CS 360: Machine Learning

Prof. Sara Mathieson

Fall 2020

Admin

- **Roster** is finalized

- **Lab 1** due Tues night

- **TA/office hours**

  Sunday 8:30-10pm (Fiona)

  Monday 9:45-11am (Sara)

  Tuesday 4:30-6pm (Sara)

  Thursday 8-9:30pm (Jason)

Outline for Sept 11

- Informal reading check-in
- Style guidelines for Python
- K-nearest neighbors (KNN) algorithm
- Featurization (intro)

Outline for Sept 11

- Informal reading check-in
- Style guidelines for Python
- K-nearest neighbors (KNN) algorithm
- Featurization (intro)

Reading Check-in 1

1. According to Duame, what is the most central concept in machine learning?

2. Is it okay to look at the test data (the features, labels, or both) during training? Why or why not?

3. Trying to predict whether a student will be most interested in courses related to History, Physics, or Art is an example of what type of learning problem?

4. Trying to predict a person's height based on their genome is an example of what type of learning problem?

Reading Check-in 1

1. According to Duame, what is the most central concept in machine learning?

   Generalization: ability to answer new questions related to the topic studied

2. Is it okay to look at the test data (the features, labels, or both) during training? Why or why not?

3. Trying to predict whether a student will be most interested in courses related to History, Physics, or Art is an example of what type of learning problem?

4. Trying to predict a person's height based on their genome is an example of what type of learning problem?

Reading Check-in 1

1. According to Duame, what is the most central concept in machine learning?

   Generalization: ability to answer new questions related to the topic studied

2. Is it okay to look at the test data (the features, labels, or both) during training? Why or why not?

   No! If we look at the test data (either the *features* or the *labels*), then any measurement of the performance of our algorithm becomes inaccurate

3. Trying to predict whether a student will be most interested in courses related to History, Physics, or Art is an example of what type of learning problem?

4. Trying to predict a person's height based on their genome is an example of what type of learning problem?

Reading Check-in 1

1. According to Duame, what is the most central concept in machine learning?

   Generalization: ability to answer new questions related to the topic studied

2. Is it okay to look at the test data (the features, labels, or both) during training? Why or why not?

   No! If we look at the test data (either the *features* or the *labels*), then any measurement of the performance of our algorithm becomes inaccurate

3. Trying to predict whether a student will be most interested in courses related to History, Physics, or Art is an example of what type of learning problem?

   Multiclass classification

4. Trying to predict a person's height based on their genome is an example of what type of learning problem?

Reading Check-in 1

1. According to Duame, what is the most central concept in machine learning?

   Generalization: ability to answer new questions related to the topic studied

2. Is it okay to look at the test data (the features, labels, or both) during training? Why or why not?

   No! If we look at the test data (either the *features* or the *labels*), then any measurement of the performance of our algorithm becomes inaccurate

3. Trying to predict whether a student will be most interested in courses related to History, Physics, or Art is an example of what type of learning problem?

   Multiclass classification

4. Trying to predict a person's height based on their genome is an example of what type of learning problem?

   Regression

Outline for Sept 11

- Informal reading check-in
- Style guidelines for Python
- K-nearest neighbors (KNN) algorithm
- Featurization (intro)

Python style

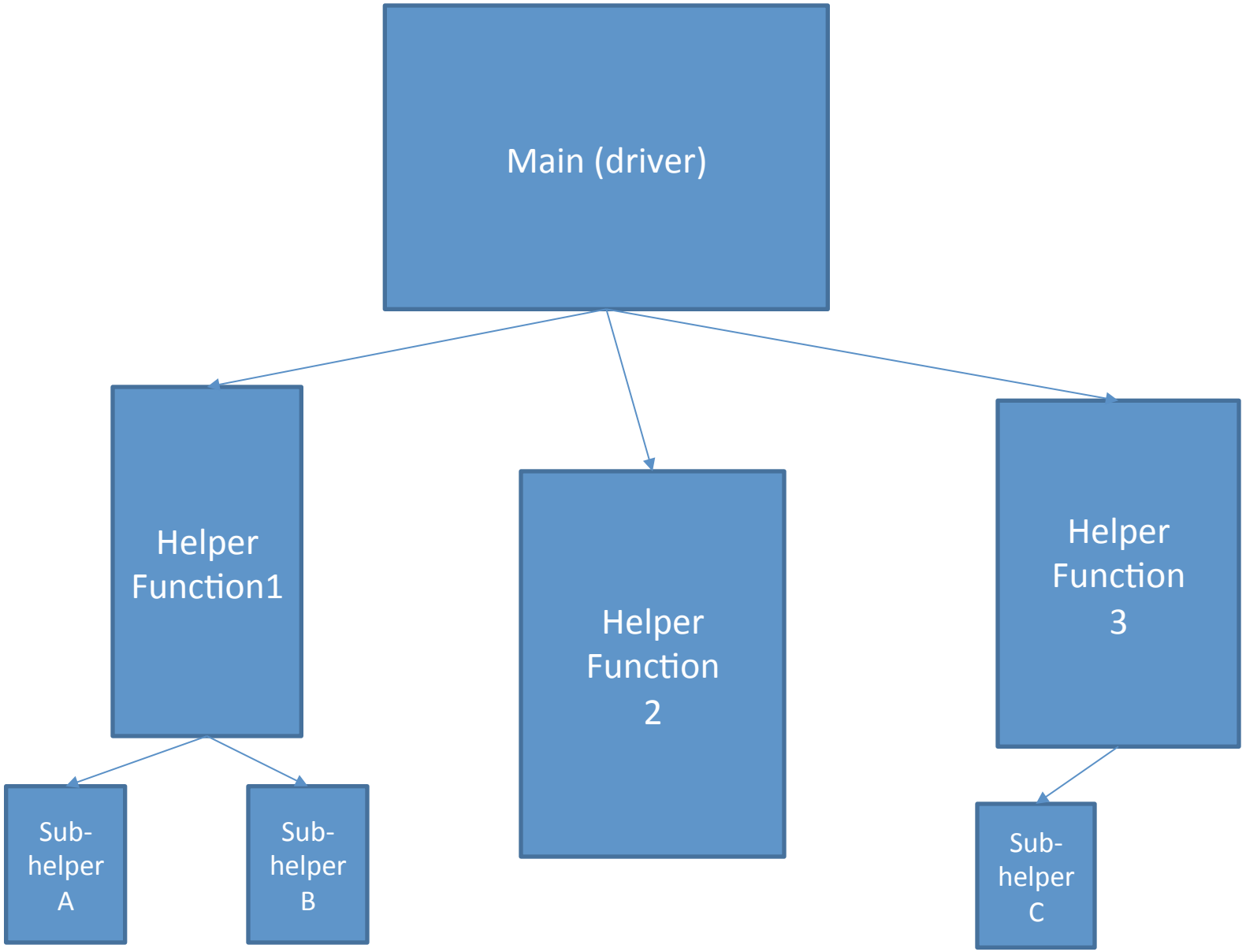
- Decompose code into natural functions
- Avoid global variables (sometimes useful)
- Include a file header with purpose, author, and date
- Include headers for each function
- No lines over 80 chars
- Variable names implicitly show type
- Include line breaks and comments!

Python style examples

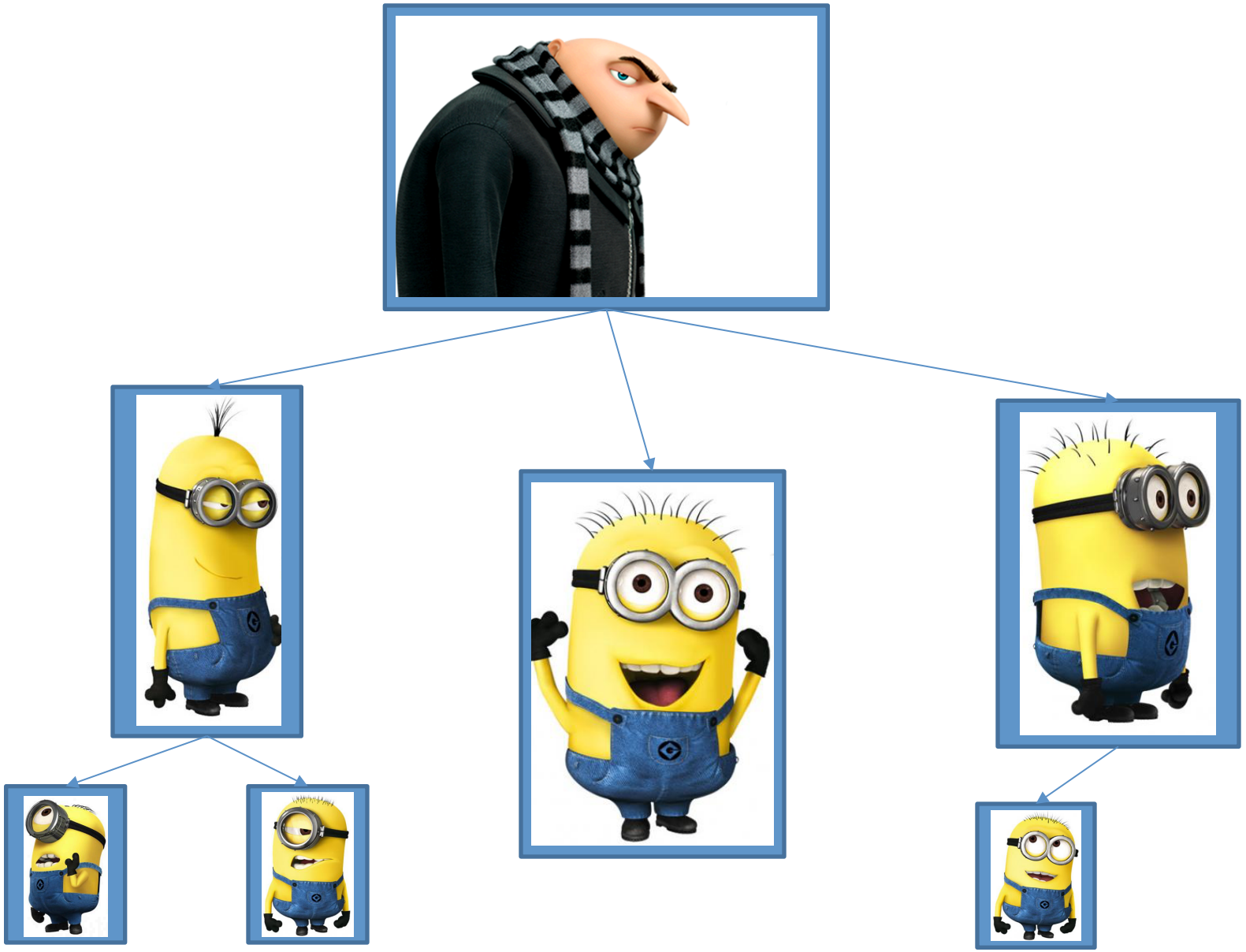
```
"""
Ask the user for their name and welcome them to CS21.
Author: Sara Mathieson
Date: 9/7/18
"""

def main():

    # ask user for their name and print greeting
    name = input("Enter your name: ")
    print("Hello", name, "!")

main()
```

```
def factorial(n):
    """
    Given a non-negative integer n, return n! = n*(n-1)*(n-2)....3*2*1.
    """
    fact = 1 # set up an accumulator variable
    for i in range(n):
        fact = fact * (i+1) # accumulator pattern
    return fact
```

Structure of main and “helper” functions



Structure of main and “helper” functions



Reminder: steps of top-down-design (TDD)

Reminder: steps of top-down-design (TDD)

1) Design a high-level main function that captures the basic idea of the program.

Reminder: steps of top-down-design (TDD)

1) Design a high-level main function that captures the basic idea of the program.

2) As you're writing/designing main, think about which details can be abstracted into small tasks. Make names for these functions and write their signatures below main.

Reminder: steps of top-down-design (TDD)

1) Design a high-level main function that captures the basic idea of the program.

2) As you're writing/designing main, think about which details can be abstracted into small tasks. Make names for these functions and write their signatures below main.

3) "Stub" out the functions. This means that they should work and return the correct type so that your code runs, but they don't do the correct task yet. For example, if a function should return a list, you can return []. Or if it returns a boolean, you can return False.

Reminder: steps of top-down-design (TDD)

1) Design a high-level main function that captures the basic idea of the program.

2) As you're writing/designing main, think about which details can be abstracted into small tasks. Make names for these functions and write their signatures below main.

3) "Stub" out the functions. This means that they should work and return the correct type so that your code runs, but they don't do the correct task yet. For example, if a function should return a list, you can return []. Or if it returns a boolean, you can return False.

4) Iterate on your design until you have a working main and stubbed out functions. Then start implementing the functions, starting from the "bottom up".

Reasons to use TDD

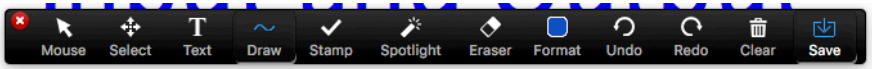
- Creates code that is easier to implement, debug, modify, and extend
- Avoids going off in the wrong direction (i.e. implementing functions that are not useful or don't serve the program)
- Creates code that is easier for you or someone else to read and understand later on

Outline for Sept 11

- Informal reading check-in
- Style guidelines for Python
- K-nearest neighbors (KNN) algorithm
- Featurization (intro)

Input and Output

rom - com



$X = \begin{bmatrix} — \vec{x}_i^T — \end{bmatrix}$ n examples rows, $n \times p$

p features

Y = label or output

$X_{train} \approx 80\%$

$X_{test} \approx 20\%$

$Y = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ \vdots \\ C \end{bmatrix}$ $n \times 1$

- Regression

  $y \in \mathbb{R}$

- Binary classification

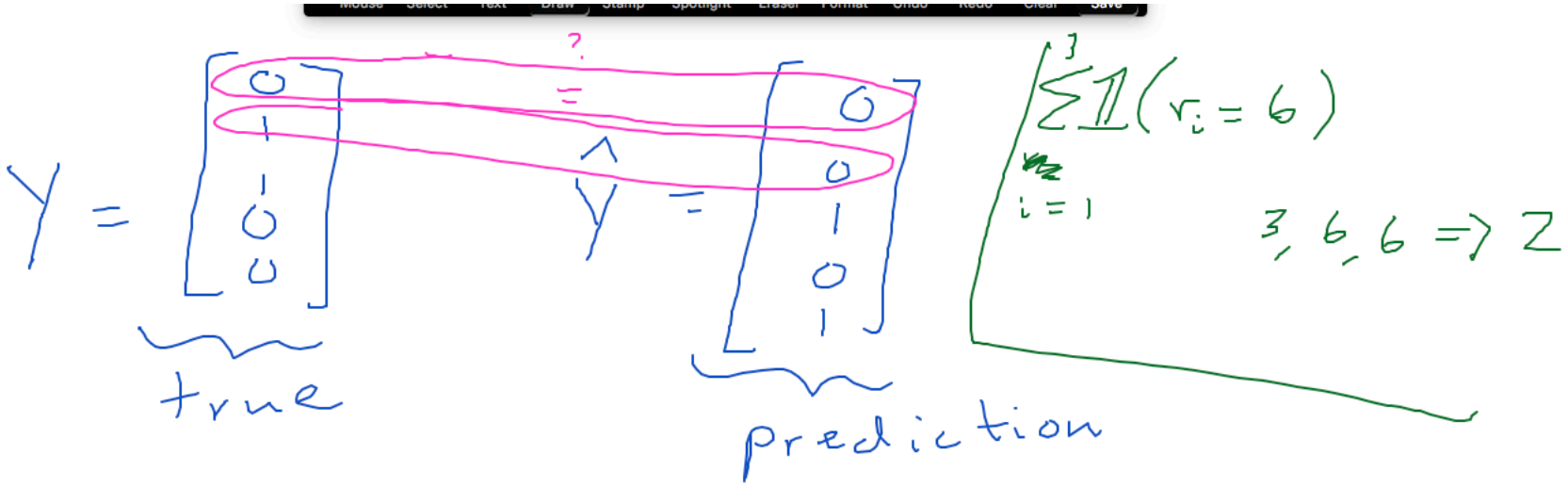
  $y \in \{0, 1\}$, $y \in \{-1, +1\}$

- Multiclass classification

  $y \in \{1, 2, 3, \cdots, C\}$

- Ranking

Accuracy



$$acc = \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}(y_i = \hat{y}_i)$$

?

$\mathbb{1}$(statement) =

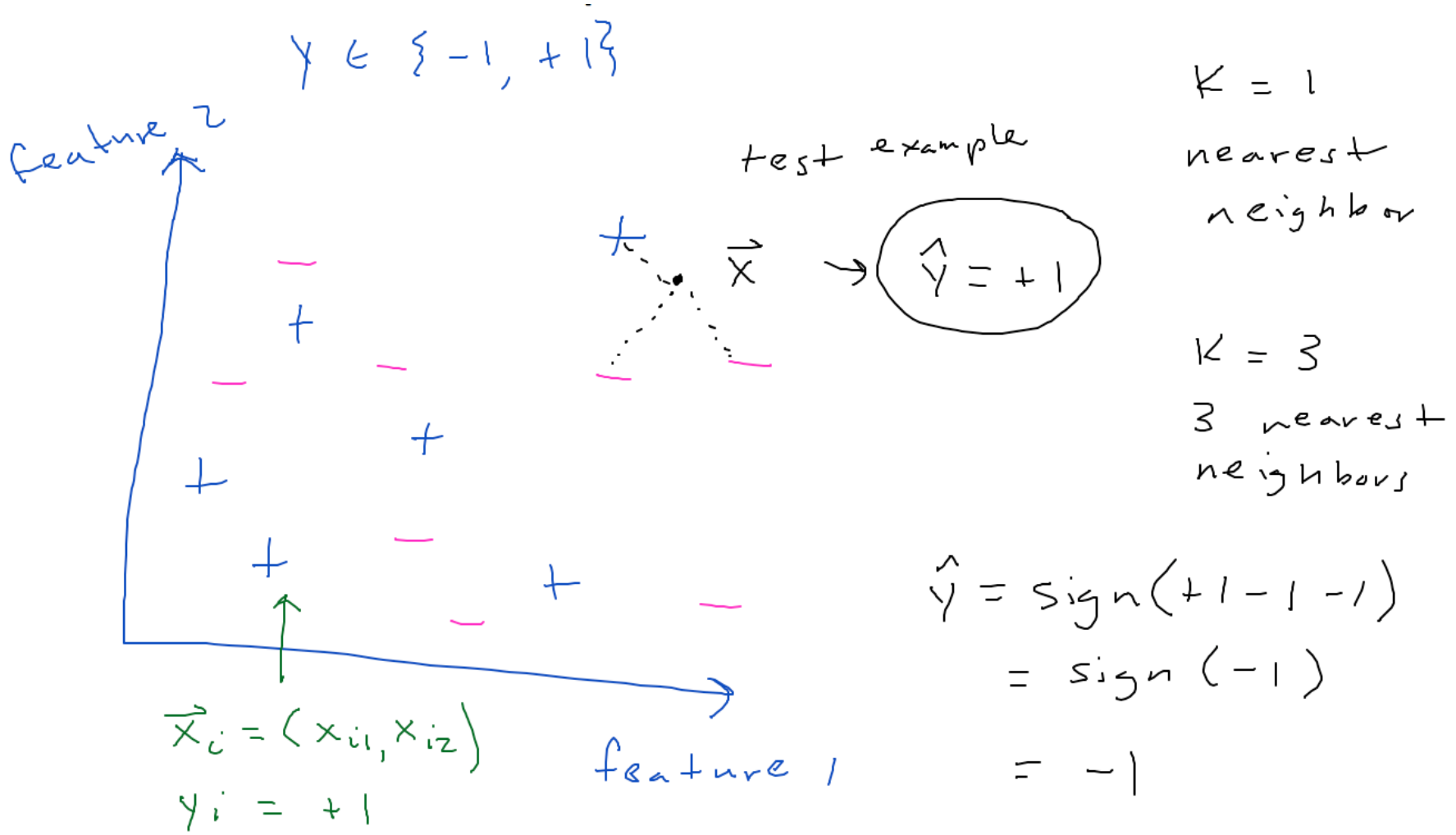
indicator { 1 if true
0 otherwise

ex : $\frac{1}{5}(1 + 0 + 1 + 1 + 0) = \frac{3}{5}$

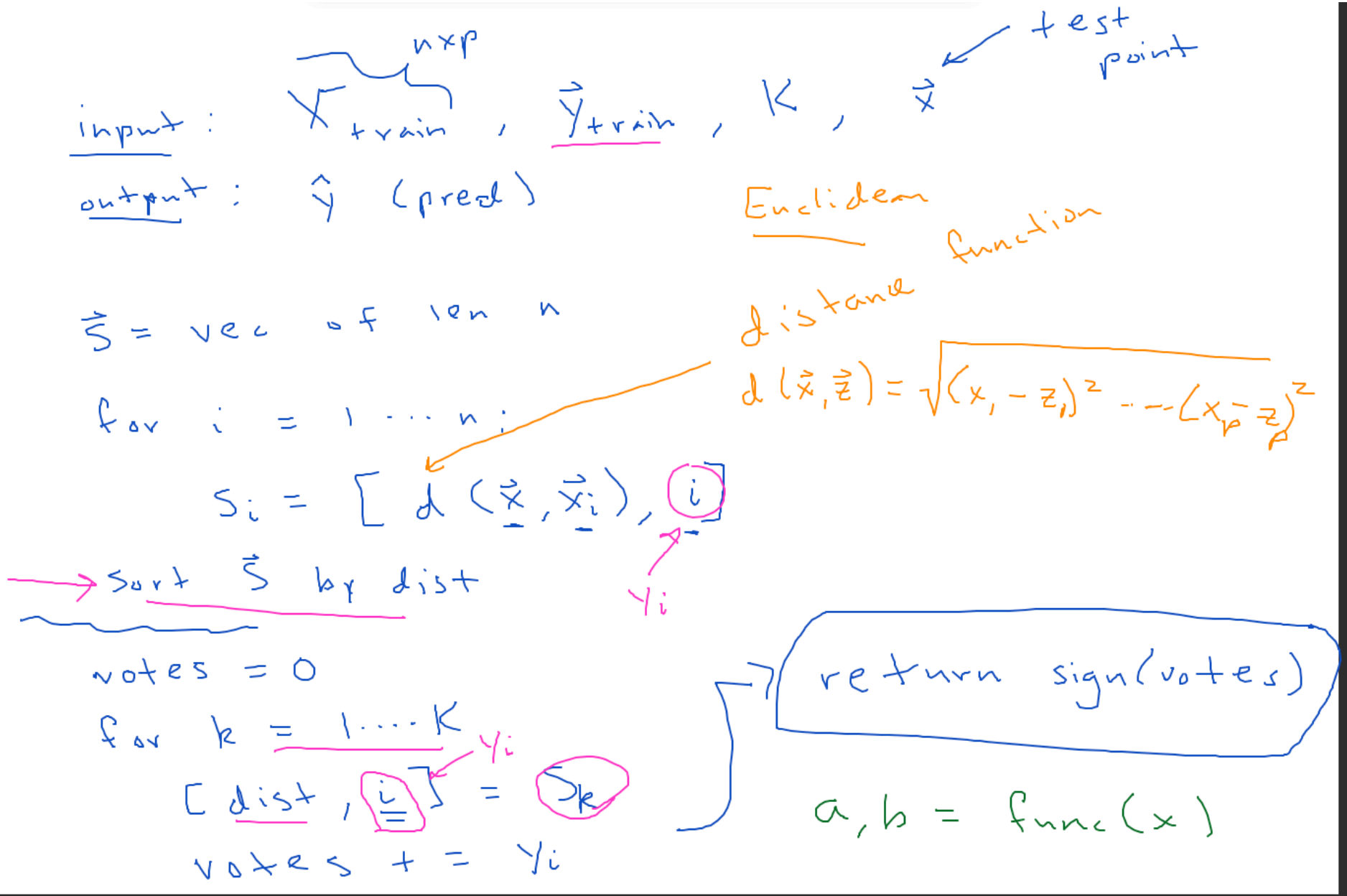
$$error = \frac{1}{n} \sum \mathbb{1}(y_i \neq \hat{y}_i) = 1 - acc$$

K-nearest neighbors

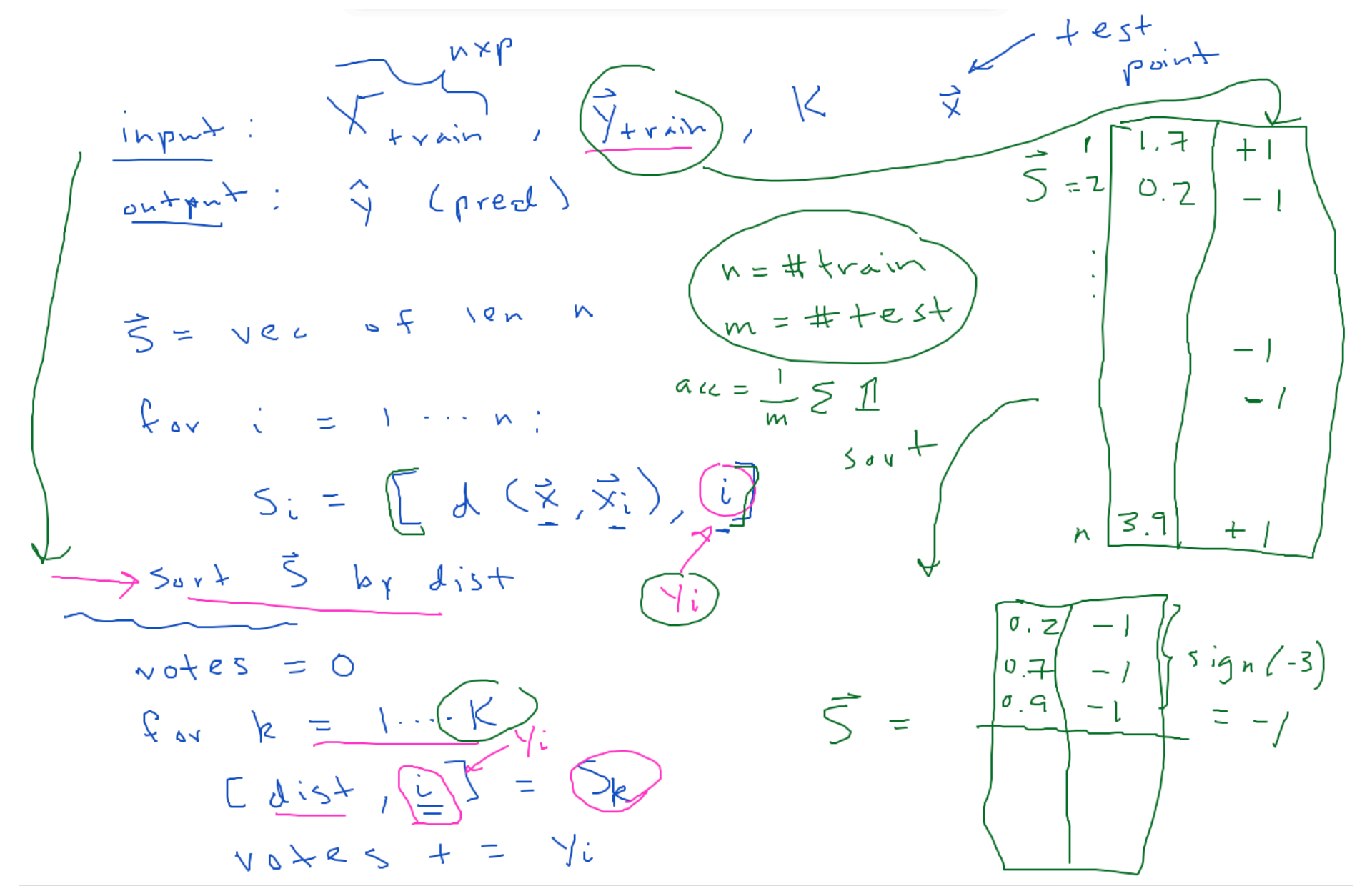
Idea: predict that the new example has the same label as the most similar example we've seen before



KNN Algorithm



KNN Algorithm



Multiclass KNN

Next time!

K-nearest neighbors creates implicit decision boundaries

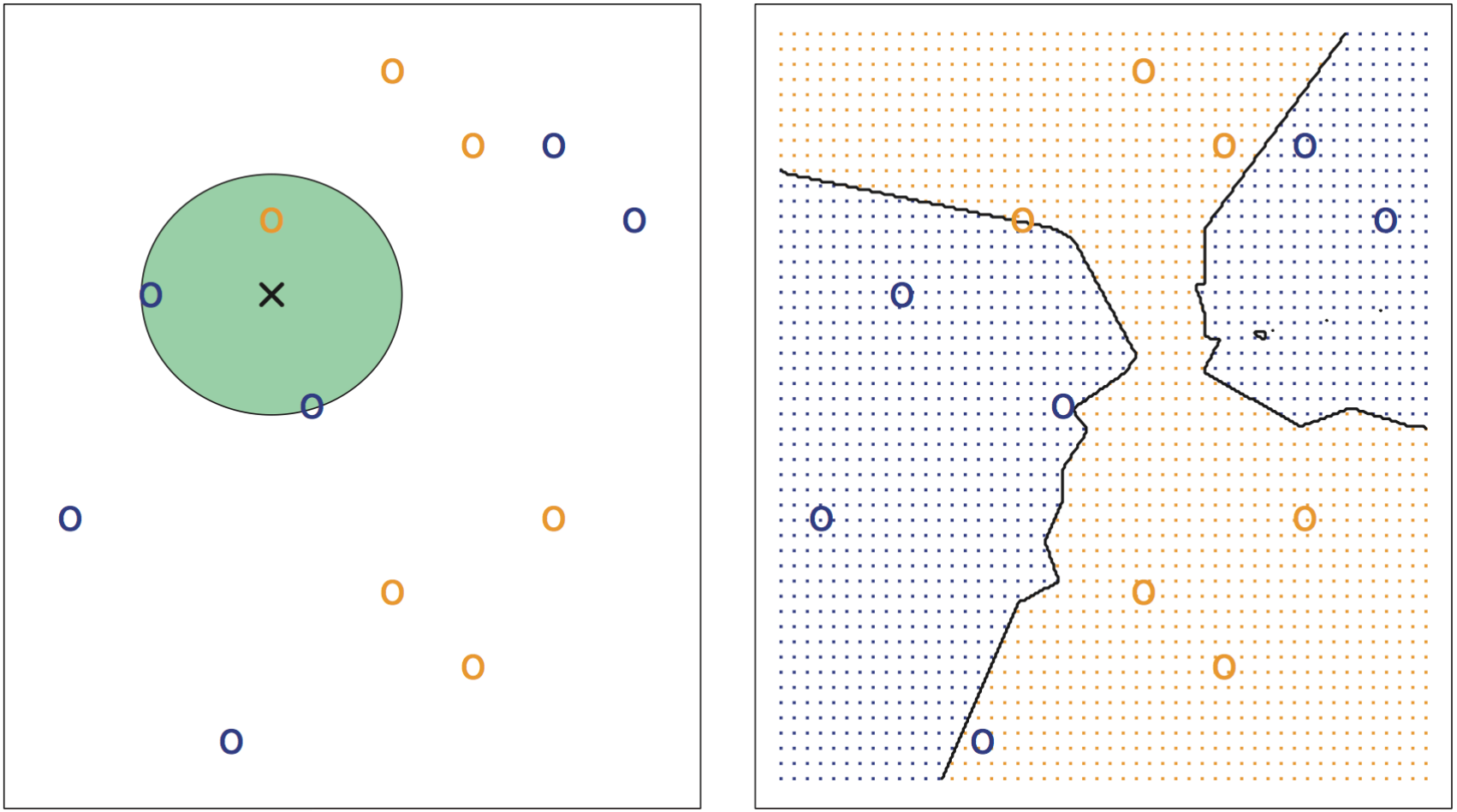


Figure 2.14 from ISL book, KNN with two classes (C=2), and K=3

Outline for Sept 11

- Informal reading check-in
- Style guidelines for Python
- K-nearest neighbors (KNN) algorithm
- Featurization (intro)

Next time!

Lab 1 notes, zip code data