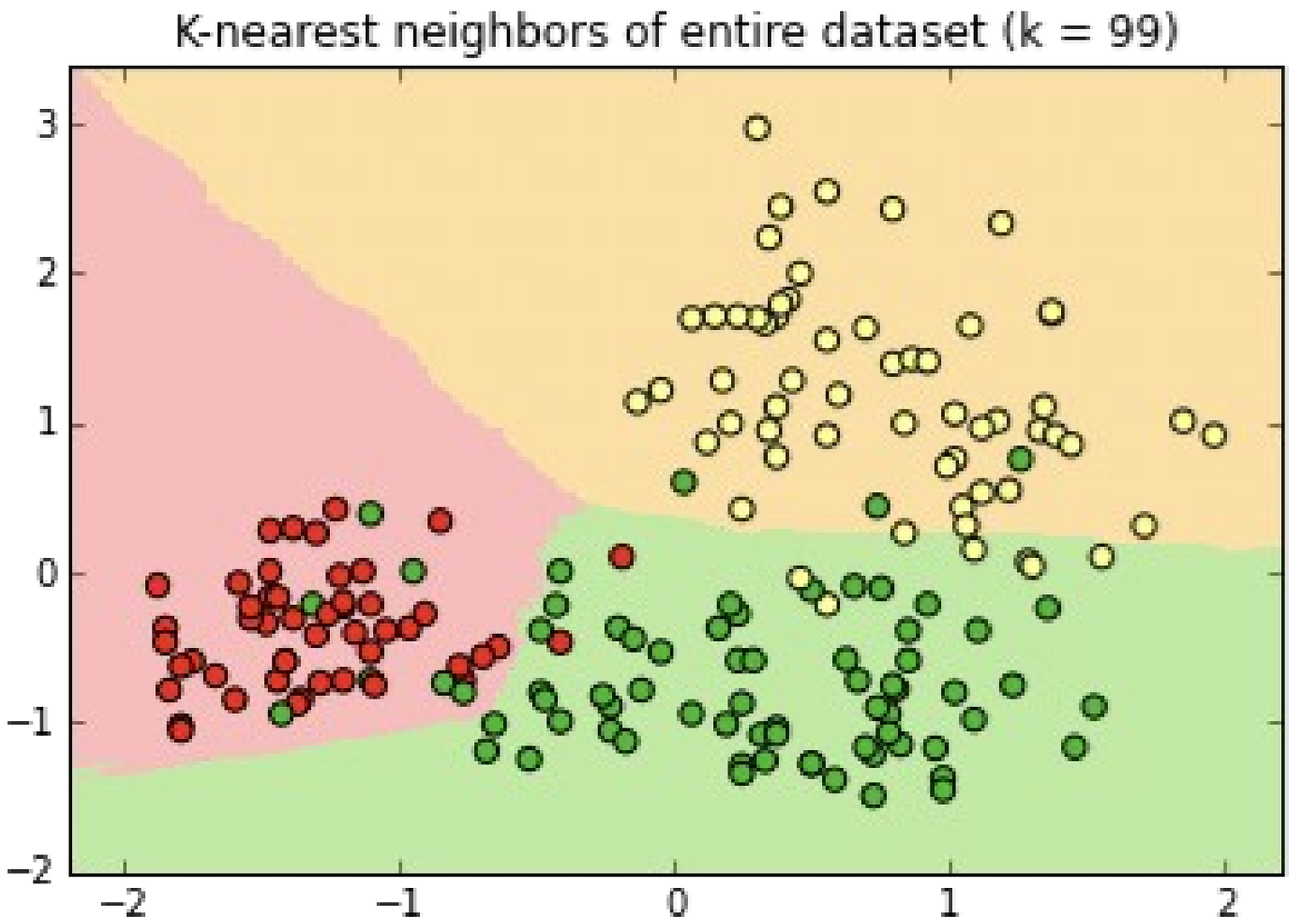



The classifier produces “decision surfaces” that determine the label of new data.

Any new point in, e.g. the red region, would be classified an apple.

At k=9 there are no halos, just three decision surfaces with sometimes jagged borders.

# kNN in action…

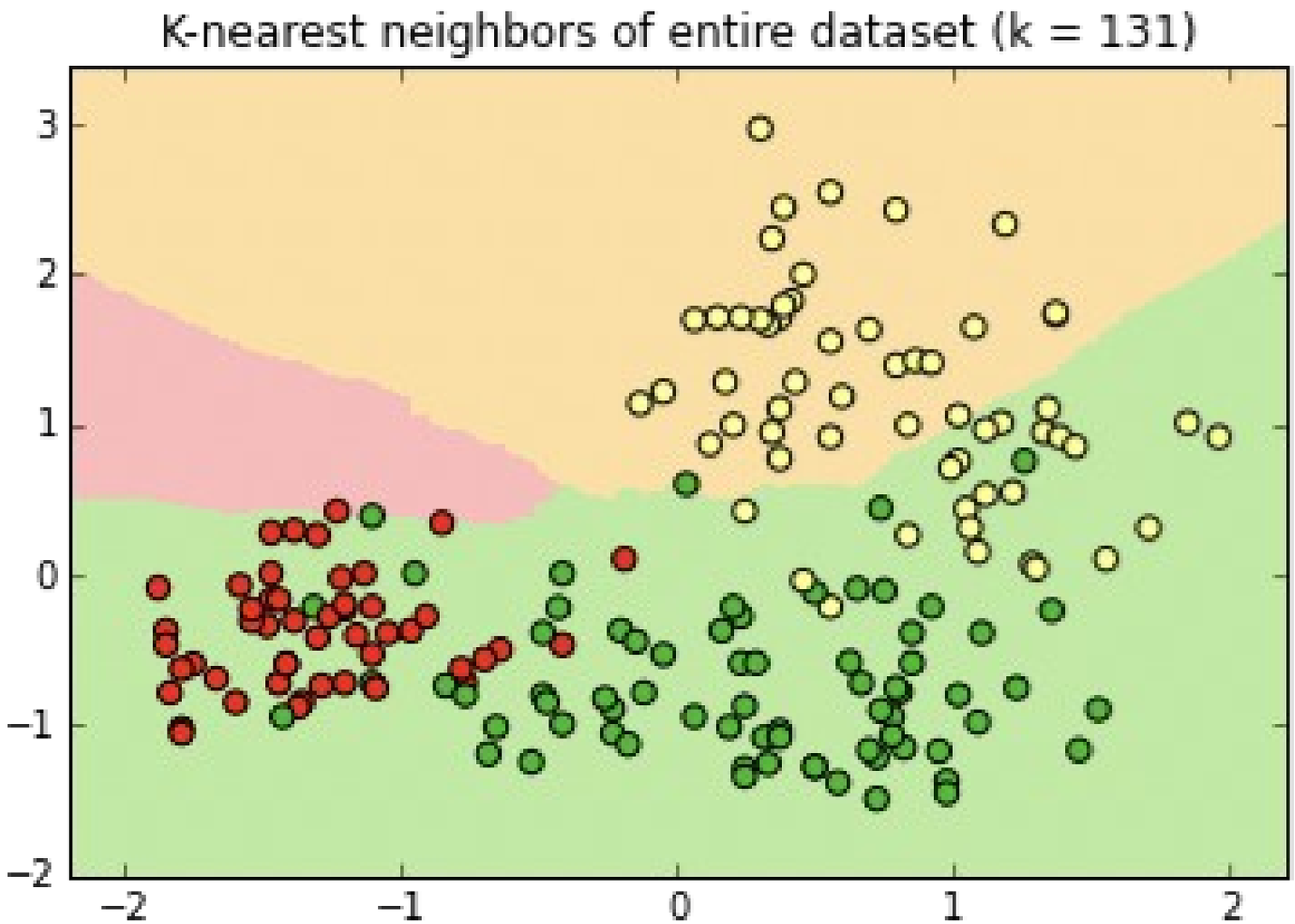



The classifier produces “decision surfaces” that determine the label of new data.

Any new point in, e.g. the red region, would be classified an apple.

At k=99 , the decision surface edges are relatively smooth.
Note the extension of the green surface in the lower left.

# kNN in action…

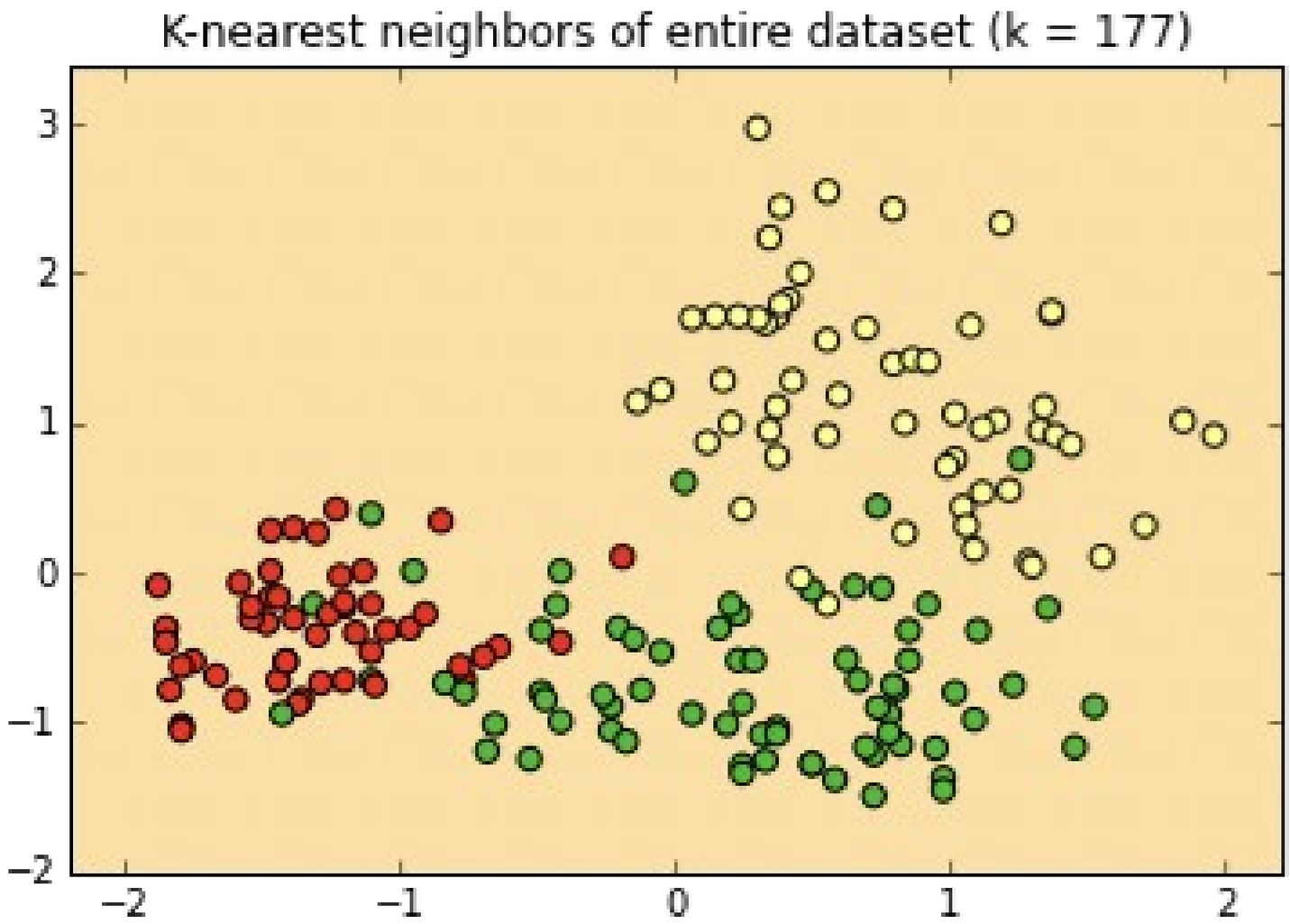



The classifier produces “decision surfaces” that determine the label of new data.

Any new point in, e.g. the red region, would be classified an apple.

At k=131 , the apple decision surface has shrunk considerably into a space with no instances.

# kNN in action…




The classifier produces “decision surfaces” that determine the label of new data.

Any new point in, e.g. the red region, would be classified an apple.

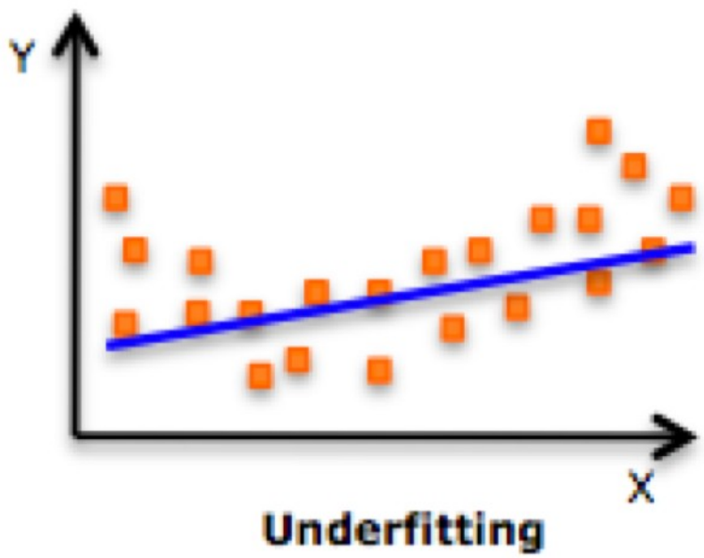
At its maximum value, k=177 , every new label is "orange" because there are more orange instances than apple or pear instance in the dataset.

How do we choose our value of k?

This is called "fitting" the algorithm.

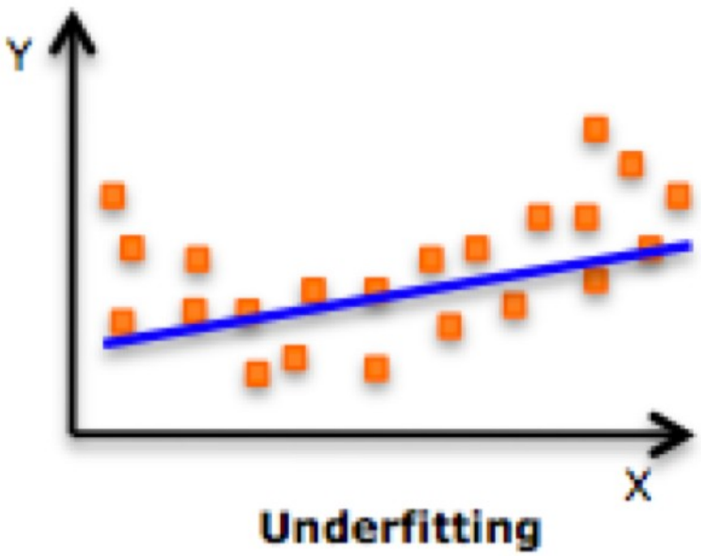
Think of a classifier as a function approximation. There is a "true" function that produced the labels, that we will never know but we try to approximate with a model.

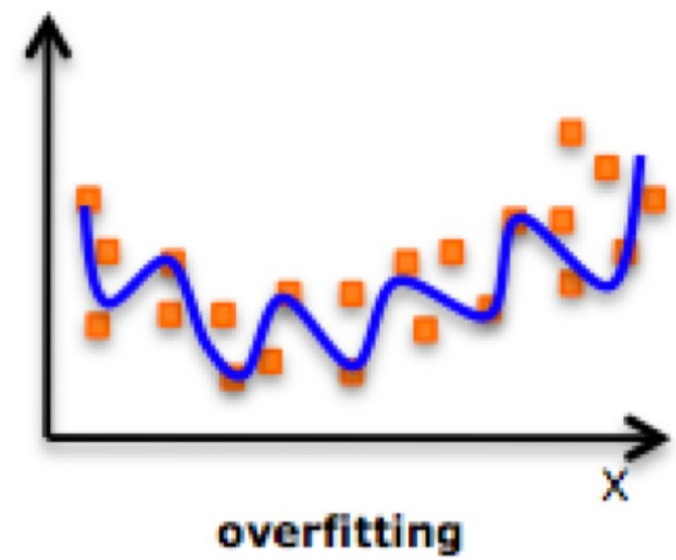
We can visualize this with a simple model, a best-fit regression of a curved dataset made of signal and noise.




If our model is too simple, we are "underfitting", i.e. the model does not adequately represent the signal.

We can visualize this with a simple model, a best-fit regression of a curved dataset made of signal and noise.

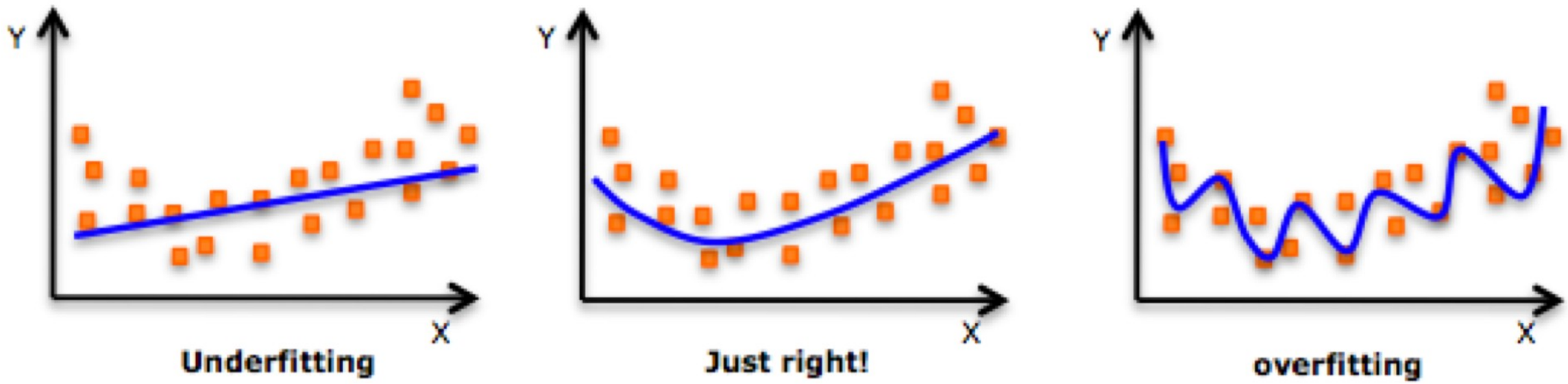



If our model is too simple, we are "underfitting", i.e. the model does not adequately represent the signal.

If our model is too complex, we are "overfitting", i.e. we are fitting noise instead of signal.

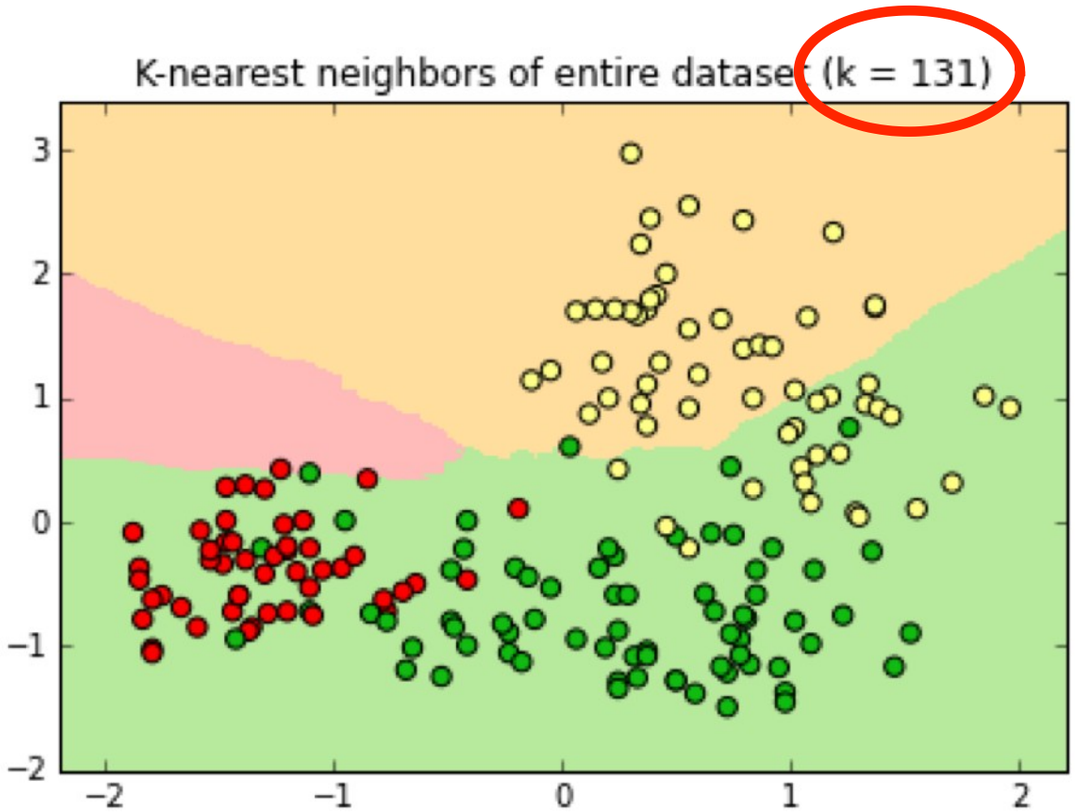
We can visualize this with a simple model, a best-fit regression of a curved dataset made of signal and noise.

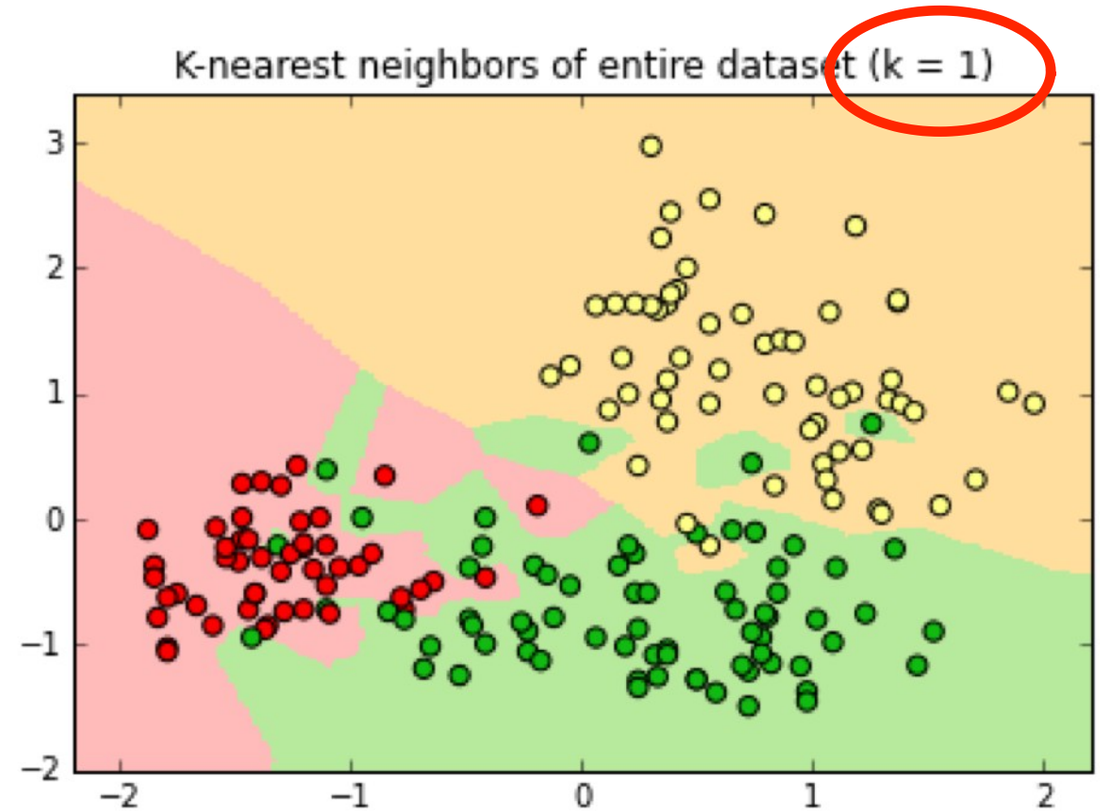


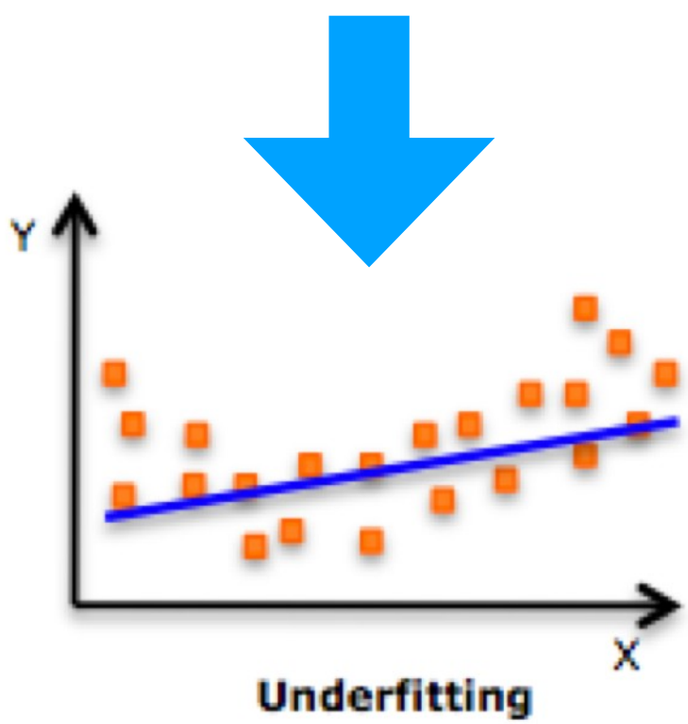

If our model is too simple, we are "underfitting", i.e. the model does not adequately represent the signal.

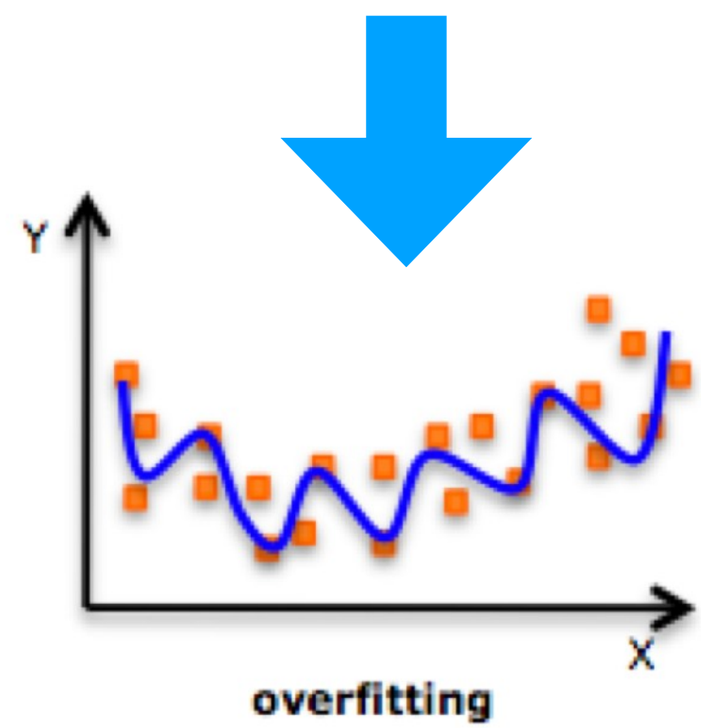
If our model is too complex, we are "overfitting", i.e. we are fitting noise instead of signal.

We try to fit as much signal as possible, without noise.


K-nearest neighbors of entire dataset (k = 131)
3
2
1
0
−1
−2
−2
−1
0
1
2


K-nearest neighbors of entire dataset (k = 1)
3
2
1
0
−1
−2
−2
−1
0
1
2


Y
X
Underfitting


Y
X
overfitting

Machine Learning uses special terminology:

# "the bias-variance tradeoff"

underfitting = bias

i.e. the model is biased and does not reflect all the signal

overfitting = variance

i.e. the model reflects the noise, or variance, not the signal.

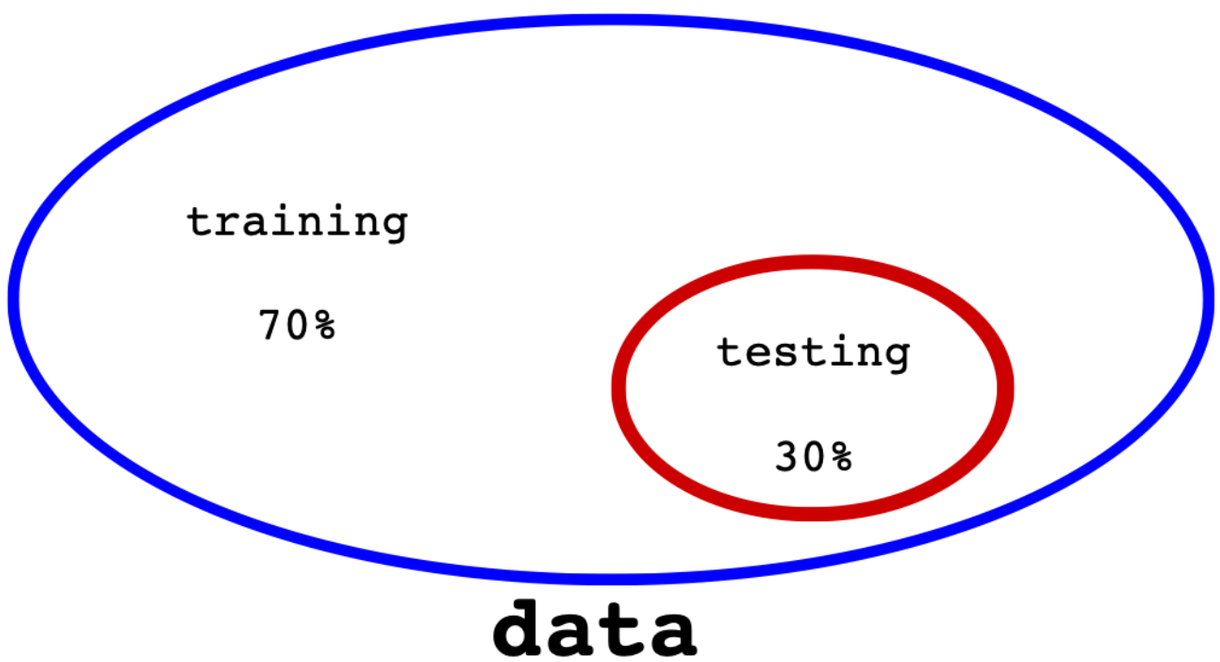
How do we choose the right model to balance bias and variance?

(in k-nearest neighbor, how do we choose the right k?)

How do we choose the right model to balance bias and variance?

(in k-nearest neighbor, how do we choose the right k?)

By randomly sequestering part of our data as a **testing set**; the remaining data becomes our **training set.**



**Always keep your training and testing data separate.**

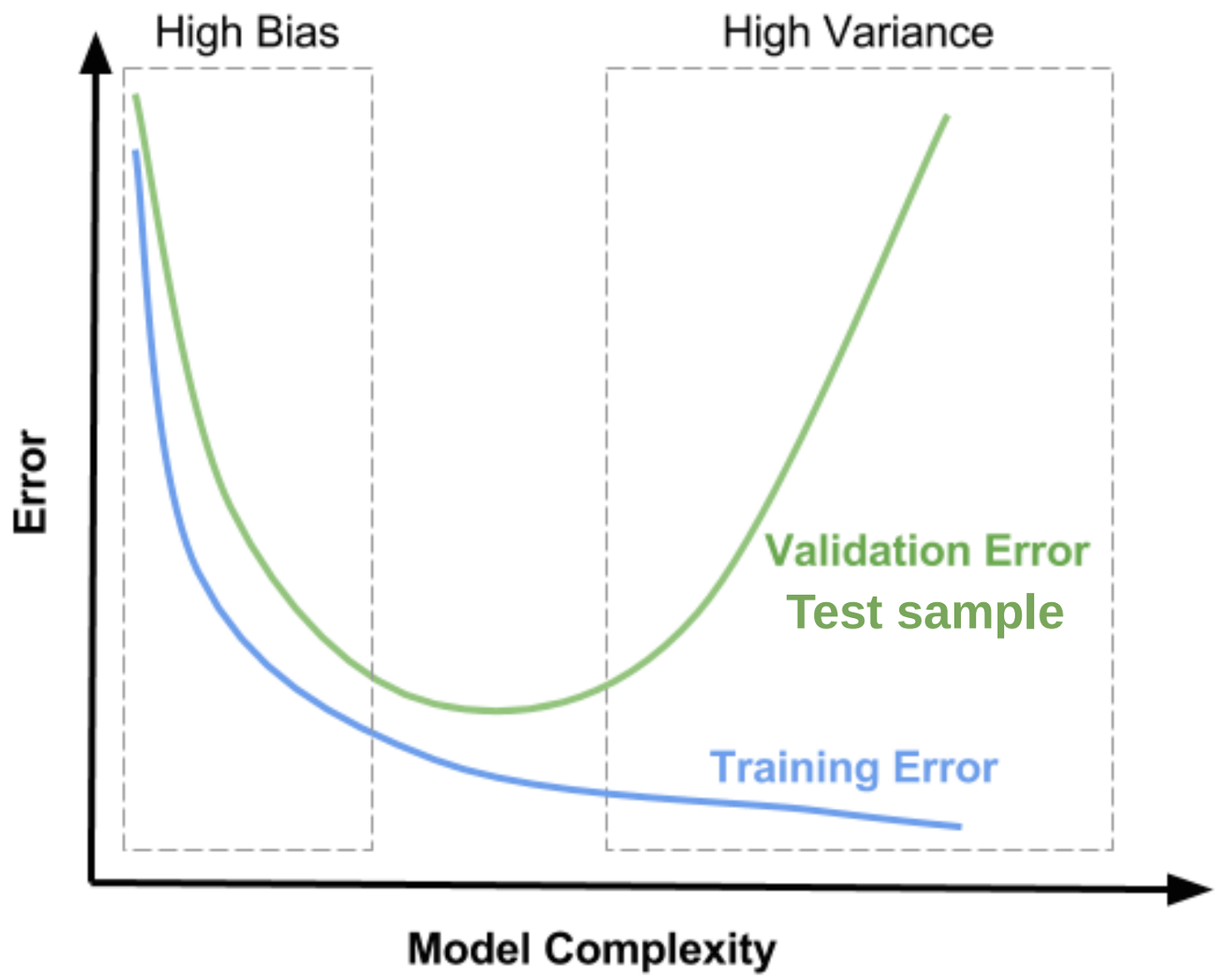
If you "cheat", you're only cheating yourself
out of a valid model.

**Always keep your training and testing data separate.**

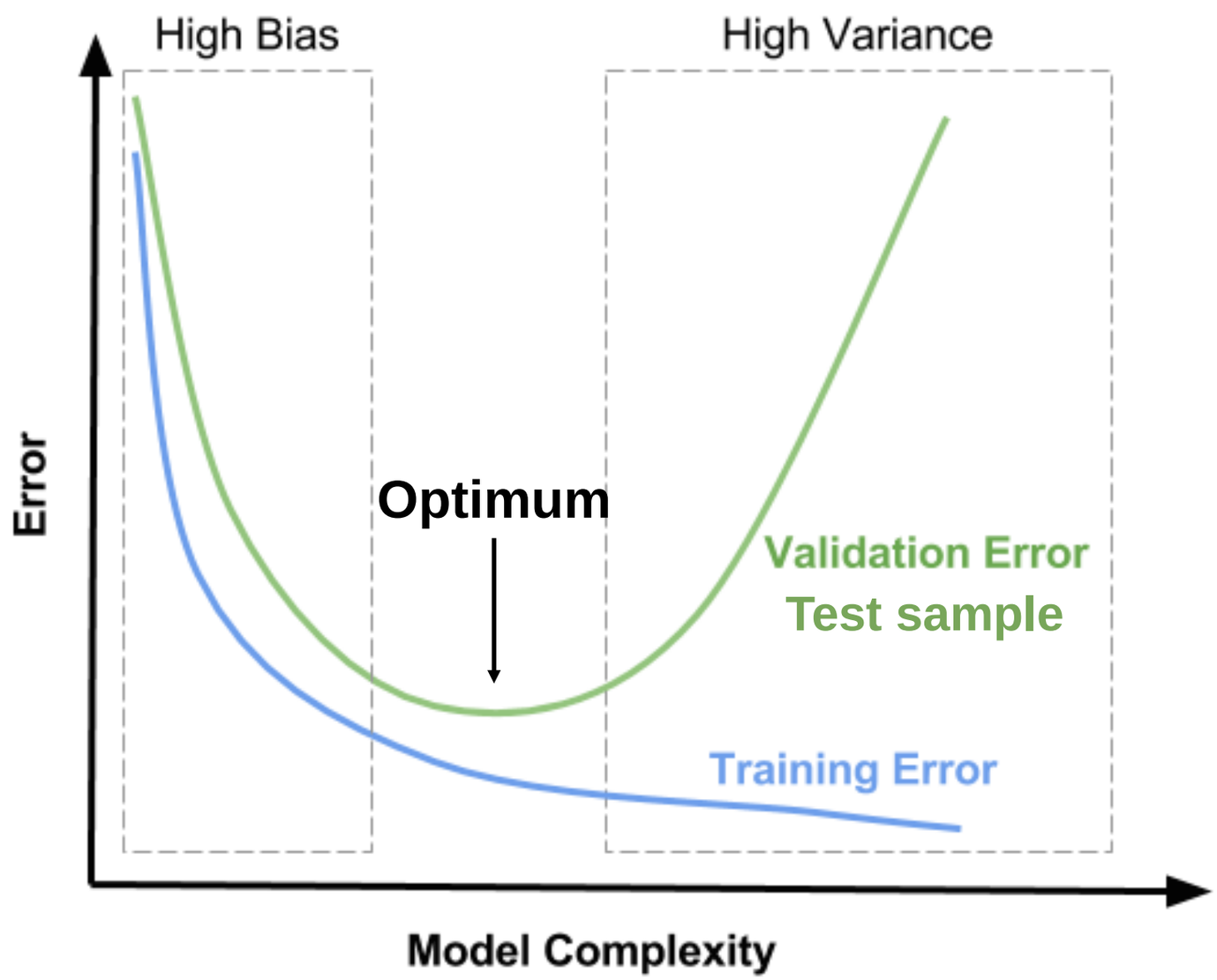
If you "cheat", you're only cheating yourself
out of a valid model.

A common approach is to run the same algorithm many times with different randomly selected training sets; this is called cross-validation.

# What does testing data sequestration tell you?

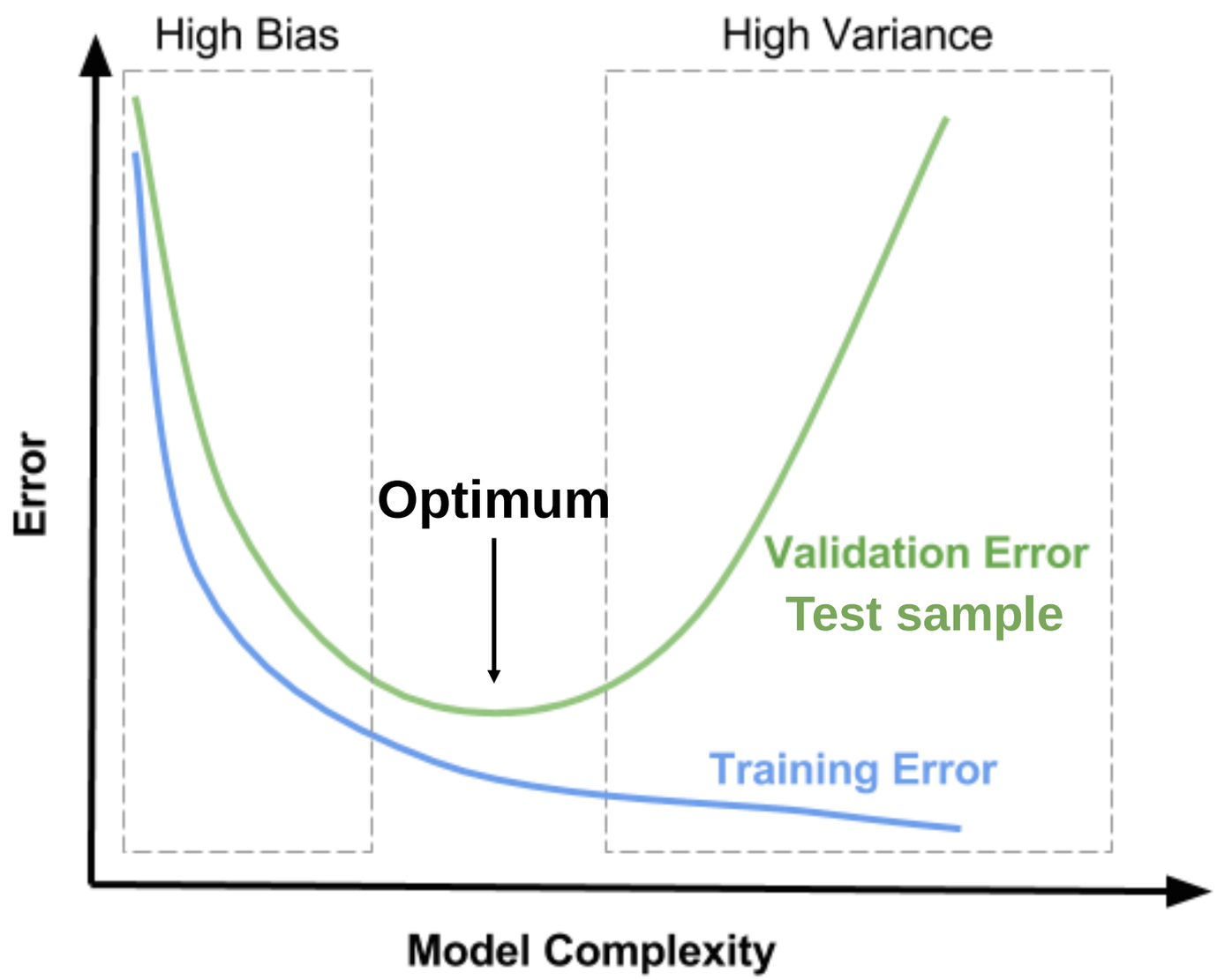


# What does testing data sequestration tell you?

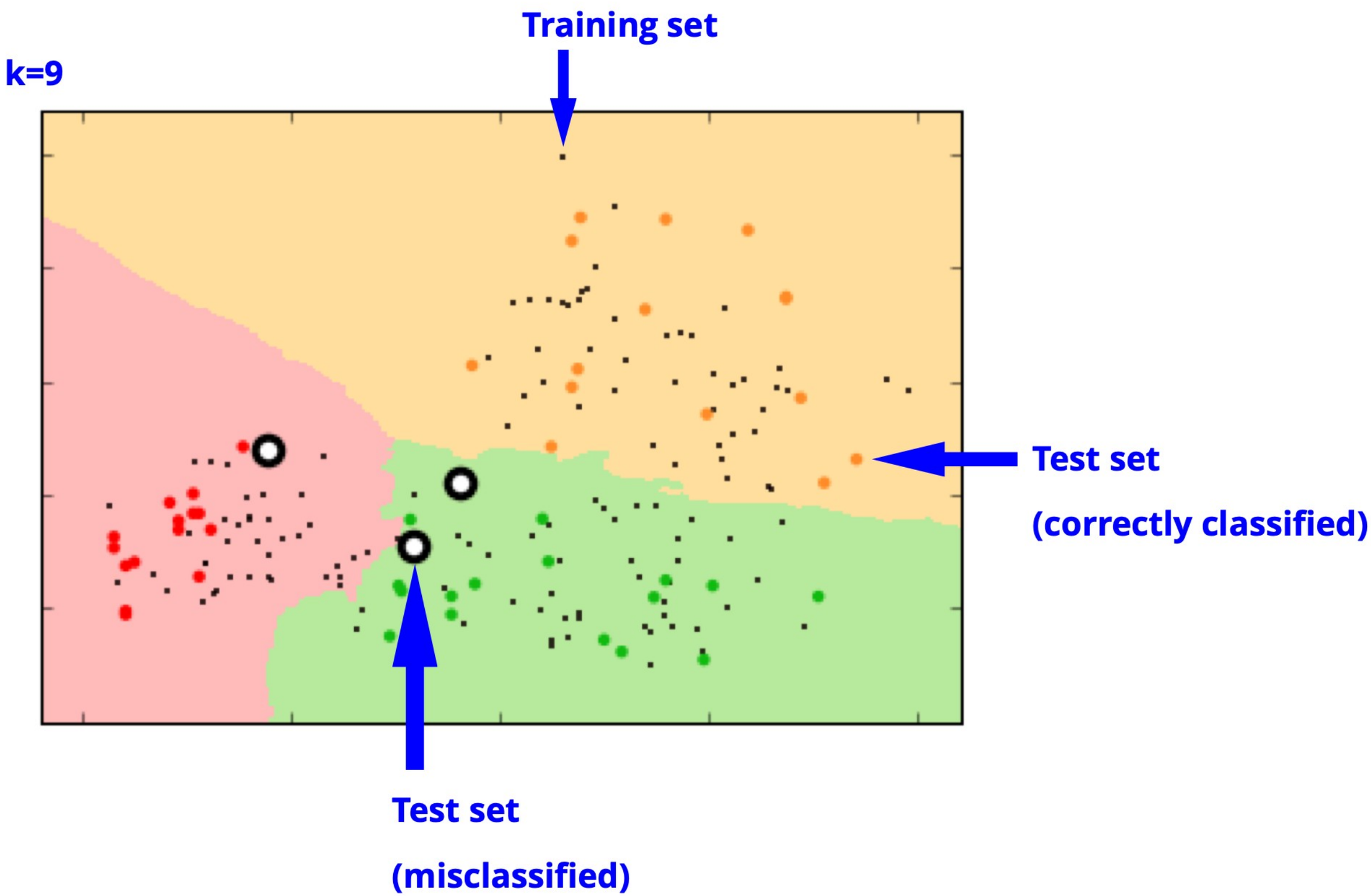



When the error rate in the test sample is at a minimum, we are fitting our model well.

kNN is a good algorithm to show bias-variance because the value of k profoundly affects the model's complexity

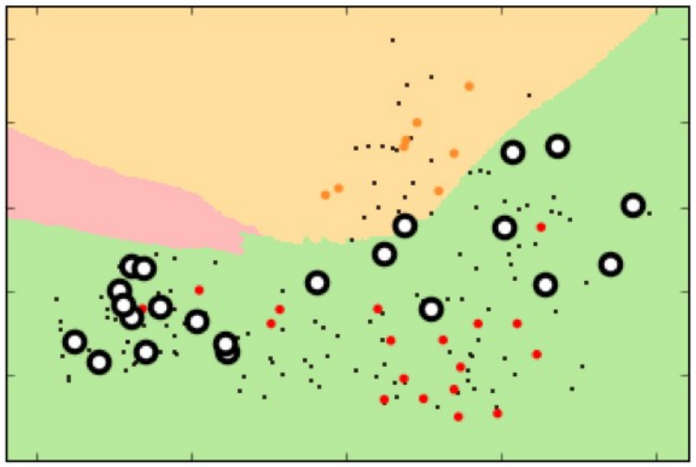



high k low k

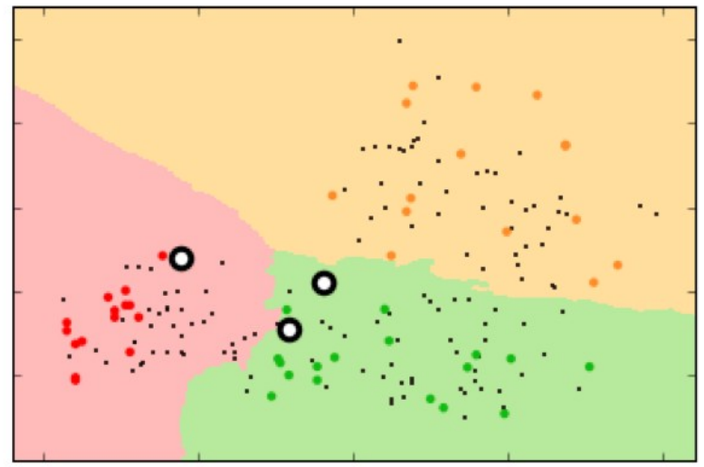

Training set
k=9
Test set
(correctly classified)
Test set
(misclassified)

One randomly chosen 70% training set

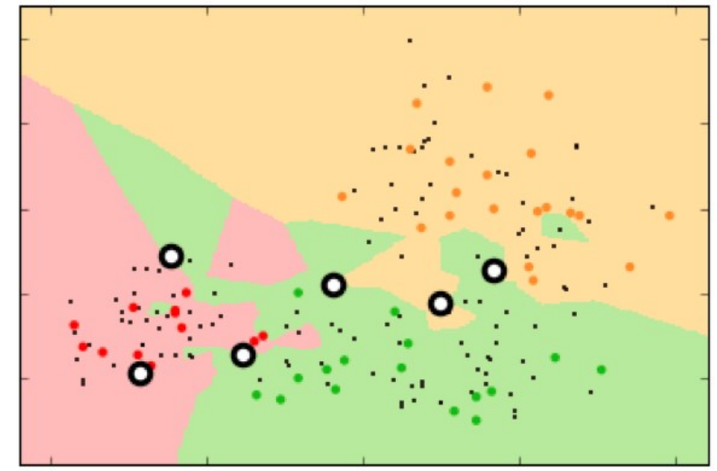
k=99

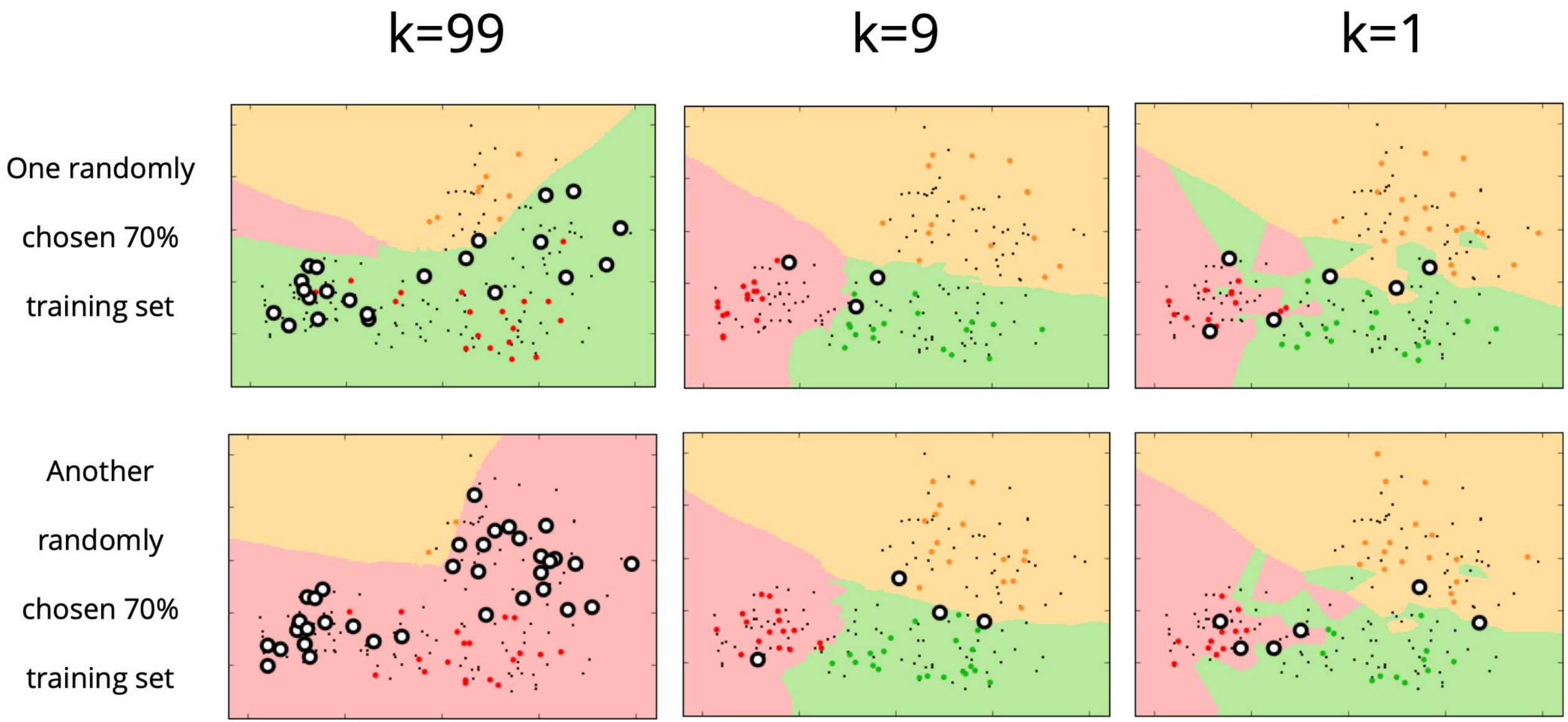


k=9



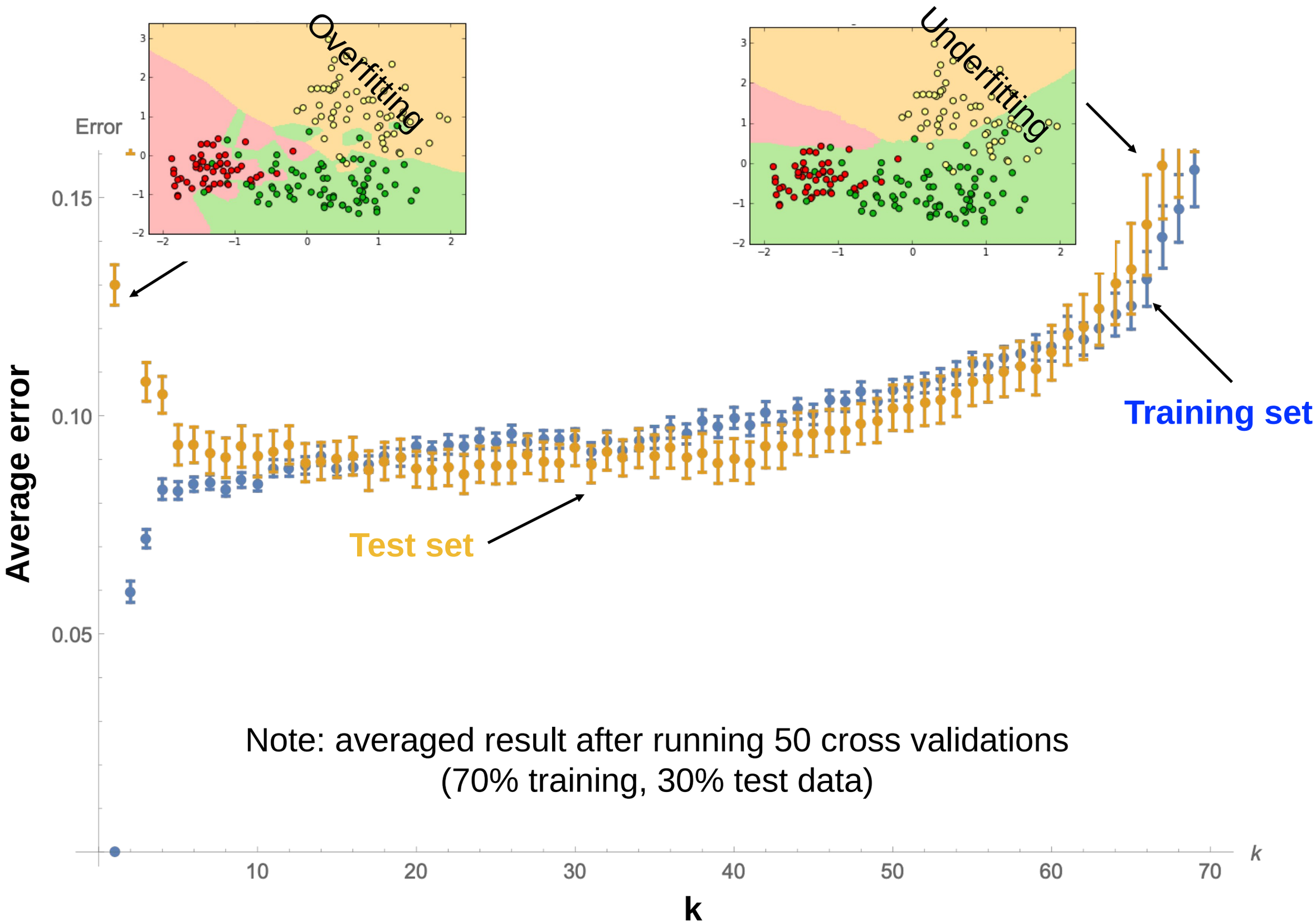
k=1






Proper fitting not only minimizes the error rate, it produces more reproducible decision surfaces/models (compare the shapes of the green areas in each replicate for the three conditions)

**Features:** *sweetness, acidity*

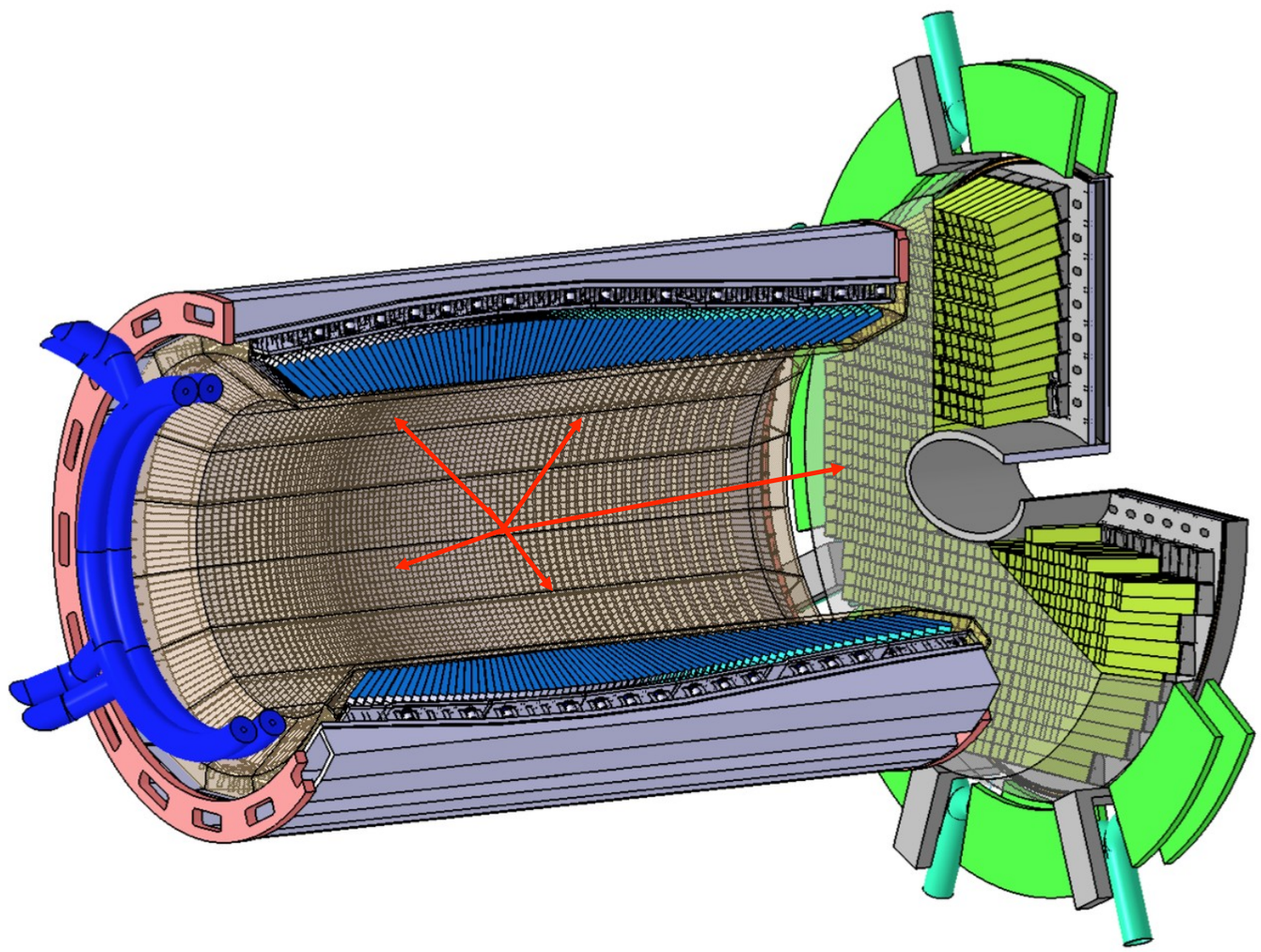


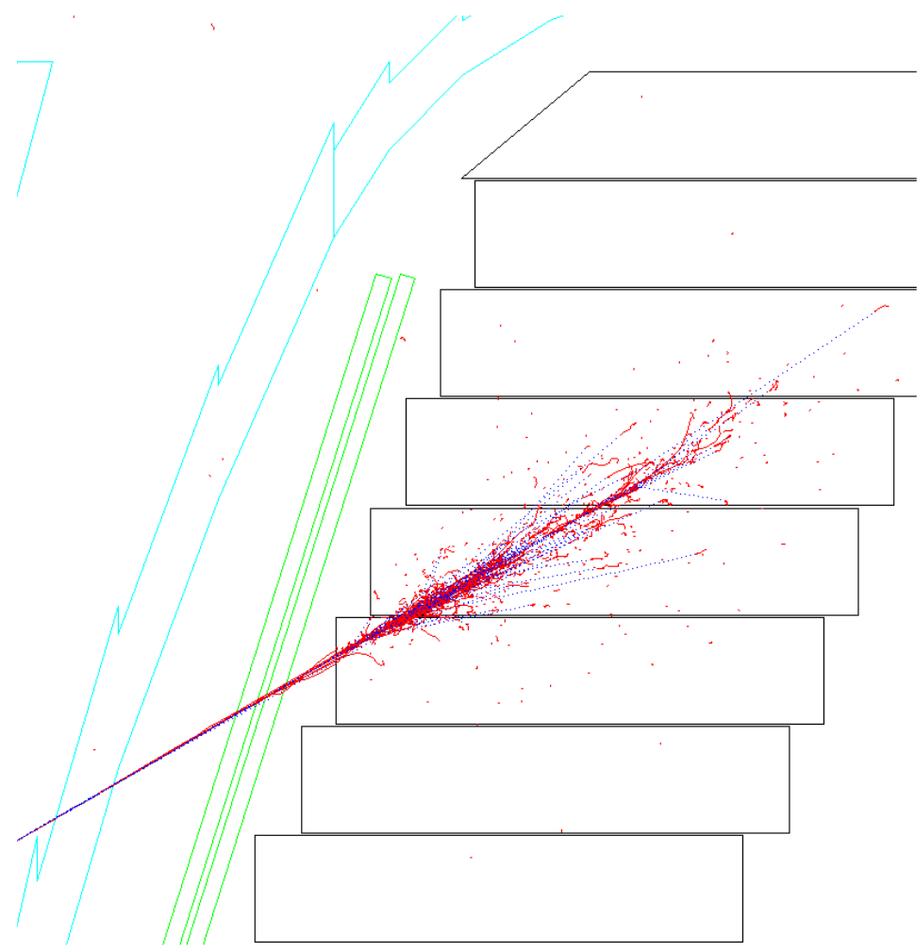

Note: averaged result after running 50 cross validations
(70% training, 30% test data)

# Demo…

# Assignment IV: Machine Learning

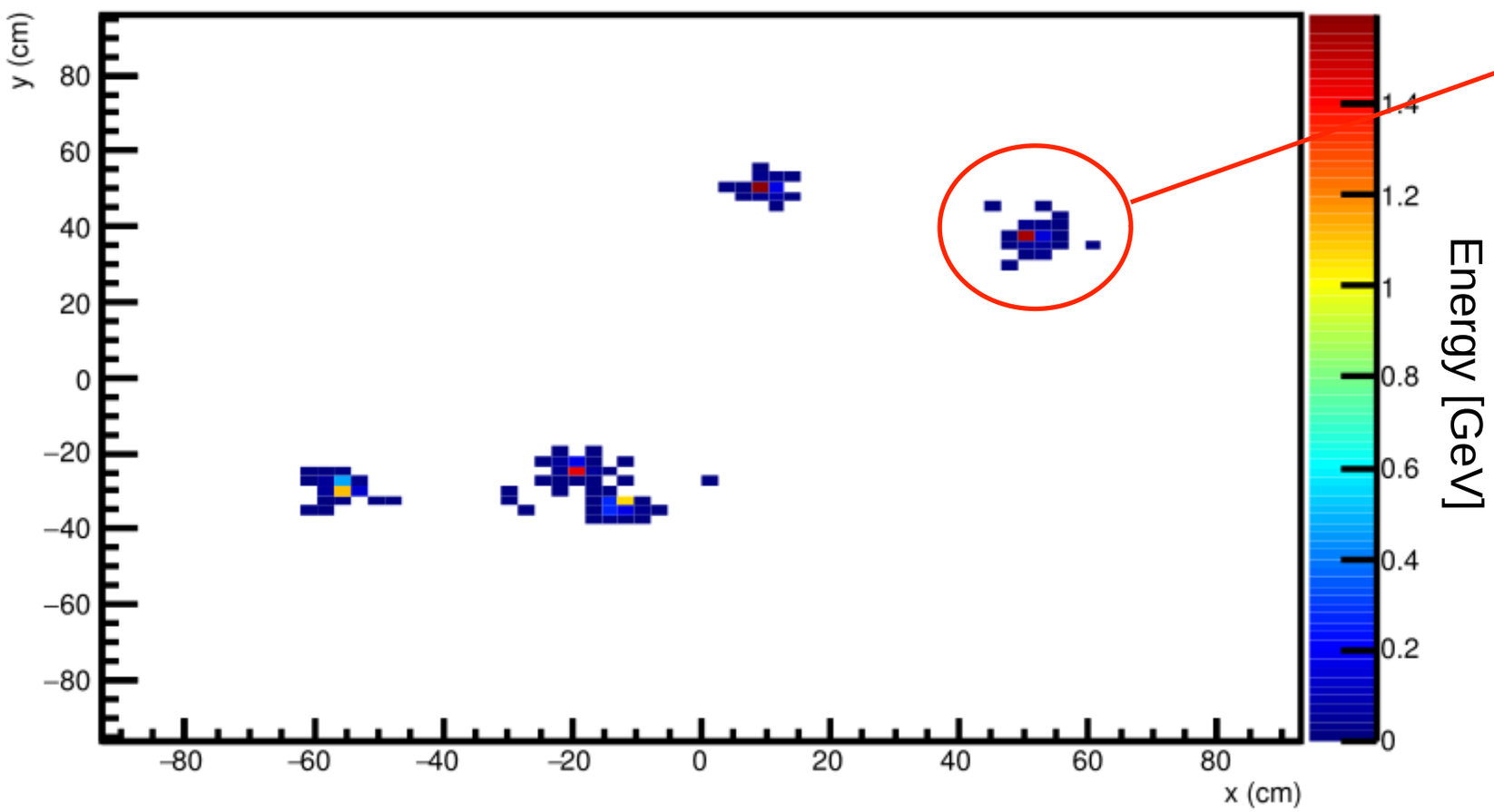
- **Monte Carlo simulated data of photons and neutrons hitting the PANDA detector**





# Assignment IV: Machine Learning

- **Instances are clusters of firing crystals with pre-processed cluster-property parameters as features!**




**Features:**

- Theta, Phi, E-dep
- NrHits, NrBumps
- E1, E1E9, E9E25
- Z20, Z53, LatMom

**Classes:**

- Photon
- Neutron

# Assignment IV: Machine Learning

- ***Challenge*: use ML to suppress neutron background while optimising the photon detection efficiency**
- **Performance metric ("figure-of-merit"):**

**Number of left-over photon clusters after classification**

$$FOM = \frac{S}{\sqrt{S+B}}$$

**Equivalent to the statistical significance accounting for background!!!**

**Number of left-over neutron clusters after classification**

- **Use 50% of data for training and 50% for testing**