

# Structuring the Data

Lecture 2

CS3400 Machine Learning

# Tabular Data

	A	B	C	D	E	F
1	Country ▼	Salesperson ▼	Order Date ▼	OrderID ▼	Units ▼	Order Amount ▼
2	USA	Fuller	1/01/2011	10392	13	1,440.00
3	UK	Gloucester	2/01/2011	10397	17	716.72
4	UK	Bromley	2/01/2011	10771	18	344.00
5	USA	Finchley	3/01/2011	10393	16	2,556.95
6	USA	Finchley	3/01/2011	10394	10	442.00
7	UK	Gillingham	3/01/2011	10395	9	2,122.92
8	USA	Finchley	6/01/2011	10396	7	1,903.80
9	USA	Callahan	8/01/2011	10399	17	1,765.60
10	USA	Fuller	8/01/2011	10404	7	1,591.25
11	USA	Fuller	9/01/2011	10398	11	2,505.60
12	USA	Coghill	9/01/2011	10403	18	855.01
13	USA	Finchley	10/01/2011	10401	7	3,868.60
14	USA	Callahan	10/01/2011	10402	11	2,713.50
15	UK	Rayleigh	13/01/2011	10406	15	1,830.78
16	USA	Callahan	14/01/2011	10408	10	1,622.40
17	USA	Farnham	14/01/2011	10409	19	319.20
18	USA	Farnham	15/01/2011	10410	16	802.00

- The easiest data to work with is tabular data
- Records are organized into rows
- Variables are organized into columns (fields)
- Each variable has a type (e.g., int, text, etc.)
- Commonly stored in relational databases or spreadsheets

# Semi-Structured

```
{  
  "orders": [  
    {  
      "orderno": "748745375",  
      "date": "June 30, 2088 1:54:23 AM",  
      "trackingno": "TN0039291",  
      "custid": "11045",  
      "customer": [  
        {  
          "custid": "11045",  
          "fname": "Sue",  
          "lname": "Hatfield",  
          "address": "1409 Silver Street",  
          "city": "Ashland",  
          "state": "NE",  
          "zip": "68003"  
        }  
      ]  
    }  
  ]  
}
```

- Tabular data is often stored in JSON or similar files
- These files do not enforce a schema
- We cannot guarantee that each record will have the same fields or that all of the values are the same type
- Semi-structured data often needs careful manipulation and validation before use

# Unstructured

barismo  
364 Broadway, Cambridge, MA

4.2 ★★★★★ 59 reviews

Sort by: Most helpful ▾

★★★★★ Lots of light and character and with some good music. It's a nice place to sit and relax, read, or do a little work. I usually come for coffee and pastries but I seriously love their lunch and brunch offerings. Recently introducing "Waffle Saturdays," I think I'll be back more often on the weekends.

**Mike Oltmans**  
3 years ago  
★★★★★ I can't say enough about this place. The Batista's love making coffee and it shows. The ever changing baked goods are great and I love the Italian style coffee bar where you can stand and drink your coffee and chat with the Batistas and ... [More](#)

**Alessandro Bahgat**  
a year ago  
★★★★★ Their pain au chocolat is one of the best I've had in Cambridge.

**Matt Suppelsa**  
9 months ago  
★★★★★ Good coffee, nice space, but no WiFi

- Some data such as text are completely unstructured
- Cannot be used directly with classic ML models but can be with newer deep learning models
- Fields such as natural language processing and information retrieval are focused on how to describe and use these data

# Non-numerical variables

- Not all variables are numerical:
  - Categorical: color, vehicle type
  - Boolean: married, automatic engine
  - Ordinal: star ratings
- Possible to engineer features (numerical variables) from non-numerical variables
- We will discuss this in CS3300 Data Science
- In this class, we'll mostly work with tabular data containing only numerical variables

# Data is Transformed for Machine Learning

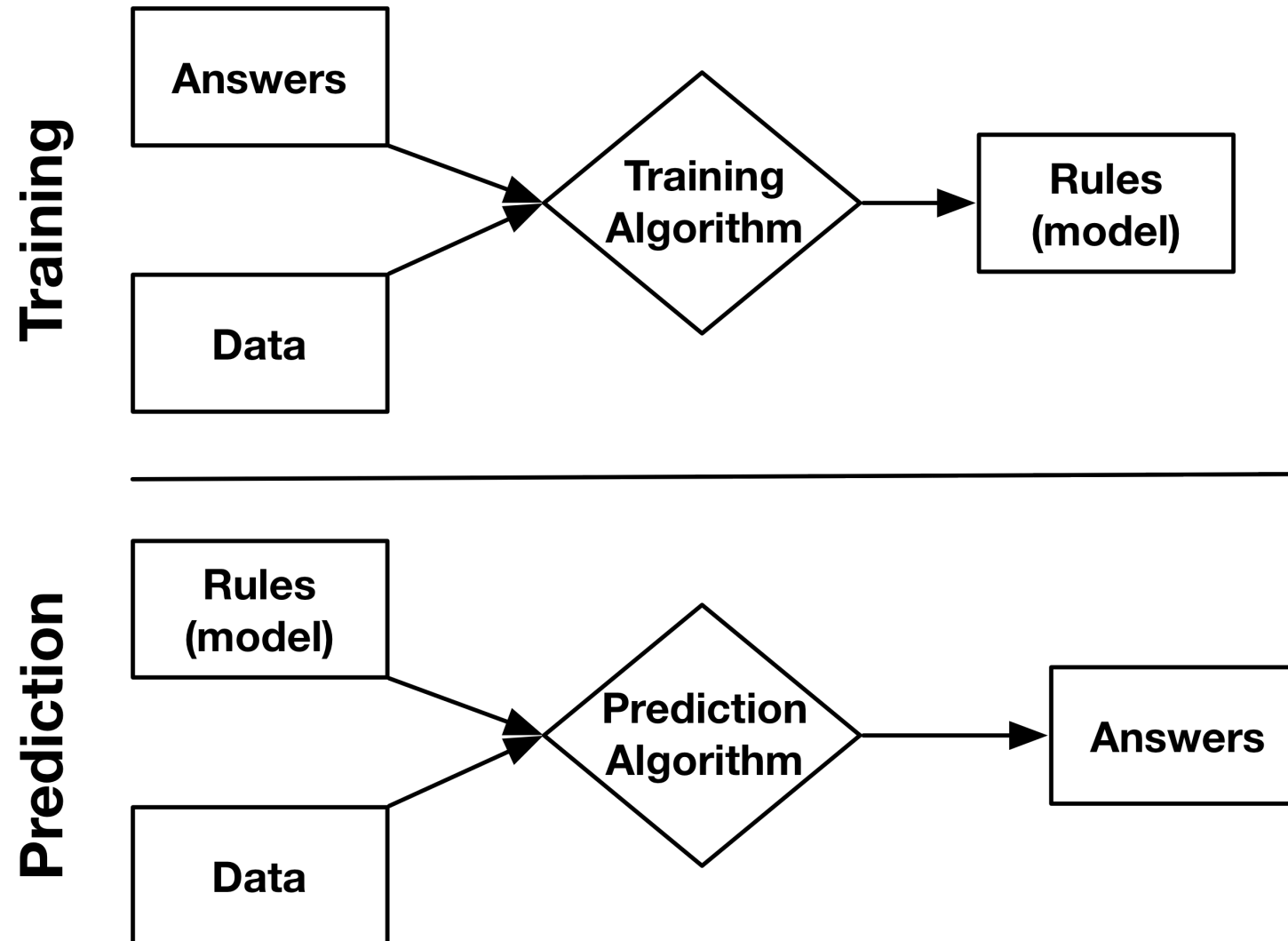
**Feature Matrices**

1	2	8	3	0	5	5	2	2	0	9	3	5	6
1	6	2	3	0	5	7	4	2	0	9	3	5	6
1	1	8	5	0	5	4	5	2	0	9	3	5	6
1	6	2	3	0	2	3	6	2	0	9	3	5	6
1	3	8	5	0	2	7	7	2	0	9	3	5	6
1	4	8	5	0	5	9	8	2	0	9	3	5	6
1	9	2	3	0	2	0	9	2	0	9	3	5	6
1	0	8	5	0	5	4	3	2	0	9	3	5	6
1	2	2	5	0	6	7	1	2	0	9	3	5	6

**Label Vectors**

1
0
1
1
0
1
0
0
1

# Machine Learning



# Two Data Sets

## Training

- Examples with known answers
- We have a feature matrix and a label vector
- We want to train a model that can predict the values in the label vector for each corresponding row in the feature matrix

## Naïve Data

- We have a feature matrix but we do not have a label vector
- We want to apply the model to predict the values that would be in the label vector



1	2	8	3	0	5	5	2	2	0	9	3	5	6
1	6	2	3	0	5	7	4	2	0	9	3	5	6
1	1	8	5	0	5	4	5	2	0	9	3	5	6
1	6	2	3	0	2	3	6	2	0	9	3	5	6
1	3	8	5	0	2	7	7	2	0	9	3	5	6
1	4	8	5	0	5	9	8	2	0	9	3	5	6
1	9	2	3	0	2	0	9	2	0	9	3	5	6
1	0	8	5	0	5	4	3	2	0	9	3	5	6
1	2	2	5	0	6	7	1	2	0	9	3	5	6

1
0
1
1
1
0
1
0
0
1

Training  
Algorithm

Rules  
(model)

Feature matrix: numerical variables  
describing the objects being classified

1	2	8	3	0	5	5	2	2	0	9	3	5	6
1	6	2	3	0	5	7	4	2	0	9	3	5	6
1	1	8	5	0	5	4	5	2	0	9	3	5	6
1	6	2	3	0	2	3	6	2	0	9	3	5	6
1	3	8	5	0	2	7	7	2	0	9	3	5	6
1	4	8	5	0	5	9	8	2	0	9	3	5	6
1	9	2	3	0	2	0	9	2	0	9	3	5	6
1	0	8	5	0	5	4	3	2	0	9	3	5	6
1	2	2	5	0	6	7	1	2	0	9	3	5	6

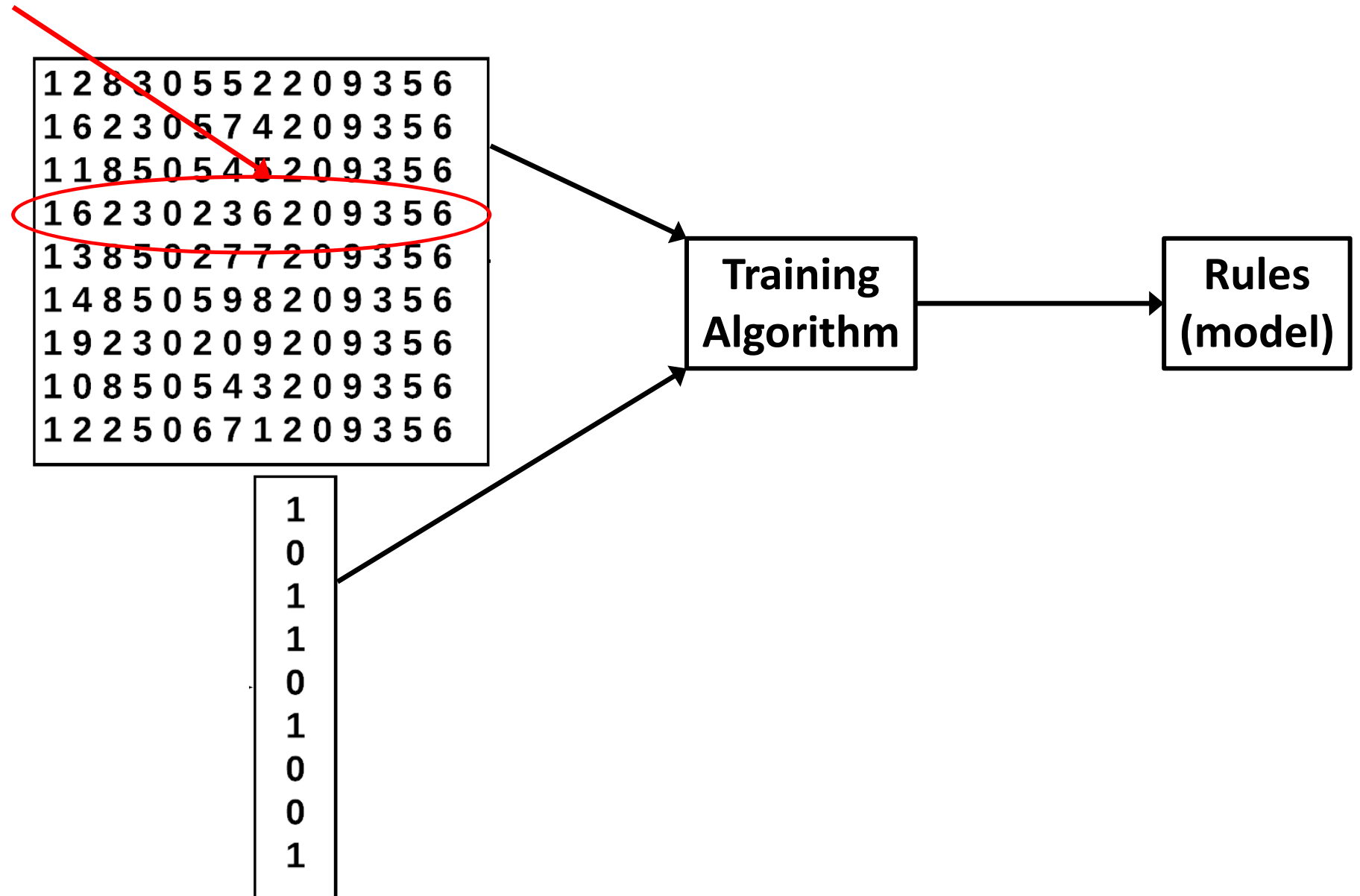
Training  
Algorithm

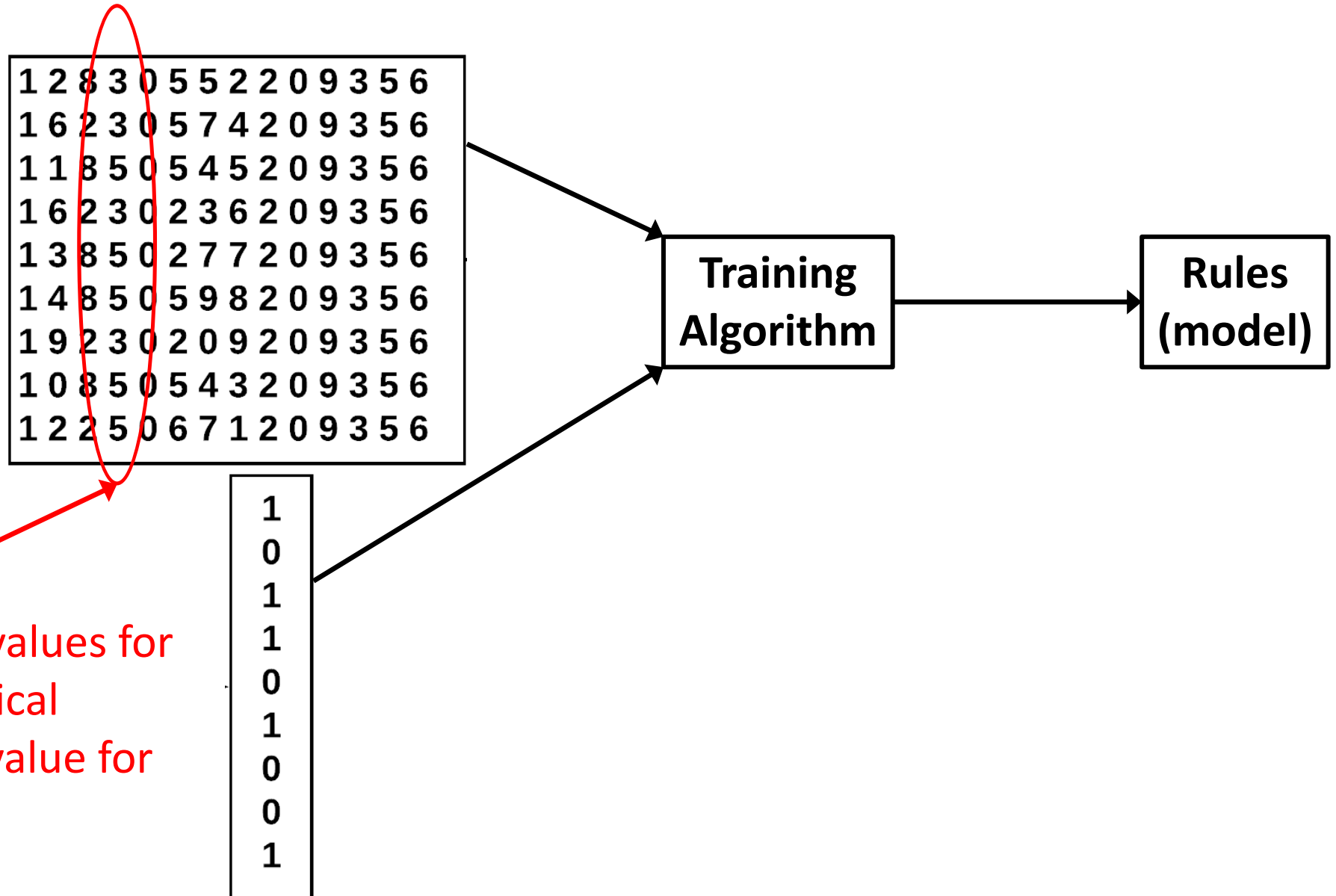
Rules  
(model)

Label vector: the desired  
output values

1  
0  
1  
1  
1  
0  
1  
0  
0  
0  
1

Each row in the feature matrix stores the features for a single sample (object)





Each column store the values for a single feature (numerical variable). There is one value for every sample.

1	2	8	3	0	5	5	2	2	0	9	3	5	6
1	6	2	3	0	5	7	4	2	0	9	3	5	6
1	1	8	5	0	5	4	5	2	0	9	3	5	6
1	6	2	3	0	2	3	6	2	0	9	3	5	6
1	3	8	5	0	2	7	7	2	0	9	3	5	6
1	4	8	5	0	5	9	8	2	0	9	3	5	6
1	9	2	3	0	2	0	9	2	0	9	3	5	6
1	0	8	5	0	5	4	3	2	0	9	3	5	6
1	2	2	5	0	6	7	1	2	0	9	3	5	6

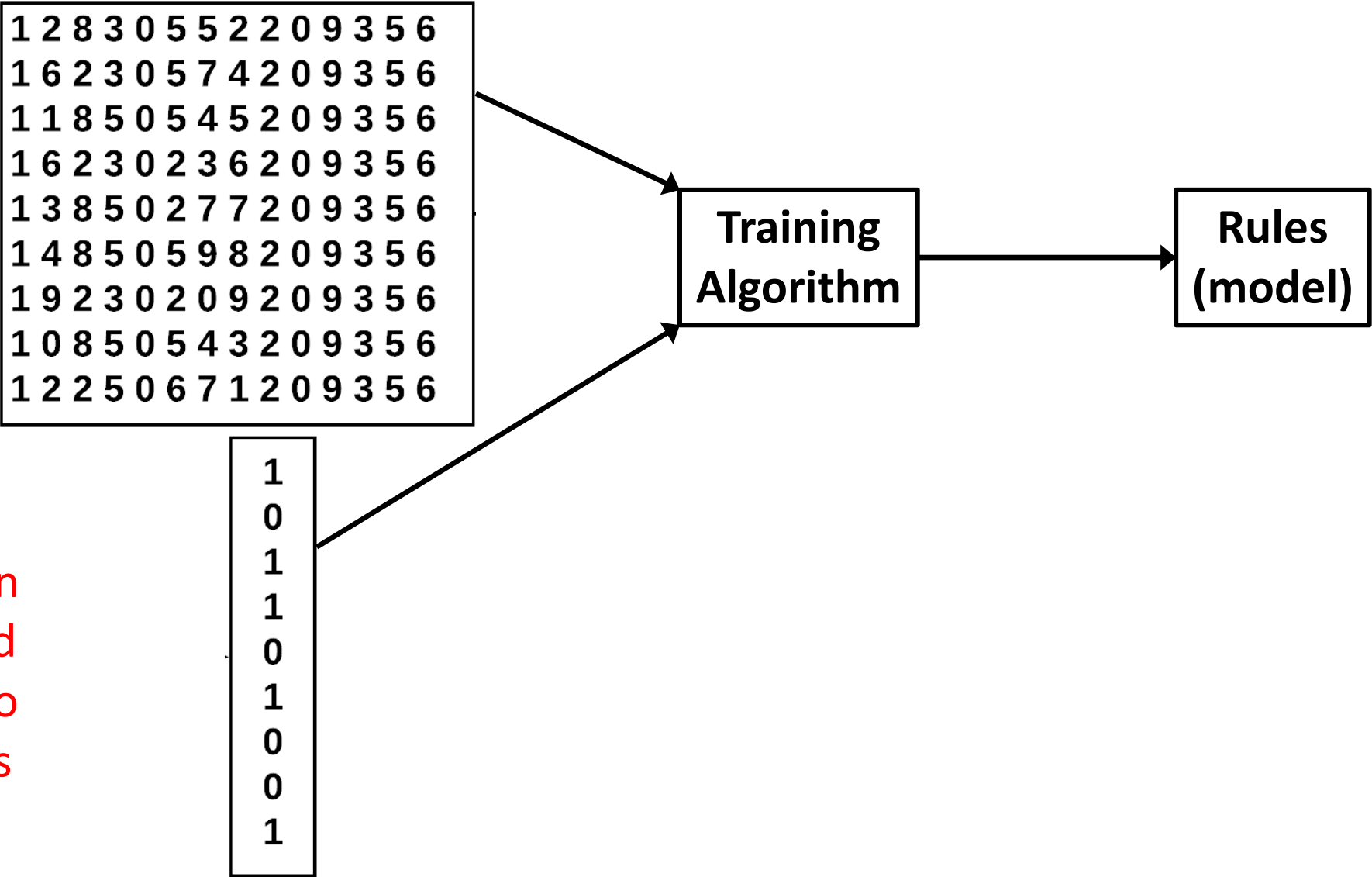
1
0
1
1
0
1
0
0
1

Training  
Algorithm

Rules  
(model)

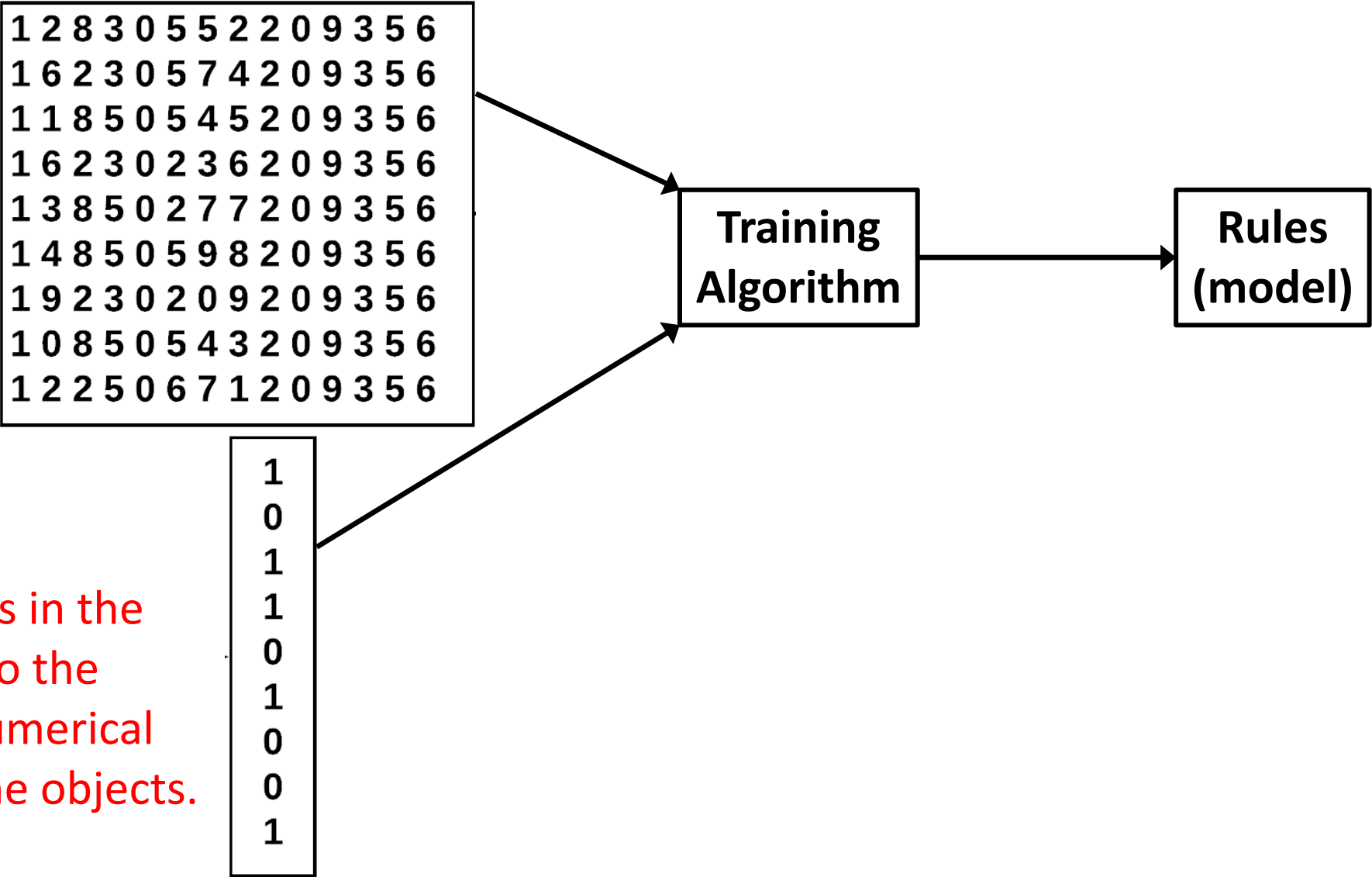
The label vector stores the desired output value for each sample.  
There is only one 1 output variable.  
Each sample has one entry.

The dimensions of the feature matrix and label vectors are directly associated with the size of the data set.



The number of rows in the feature matrix and label vector is equal to the number of objects in the data set.

The dimensions of the feature matrix and label vectors are directly associated with the size of the data set.



The number of columns in the feature matrix equals to the number of features (numerical variables) describing the objects.

1	2	8	3	0	5	5	2	2	0	9	3	5	6
1	6	2	3	0	5	7	4	2	0	9	3	5	6
1	1	8	5	0	5	4	5	2	0	9	3	5	6
1	6	2	3	0	2	3	6	2	0	9	3	5	6
1	3	8	5	0	2	7	7	2	0	9	3	5	6
1	4	8	5	0	5	9	8	2	0	9	3	5	6
1	9	2	3	0	2	0	9	2	0	9	3	5	6
1	0	8	5	0	5	4	3	2	0	9	3	5	6
1	2	2	5	0	6	7	1	2	0	9	3	5	6

1
0
1
1
0
1
0
0
1

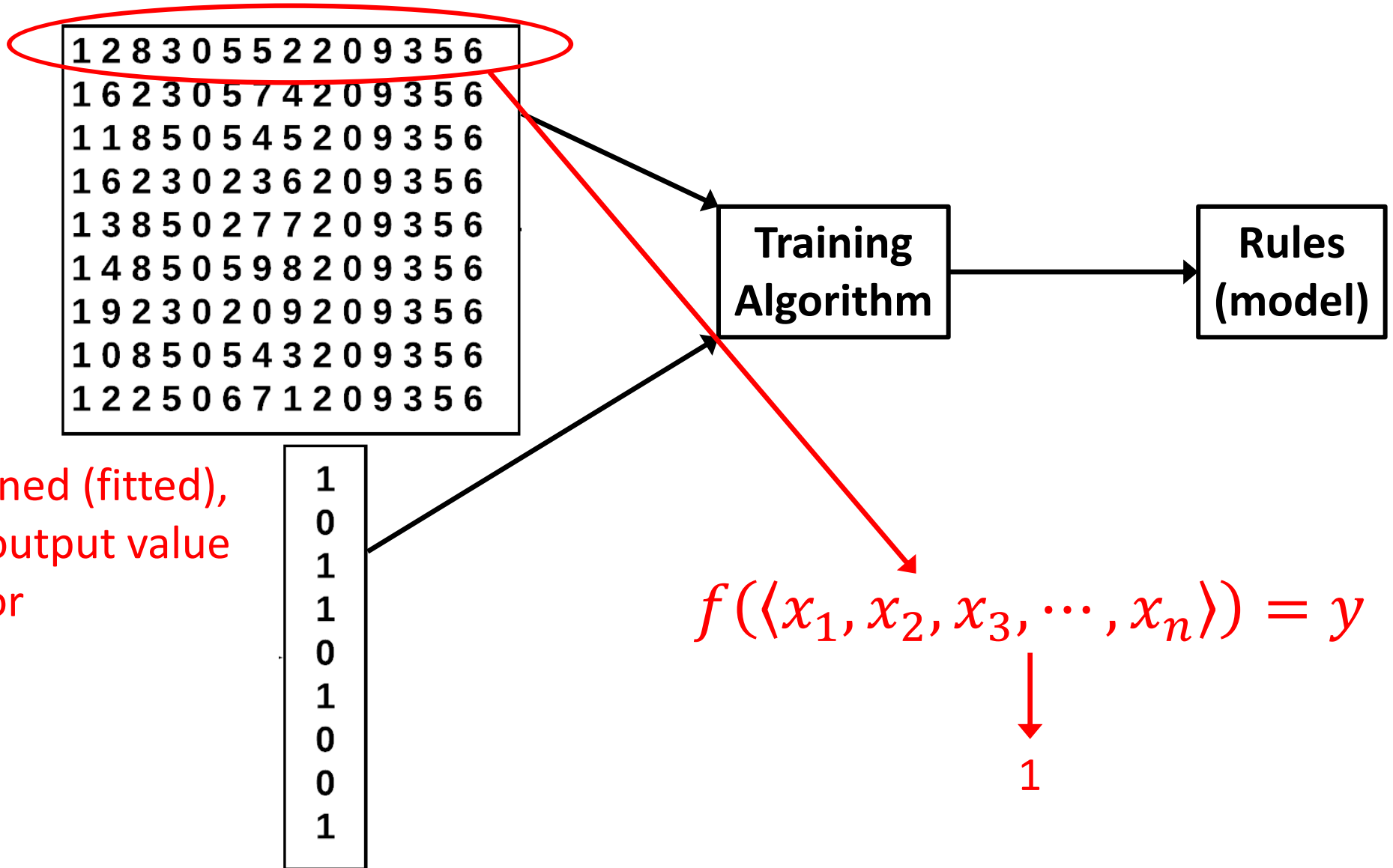
Training  
Algorithm

Rules  
(model)

A machine learning model is a mathematical function that can estimate the output value for a single object from the object's feature vector.

$$f(\langle x_1, x_2, x_3, \dots, x_n \rangle) = y$$





Once a model is trained (fitted),  
it can estimate the output value  
from a feature vector

# Iris Data Set

- Trying to classify irises as one of three species
- Response: species (3)
- Features (predictors) are:
  - Petal width
  - Petal height
  - Sepal width
  - Sepal height
- 150 objects
- 4 features



# Iris Features (150 x 4)



Sepal length

Petal length

Sepal width

Petal width

[ 5.1 3.5 1.4 0.2]  
[ 4.9 3. 1.4 0.2]  
[ 4.7 3.2 1.3 0.2]  
[ 4.6 3.1 1.5 0.2]  
[ 5. 3.6 1.4 0.2]  
[ 5.4 3.9 1.7 0.4]  
[ 4.6 3.4 1.4 0.3]  
[ 5. 3.4 1.5 0.2]  
[ 4.4 2.9 1.4 0.2]  
[ 4.9 3.1 1.5 0.1]  
[ 5.4 3.7 1.5 0.2]  
[ 4.8 3.4 1.6 0.2]  
[ 4.8 3. 1.4 0.1]  
[ 4.3 3. 1.1 0.1]  
[ 5.8 4. 1.2 0.2]  
[ 5.7 4.4 1.5 0.4]  
[ 5.4 3.9 1.3 0.4]  
[ 5.1 3.5 1.4 0.3]  
[ 5.7 3.8 1.7 0.3]  
[ - - - - ]

# Iris Features (150 x 4)



Sepal length

Petal length

Sepal width

Petal width

[ [5.1 3.5 1.4 0.2]  
[4.9 3. 1.4 0.2]  
[4.7 3.2 1.3 0.2]  
[4.6 3.1 1.5 0.2]  
[5. 3.6 1.4 0.2]  
[5.4 3.9 1.7 0.4]  
[4.6 3.4 1.4 0.3]  
[5. 3.4 1.5 0.2]  
[4.4 2.9 1.4 0.2]  
[4.9 3.1 1.5 0.1]  
[5.4 3.7 1.5 0.2]  
[4.8 3.4 1.6 0.2]  
[4.8 3. 1.4 0.1]  
[4.3 3. 1.1 0.1]  
[5.8 4. 1.2 0.2]  
[5.7 4.4 1.5 0.4]  
[5.4 3.9 1.3 0.4]  
[5.1 3.5 1.4 0.3]  
[5.7 3.8 1.7 0.3]  
- - - -



# Iris Features (150 x 4)



Sepal length

Petal length

Sepal width

Petal width

```
[ [5.1 3.5 1.4 0.2]
  [4.9 3. 1.4 0.2]
  [4.7 3.2 1.3 0.2]
  [4.6 3.1 1.5 0.2]
  [5. 3.6 1.4 0.2]
  [5.4 3.9 1.7 0.4]
  [4.6 3.4 1.4 0.3]
  [5. 3.4 1.5 0.2]
  [4.4 2.9 1.4 0.2]
  [4.9 3.1 1.5 0.1]
  [5.4 3.7 1.5 0.2]
  [4.8 3.4 1.6 0.2]
  [4.8 3. 1.4 0.1]
  [4.3 3. 1.1 0.1]
  [5.8 4. 1.2 0.2]
  [5.7 4.4 1.5 0.4]
  [5.4 3.9 1.3 0.4]
  [5.1 3.5 1.4 0.3]
  [5.7 3.8 1.7 0.3]
```

# Iris Features (150 x 4)



Sepal length

Petal length

Sepal width

Petal width

```
[ [5.1 3.5 1.4 0.2]
  [4.9 3.  1.4 0.2]
  [4.7 3.2 1.3 0.2]
  [4.6 3.1 1.5 0.2]
  [5.  3.6 1.4 0.2]
  [5.4 3.9 1.7 0.4]
  [4.6 3.4 1.4 0.3]
  [5.  3.4 1.5 0.2]
  [4.4 2.9 1.4 0.2]
  [4.9 3.1 1.5 0.1]
  [5.4 3.7 1.5 0.2]
  [4.8 3.4 1.6 0.2]
  [4.8 3.  1.4 0.1]
  [4.3 3.  1.1 0.1]
  [5.8 4.  1.2 0.2]
  [5.7 4.4 1.5 0.4]
  [5.4 3.9 1.3 0.4]
  [5.1 3.5 1.4 0.3]
  [5.7 3.8 1.7 0.3]
```

# Iris Labels (150)



Setosa

Versicolor

Virginica

[ 0 0 0 0 0 0 0 0 0 0 0 0  
0 0 0 0 0 0 0 0 0 0 0 0  
1 1 1 1 1 1 1 1 1 1 1 1  
2 2 2 2 2 2 2 2 2 2 2 2  
2 2 ]



# Iris Labels (150)



Setosa

Versicolor

Virginica

```
[ 0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0
  1  1  1  1  1  1  1  1  1  1  1  1  1
  2  2  2  2  2  2  2  2  2  2  2  2  2
  2  2]
```

A red arrow points from the 'Versicolor' label to the second row of the matrix. A red oval highlights the third, fourth, and fifth rows of the matrix.



# Iris Labels (150)



Setosa

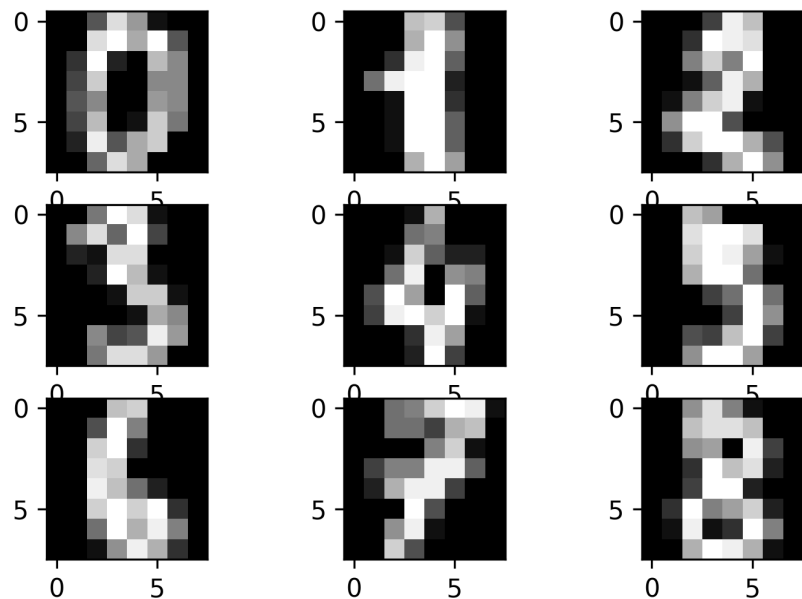
Versicolor

Virginica

```
[ 0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0
  1  1  1  1  1  1  1  1  1  1  1  1  1
  2  2  2  2  2  2  2  2  2  2  2  2  2
  2  2]
```

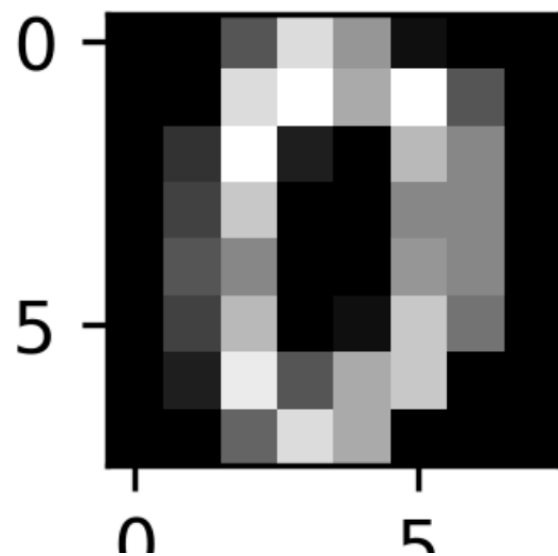
A red arrow points from the 'Virginica' label to the 7th column of the matrix. A red oval highlights the last two rows of the matrix.

# Digits Data Set

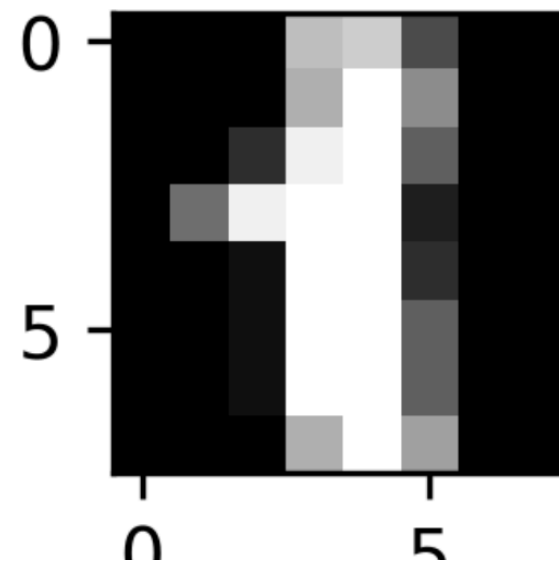


- 1,797 pictures of hand-written digits
- Each image is 8 x 8 pixels
- The pixel colors are greyscale and stored as intensities using the integers 0 - 16
- Each image can be classified into 1 of 10 mutually-exclusive classes (digits 0 - 9)
- The goal is to identify the written digit in the image

# Representing Digits



```
[ 0.  0.  5. 13.  9.  1.  0.  0.]
[ 0.  0. 13. 15. 10. 15.  5.  0.]
[ 0.  3. 15.  2.  0. 11.  8.  0.]
[ 0.  4. 12.  0.  0.  8.  8.  0.]
[ 0.  5.  8.  0.  0.  9.  8.  0.]
[ 0.  4. 11.  0.  1. 12.  7.  0.]
[ 0.  2. 14.  5. 10. 12.  0.  0.]
[ 0.  0.  6. 13. 10.  0.  0.  0.]
```



```
[ 0.  0.  0. 12. 13.  5.  0.  0.]
[ 0.  0.  0. 11. 16.  9.  0.  0.]
[ 0.  0.  3. 15. 16.  6.  0.  0.]
[ 0.  7. 15. 16. 16.  2.  0.  0.]
[ 0.  0.  1. 16. 16.  3.  0.  0.]
[ 0.  0.  1. 16. 16.  6.  0.  0.]
[ 0.  0.  1. 16. 16.  6.  0.  0.]
[ 0.  0.  0. 11. 16. 10.  0.  0.]
```

# Reshaping into a Feature Matrix

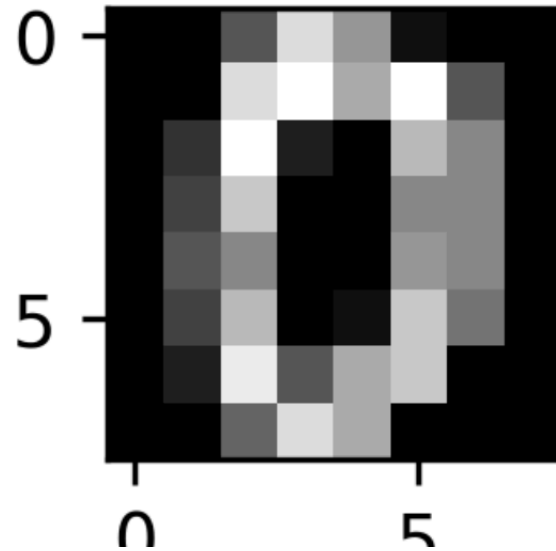
- We describe objects using 1D feature vectors
- We describe a collection of objects using a 2D feature matrix
  - 1 object per row
  - 1 feature per column
- Right now, we have 1,797 8x8 2D matrices
- How do we convert each 2D matrix into a 1D vector with 64 entries?

# Reshaping into a Feature Matrix

- We describe objects using 1D feature vectors
- We describe a collection of objects using a 2D feature matrix
  - 1 object per row
  - 1 feature per column
- Right now, we have 1,797 8x8 2D matrices
- How do we convert each 2D matrix into a 1D vector with 64 entries?

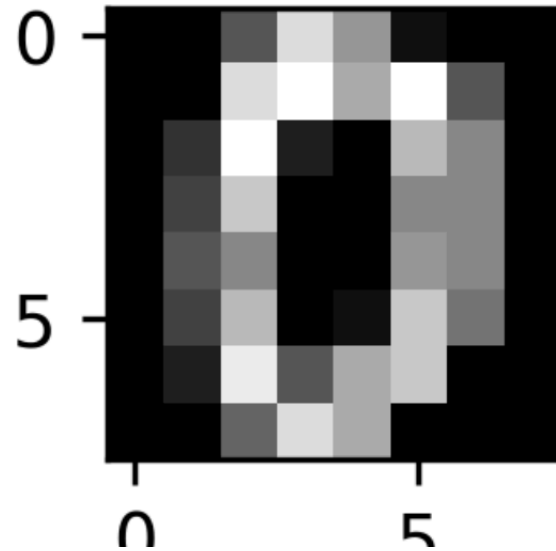
"Reshape" the image matrices into feature vectors and then form a feature matrix from the vectors

# Representing Digits (Reshaping)



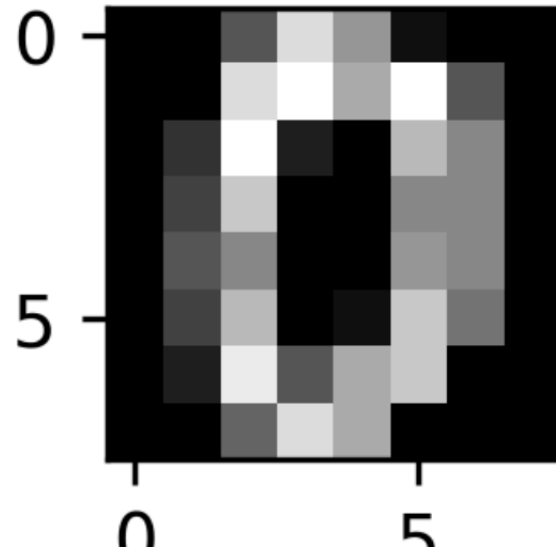
```
[ 0.  0.  5. 13.  9.  1.  0.  0.]  
[ 0.  0. 13. 15. 10. 15.  5.  0.]  
[ 0.  3. 15.  2.  0. 11.  8.  0.]  
[ 0.  4. 12.  0.  0.  8.  8.  0.]  
[ 0.  5.  8.  0.  0.  9.  8.  0.]  
[ 0.  4. 11.  0.  1. 12.  7.  0.]  
[ 0.  2. 14.  5. 10. 12.  0.  0.]  
[ 0.  0.  6. 13. 10.  0.  0.  0.]
```

# Representing Digits (Reshaping)



```
[ 0.  0.  5. 13.  9.  1.  0.  0.]  
[ 0.  0. 13. 15. 10. 15.  5.  0.]  
[ 0.  3. 15.  2.  0. 11.  8.  0.]  
[ 0.  4. 12.  0.  0.  8.  8.  0.]  
[ 0.  5.  8.  0.  0.  9.  8.  0.]  
[ 0.  4. 11.  0.  1. 12.  7.  0.]  
[ 0.  2. 14.  5. 10. 12.  0.  0.]  
[ 0.  0.  6. 13. 10.  0.  0.  0.]
```

# Representing Digits (Reshaping)

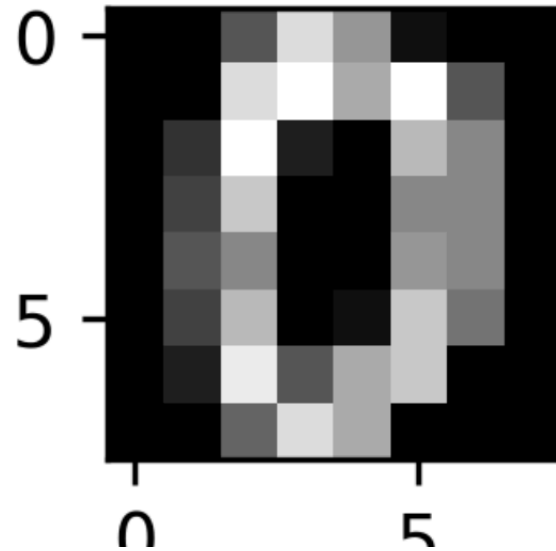


```
[ 0.  0.  5. 13.  9.  1.  0.  0.]  
[ 0.  0. 13. 15. 10. 15.  5.  0.]  
[ 0.  3. 15.  2.  0. 11.  8.  0.]  
[ 0.  4. 12.  0.  0.  8.  8.  0.]  
[ 0.  5.  8.  0.  0.  9.  8.  0.]  
[ 0.  4. 11.  0.  1. 12.  7.  0.]  
[ 0.  2. 14.  5. 10. 12.  0.  0.]  
[ 0.  0.  6. 13. 10.  0.  0.  0.]
```

```
[ 0.  0.  5. 13.  9.  1.  0.  0.]
```



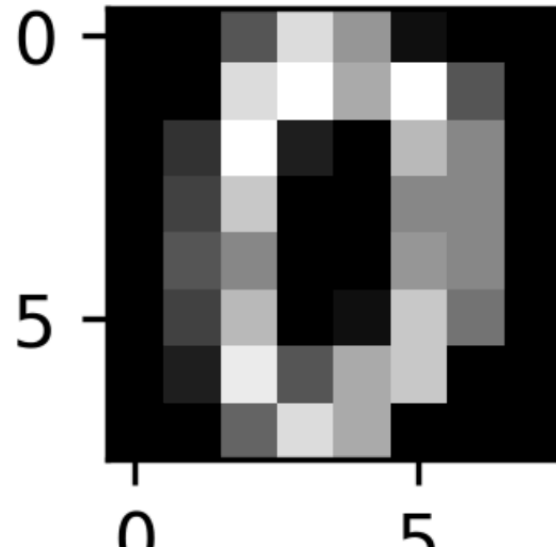
# Representing Digits (Reshaping)



```
[ 0.  0.  5. 13.  9.  1.  0.  0.]  
[ 0.  0. 13. 15. 10. 15.  5.  0.]  
[ 0.  3. 15.  2.  0. 11.  8.  0.]  
[ 0.  4. 12.  0.  0.  8.  8.  0.]  
[ 0.  5.  8.  0.  0.  9.  8.  0.]  
[ 0.  4. 11.  0.  1. 12.  7.  0.]  
[ 0.  2. 14.  5. 10. 12.  0.  0.]  
[ 0.  0.  6. 13. 10.  0.  0.  0.]
```

```
[ 0.  0.  5. 13.  9.  1.  0.  0.]
```

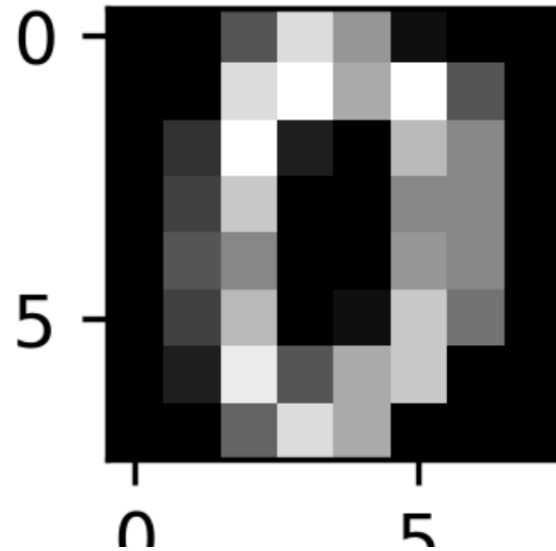
# Representing Digits (Reshaping)



```
[ 0.  0.  5. 13.  9.  1.  0.  0.]  
[ 0.  0. 13. 15. 10. 15.  5.  0.]  
[ 0.  3. 15.  2.  0. 11.  8.  0.]  
[ 0.  4. 12.  0.  0.  8.  8.  0.]  
[ 0.  5.  8.  0.  0.  9.  8.  0.]  
[ 0.  4. 11.  0.  1. 12.  7.  0.]  
[ 0.  2. 14.  5. 10. 12.  0.  0.]  
[ 0.  0.  6. 13. 10.  0.  0.  0.]
```

```
[ 0.  0.  5. 13.  9.  1.  0.  0.  0.  0. 13. 15. 10. 15.  5.  0.]
```

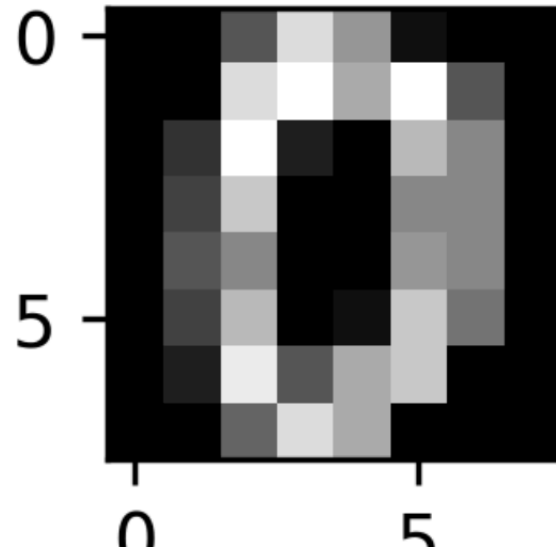
# Representing Digits (Reshaping)



```
[ 0.  0.  5. 13.  9.  1.  0.  0.]  
[ 0.  0. 13. 15. 10. 15.  5.  0.]  
[ 0.  3. 15.  2.  0. 11.  8.  0.]  
[ 0.  4. 12.  0.  0.  8.  8.  0.]  
[ 0.  5.  8.  0.  0.  9.  8.  0.]  
[ 0.  4. 11.  0.  1. 12.  7.  0.]  
[ 0.  2. 14.  5. 10. 12.  0.  0.]  
[ 0.  0.  6. 13. 10.  0.  0.  0.]
```

```
[ 0.  0.  5. 13.  9.  1.  0.  0.  0.  0. 13. 15. 10. 15.  5.  0.]
```

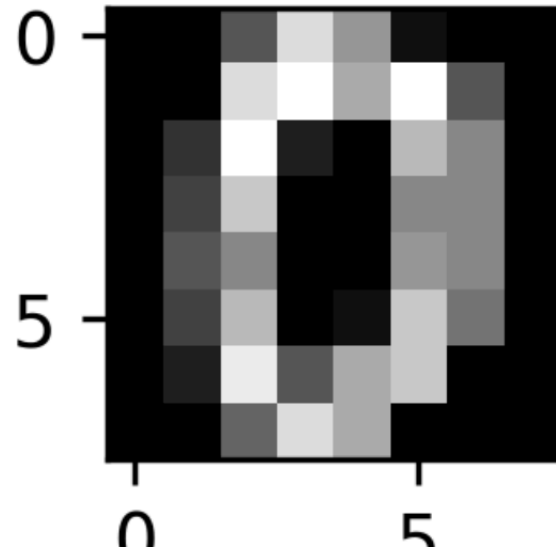
# Representing Digits (Reshaping)



```
[ 0.  0.  5. 13.  9.  1.  0.  0.]  
[ 0.  0. 13. 15. 10. 15.  5.  0.]  
[ 0.  3. 15.  2.  0. 11.  8.  0.]  
[ 0.  4. 12.  0.  0.  8.  8.  0.]  
[ 0.  5.  8.  0.  0.  9.  8.  0.]  
[ 0.  4. 11.  0.  1. 12.  7.  0.]  
[ 0.  2. 14.  5. 10. 12.  0.  0.]  
[ 0.  0.  6. 13. 10.  0.  0.  0.]
```

```
[ 0.  0.  5. 13.  9.  1.  0.  0.  0.  0. 13. 15. 10. 15.  5.  0.  0.  3. 15.  2.  0. 11.  8.  0.]
```

# Representing Digits (Reshaping)



```
[ 0.  0.  5. 13.  9.  1.  0.  0.]  
[ 0.  0. 13. 15. 10. 15.  5.  0.]  
[ 0.  3. 15.  2.  0. 11.  8.  0.]  
[ 0.  4. 12.  0.  0.  8.  8.  0.]  
[ 0.  5.  8.  0.  0.  9.  8.  0.]  
[ 0.  4. 11.  0.  1. 12.  7.  0.]  
[ 0.  2. 14.  5. 10. 12.  0.  0.]  
[ 0.  0.  6. 13. 10.  0.  0.  0.]
```

```
[ 0.  0.  5. 13.  9.  1.  0.  0.  0.  0. 13. 15. 10. 15.  5.  0.  0.  3. 15.  2.  0. 11.  8.  0.  
[ 0.  4. 12.  0.  0.  8.  8.  0.  0.  5.  8.  0.  0.  9.  8.  0.] 0.  4. 11.  0.  1. 12.  7.  0.  
[ 0.  2. 14.  5. 10. 12.  0.  0.  0.  0.  6. 13. 10.  0.  0.  0.]
```

# Digits Feature Matrix

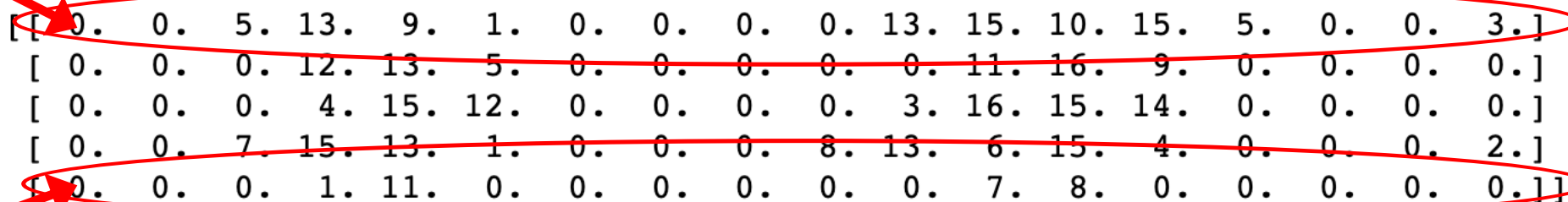
Excerpt of feature matrix. 5 rows, 18 columns.

```
[[ 0.  0.  5. 13.  9.  1.  0.  0.  0.  0. 13. 15. 10. 15.  5.  0.  0.  3.]  
 [ 0.  0.  0. 12. 13.  5.  0.  0.  0.  0.  0. 11. 16.  9.  0.  0.  0.  0.]  
 [ 0.  0.  0.  4. 15. 12.  0.  0.  0.  0.  3. 16. 15. 14.  0.  0.  0.  0.]  
 [ 0.  0.  7. 15. 13.  1.  0.  0.  0.  8. 13.  6. 15.  4.  0.  0.  0.  2.]  
 [ 0.  0.  0.  1. 11.  0.  0.  0.  0.  0.  0.  7.  8.  0.  0.  0.  0.  0.]
```

# Digits Feature Matrix

Excerpt of feature matrix. 5 rows, 18 columns.

First Image



[	0.	0.	5.	13.	9.	1.	0.	0.	0.	0.	13.	15.	10.	15.	5.	0.	0.	3.]
[	0.	0.	0.	12.	13.	5.	0.	0.	0.	0.	0.	11.	16.	9.	0.	0.	0.	0.]
[	0.	0.	0.	4.	15.	12.	0.	0.	0.	0.	3.	16.	15.	14.	0.	0.	0.	0.]
[	0.	0.	7.	15.	13.	1.	0.	0.	0.	8.	13.	6.	15.	4.	0.	0.	0.	2.]
[	0.	0.	0.	1.	11.	0.	0.	0.	0.	0.	0.	7.	8.	0.	0.	0.	0.	0.]

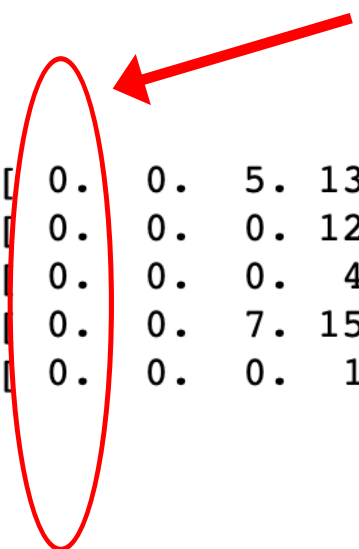
Fifth Image

Each row contains the pixel values for a single image

# Digits Feature Matrix

Excerpt of feature matrix. 5 rows, 18 columns.

The first column contains the upper-left pixel values for each image



[	0.	0.	5.	13.	9.	1.	0.	0.	0.	0.	13.	15.	10.	15.	5.	0.	0.	3.]
[	0.	0.	0.	12.	13.	5.	0.	0.	0.	0.	0.	11.	16.	9.	0.	0.	0.	0.]
[	0.	0.	0.	4.	15.	12.	0.	0.	0.	0.	3.	16.	15.	14.	0.	0.	0.	0.]
[	0.	0.	7.	15.	13.	1.	0.	0.	0.	8.	13.	6.	15.	4.	0.	0.	0.	2.]
[	0.	0.	0.	1.	11.	0.	0.	0.	0.	0.	0.	7.	8.	0.	0.	0.	0.	0.]



# Digits Label Vector (1,797)

10 digits (0 – 9). Each digit represented as an integer in the label vector.  
There are 1,797 entries in the label vector – one for each image.

```
[0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 9 5 5 6 5 0
 9 8 9 8 4 1 7 7 3 5 1 0 0 2 2 7 8 2 0 1 2 6 3 3 7 3 3 4 6 6 6 4 9 1 5 0 9
 5 2 8 2 0 0 1 7 6 3 2 1 7 4 6 3 1 3 9 1 7 6 8 4 3 1 4 0 5 3 6 9 6 1 7 5 4
 4 7 2 8 2 2 5 7 9 5 4 8 8 4 9 0 8 9 8 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7
 8 9]
```

# Numpy and Scipy

- Numpy and Scipy are libraries for numerical and scientific computing
- Provide memory-efficient data structures for numerical data
  - Arrays – Numpy
  - Sparse Matrices – Scipy
- Expressive API for indexing and operations
- Efficient algorithms for manipulating array data
  - Call C and Fortran libraries such as Lapack where possible

# Numpy Array Data Structure

- Multidimensional (1, 2, 3, etc. dims)
- All values have the same type (e.g., float or int)
- Allocates a large block of contiguous memory
  - Very little memory overhead
  - Single object for array itself
  - Faster to iterate through memory in-order due to cache effects
- More options than what Python provides
  - e.g., unsigned ints, 2, 4, and 8 bytes
  - 4 or 8 byte floats
- Arrays do not grow dynamically – fixed size!

# Creating Numpy Arrays

- From Python lists

```
array = np.array([1, 2, 3, 4, 5], dtype=np.float32) # 1D array of length 5
```

- Using special functions

- fill with all zeros

```
array = np.zeros(4, dtype=np.int32) # 1D array of length 4
```

- fill with all ones

```
array = np.ones((4, 5)) # 4 x 5 2D array
```

# Array Attributes

- Dimensions

`array.ndim`            # int

- Shape

`array.shape` # tuple of ints

- Data Type

`array.dtype` # `np.int32`, `np.float32`, etc.

# Indexing

- Single element (returns a scalar)

```
array[0]           # 1D array  
array[1, 2]        # 2D array  
array[1, 2, 3, 4, 5] # 5D array
```

- Selecting a single dimension

```
array[:, 5, :] - returns a 2D array
```

# Indexing

- Reversing a 1D array

```
reversed_array = array[::-1]
```

# Fancy Indexing

Unlike Python lists, arrays can be indexed by:

- Array or list of indices

```
# returns array with elements at indices in  
list  
subset = array[[1, 8, 4, 5, 2]]  
subset = array[another_array]
```

This can be used to select certain elements or re-arrange the order of elements in an array



# Fancy Indexing

- Array or list of boolean values

```
# Entries with a corresponding True are returned.  
subset = array[[True, False, False, True, True]]
```

This is often used with a "boolean mask" array

# Sorting / Maximums / Minimums

- Sorting

```
sorted = np.sort(array)           # 1D array
```

- Maximum:

```
max = np.amax(array)             # 1D array
```

- Minimum

```
min = np.amin(array)            # 1D array
```

# Sorting / Maximums / Minimums

There are also equivalent functions that return the indices

```
sorted_idx = np.argsort(array)    # 1D array
```

```
array[sorted_idx[0]]              # smallest element of array
```

```
array[sorted_idx[-1]]             # largest element of array
```

```
# return a sorted version of array
```

```
sorted_array = array[sorted_idx]
```

# Fancy Indexing

## Selecting entries that match a given value

```
# all values equal to 1
mask = array == 1          # returns a boolean array
selected = array[mask]
```

```
# all values greater than 5
mask = array > 5           # returns a boolean array
selected = array[mask]
```

# Reshaping

We can use the Numpy reshape operation to reshape the images from the digits example into a feature matrix.

The digit images are stored in a 3D ndarray:

```
images.shape -> (1797, 8, 8) # 1,798 8x8 images
```

```
features = images.reshape(1797, 64)
```

```
features.shape -> (1797, 64) # 1,798 feature  
vectors of 64 entries
```

# Concatenating Arrays

Sometimes, it's easier to create variables through separate processes and then join them into a single matrix at the end. The horizontal stack function makes this easy:

```
X1 = np.zeros((4, 5))  
X2 = np.zeros((4, 7))  
X = np.hstack([X1, X2])  
X.shape == (4, 12)
```

Note that the input matrices must have the same number of rows!