



# K-Nearest Neighbor

Applied Machine Learning  
Derek Hoiem

# What breed is this?



(a) Siberian Husky



(b) Australian Cattle Dog



(c) Alaskan Malamut



(d) Bernese Mountain Dog

# How did you do it?



(a) Siberian Husky



(b) Australian Cattle Dog



(c) Alaskan Malamut



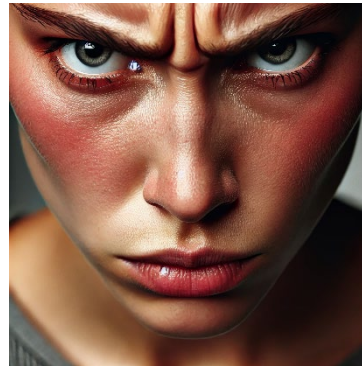
(d) Bernese Mountain Dog

# Foundation of Learning: Association

Similar inputs tend to produce similar outputs

Have a nice ... ?

Happy or  
angry?



Affordable or  
expensive?



# Key principle of machine learning

Given feature/target pairs  $(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)$ :

if  $\mathbf{x}_i$  is similar to  $\mathbf{x}_j$ , then  $y_i$  is probably similar to  $y_j$

But how do we determine whether two things are similar?

- Need to convert an image, audio, signal, text, etc. into a number or vector
- Need to select a distance measure and/or decide the importance of different aspects of similarity

In ML, every object/concept/document is a number or series of numbers.

The main challenge is creating those numbers in a way that similarity makes sense.

# Data and Information

- Data are stored numbers
- Information is the predictive power of data
- Processing data never adds information, but it can make the information easier to access

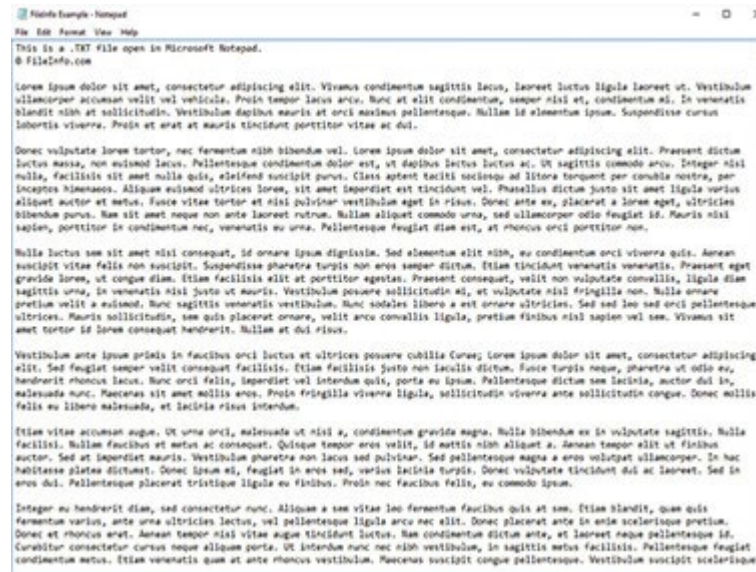


# How do we represent data?

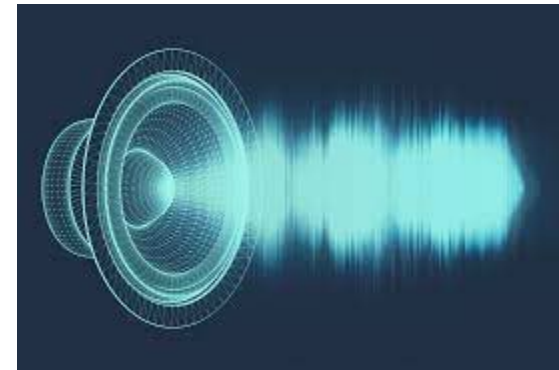
- As humans: media we can see, read, and hear
  - Words, imagery, sounds, tables, plots



<https://www.rd.com/list/funny-photos/>



<https://fileinfo.com/extension/txt>



<https://www.canto.com/blog/audio-file-types/>

Sometimes, we can transform the data while preserving much or all of the information

- Resize an image
- Rephrase a paragraph
- 1.5x an audio book

Sometimes, we can even transform the data so that it is more informative (to humans)

- Perform denoising on an image
- Identify key points and insights in a document
- Remove background noise from audio
- None of these operations add information to the data, but they re-organize and/or remove distracting information

# In computers, data are numbers

- The numbers do not “mean” anything by themselves
- The meaning comes from the way the numbers were produced and how they relate to each other
- The meaning can be contained in each number by itself, or commonly by patterns in groups of numbers

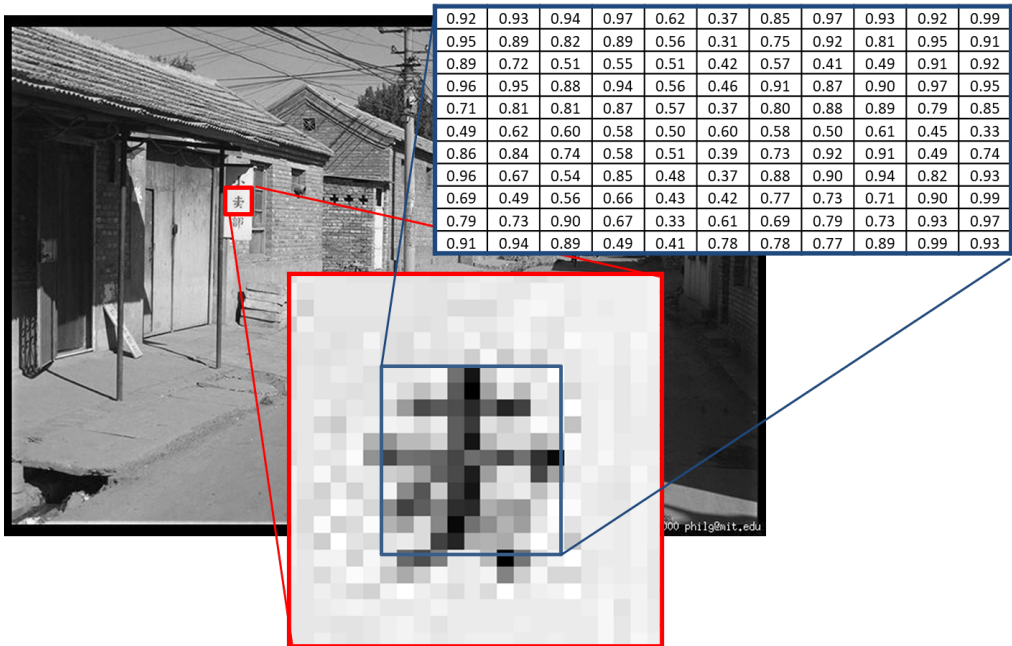
We can transform the data while preserving much or all of the information

- Add or multiply by a constant value
- Represent as a 16-bit or 32-bit float or integer
- Compress a document, or store in a different file format

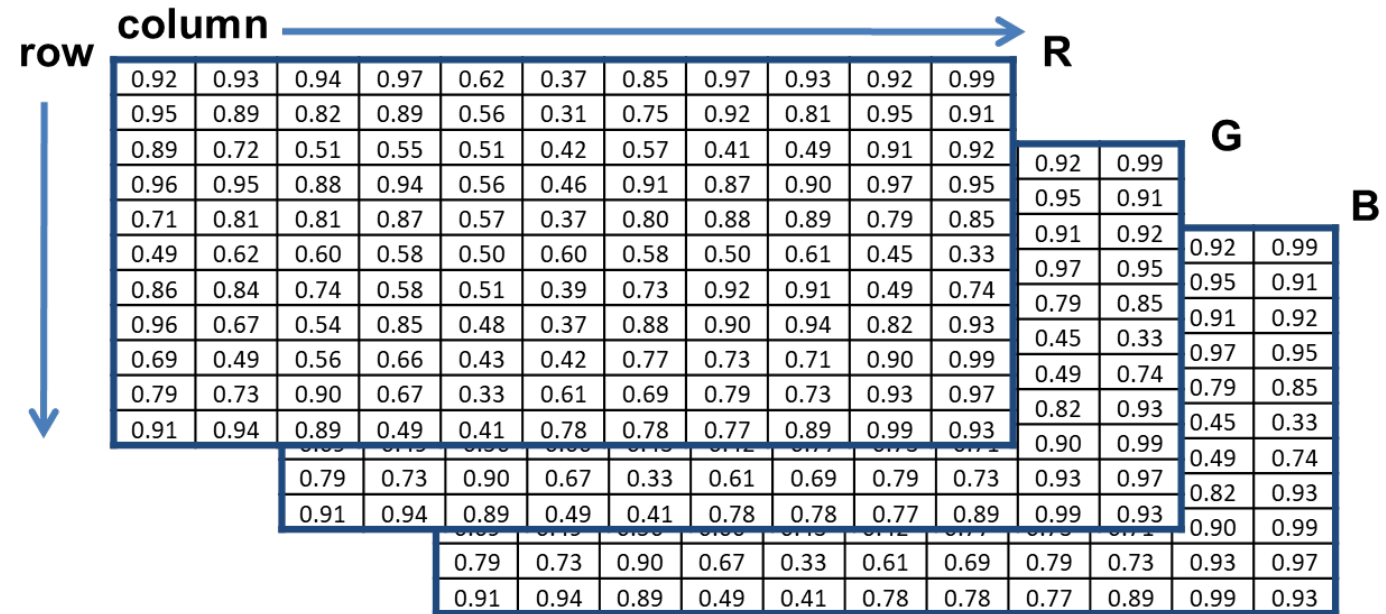
Sometimes, we can even transform the data so that it is more informative (i.e. similarity is more meaningful)

- Center and rescale images of digits so they are easier to compare to each other
- Normalize (subtract means and divide by standard deviations) cancer cell measurements to make simple similarity measures better reflect malignancy
- Select features or create new ones out of combinations of inputs

# Images can be represented as 3D matrices (row, col, color)



|      |      |      |      |      |      |      |      |      |      |      |
|------|------|------|------|------|------|------|------|------|------|------|
| 0.92 | 0.93 | 0.94 | 0.97 | 0.62 | 0.37 | 0.85 | 0.97 | 0.93 | 0.92 | 0.99 |
| 0.95 | 0.89 | 0.82 | 0.89 | 0.56 | 0.31 | 0.75 | 0.92 | 0.81 | 0.95 | 0.91 |
| 0.89 | 0.72 | 0.51 | 0.55 | 0.51 | 0.42 | 0.57 | 0.41 | 0.49 | 0.91 | 0.92 |
| 0.96 | 0.95 | 0.88 | 0.94 | 0.56 | 0.46 | 0.91 | 0.87 | 0.90 | 0.97 | 0.95 |
| 0.71 | 0.81 | 0.81 | 0.87 | 0.57 | 0.37 | 0.80 | 0.88 | 0.89 | 0.79 | 0.85 |
| 0.49 | 0.62 | 0.60 | 0.58 | 0.50 | 0.60 | 0.58 | 0.50 | 0.61 | 0.45 | 0.33 |
| 0.86 | 0.84 | 0.74 | 0.58 | 0.51 | 0.39 | 0.73 | 0.92 | 0.91 | 0.49 | 0.74 |
| 0.96 | 0.67 | 0.54 | 0.85 | 0.48 | 0.37 | 0.88 | 0.90 | 0.94 | 0.82 | 0.93 |
| 0.69 | 0.49 | 0.56 | 0.66 | 0.43 | 0.42 | 0.77 | 0.73 | 0.71 | 0.90 | 0.99 |
| 0.79 | 0.73 | 0.90 | 0.67 | 0.33 | 0.61 | 0.69 | 0.79 | 0.73 | 0.93 | 0.97 |
| 0.91 | 0.94 | 0.89 | 0.49 | 0.41 | 0.78 | 0.78 | 0.77 | 0.89 | 0.99 | 0.93 |



# We can change the structure of data to make it easier to process

Image as matrix

|      |      |      |      |      |      |      |      |      |      |      |
|------|------|------|------|------|------|------|------|------|------|------|
| 0.92 | 0.93 | 0.94 | 0.97 | 0.62 | 0.37 | 0.85 | 0.97 | 0.93 | 0.92 | 0.99 |
| 0.95 | 0.89 | 0.82 | 0.89 | 0.56 | 0.31 | 0.75 | 0.92 | 0.81 | 0.95 | 0.91 |
| 0.89 | 0.72 | 0.51 | 0.55 | 0.51 | 0.42 | 0.57 | 0.41 | 0.49 | 0.91 | 0.92 |
| 0.96 | 0.95 | 0.88 | 0.94 | 0.56 | 0.46 | 0.91 | 0.87 | 0.90 | 0.97 | 0.95 |
| 0.71 | 0.81 | 0.81 | 0.87 | 0.57 | 0.37 | 0.80 | 0.88 | 0.89 | 0.79 | 0.85 |
| 0.49 | 0.62 | 0.60 | 0.58 | 0.50 | 0.60 | 0.58 | 0.50 | 0.61 | 0.45 | 0.33 |
| 0.86 | 0.84 | 0.74 | 0.58 | 0.51 | 0.39 | 0.73 | 0.92 | 0.91 | 0.49 | 0.74 |
| 0.96 | 0.67 | 0.54 | 0.85 | 0.48 | 0.37 | 0.88 | 0.90 | 0.94 | 0.82 | 0.93 |
| 0.69 | 0.49 | 0.56 | 0.66 | 0.43 | 0.42 | 0.77 | 0.73 | 0.71 | 0.90 | 0.99 |
| 0.79 | 0.73 | 0.90 | 0.67 | 0.33 | 0.61 | 0.69 | 0.79 | 0.73 | 0.93 | 0.97 |
| 0.91 | 0.94 | 0.89 | 0.49 | 0.41 | 0.78 | 0.78 | 0.77 | 0.89 | 0.99 | 0.93 |

**Convenient for local pattern analysis**

Image as vector

|      |
|------|
| 0.92 |
| 0.95 |
| 0.89 |
| 0.96 |
| 0.71 |
| 0.49 |
| 0.86 |
| 0.96 |
| 0.69 |
| 0.79 |
| 0.91 |
| 0.93 |
| 0.89 |
| 0.72 |
| 0.95 |
| 0.81 |
| 0.62 |
| 0.84 |
| 0.67 |
| 0.49 |
| 0.73 |
| 0.94 |
| ...  |
| 0.93 |

**Convenient for linear projection or computing similarity**

This does not change the information in the data.



# Text can be represented as a sequence of integers

- Each character can map to a byte value, and then we have a sequence of bytes

`"Dog ate" → [4 15 7 27 1 20 5]`

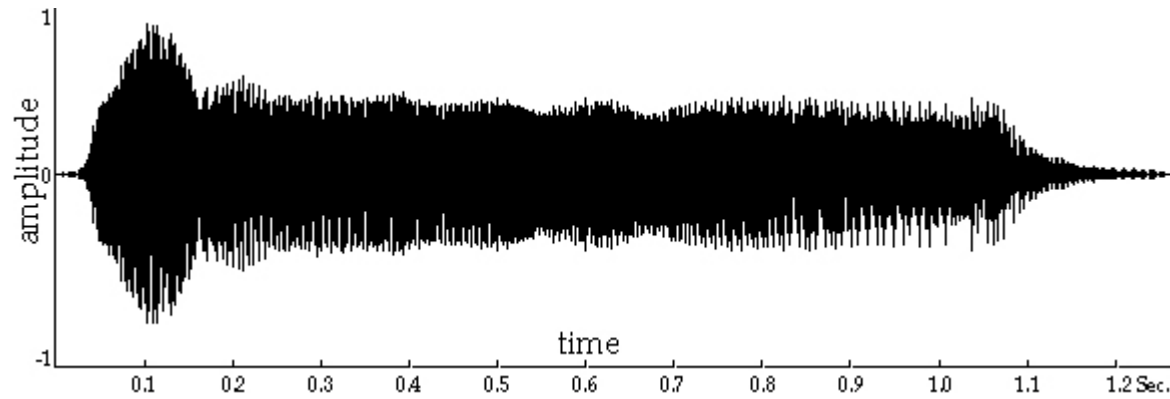
- Each complete word can map to an integer value, and we have a sequence of integers

`"Dog ate" → [437 1256]`

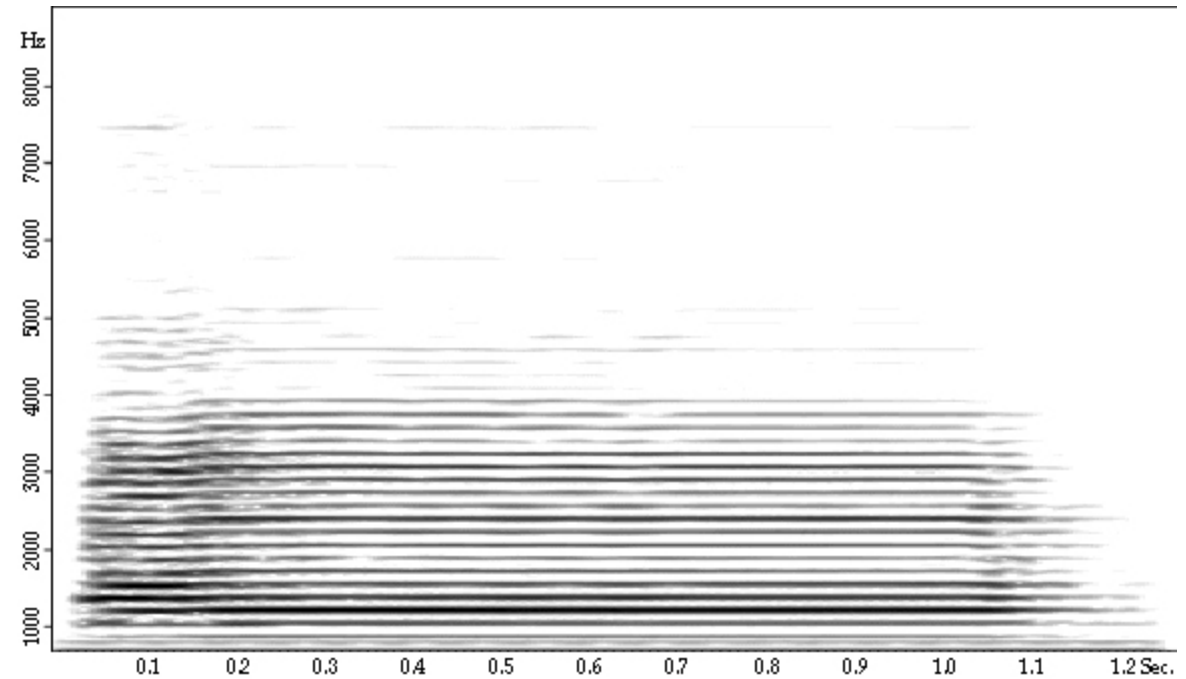
- Common groups of letters can be mapped to subwords and then to integers

`"Bedroom 1521" → [bed-room- -1-5-2-1] → [125 631 27 28 32 29 27]`

# Audio can be represented as a waveform or spectrum



Amplitude vs Time



Frequency-Amplitude vs Time

# Other kinds of data

- Measurements and continuous values typically represented as floating point numbers
  - Temperature, length, area, dollars
- Categorical values represented as integers or one-hot vectors
  - Integer: Happy/Indifferent/Sad  $\rightarrow$  0/1/2
  - One-hot: Happy  $\rightarrow$  [1 0 0]
  - Another example: Red/Green/Blue/Orange/Other  $\rightarrow$  0/1/2/3/4
- Different kinds of values (text, images, measurements) can be reshaped and concatenated into a long feature vector

The same information content can be represented in many ways.

If the original numbers can be recovered, then a change in representation does not change the information content.

All types of data can be stored as 1D vectors/arrays.

# From data point to data set

$\mathbf{x} = \{x_0, \dots, x_M\} \sim D$ :  $\mathbf{x}$  is an  $M$ -dimensional vector drawn from some **distribution**  $D$

We can sample many  $\mathbf{x}$  (e.g. download documents from the Internet, take pictures, take measurements) to create a **dataset**

$$\mathbf{X} = \{\mathbf{x}_0, \dots, \mathbf{x}_N\}$$

We may repeat this collection multiple times, or collect one large dataset and randomly sample it to get  $\mathbf{X}_{train}, \mathbf{X}_{test}$

Typically, we assume that all of the data samples within  $\mathbf{X}_{train}$  and  $\mathbf{X}_{test}$  come from the same distribution and are independent of each other (called **i.i.d. = independent and identically distributed**). That means, e.g. that  $\mathbf{x}_0$  does not tell us anything about  $\mathbf{x}_1$  if we already know the sampling distribution  $D$

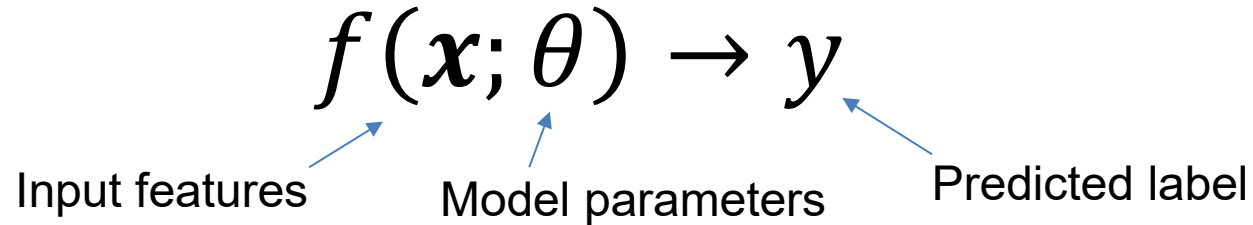
For prediction problems, we often have pairs of features / target labels:  $(\mathbf{X}, \mathbf{y}) = \{(\mathbf{x}_0, y_0), \dots, (\mathbf{x}_N, y_N)\}$

# Answer Q1











<https://tinyurl.com/441-fa24-L2>



# Classification problem













## Digit classification example

|     |   |   |   |  |   |   |   |   |   |   |
|-----|---|---|---|--|---|---|---|---|---|---|
| $x$ |  |  |  |  |  |  |  |  |  |  |
| $y$ | 3   | 8   | 7   | 9  | 9   | 0   | 1   | 1   | 5   | 2   |

Developing a classifier involves working with training, validation, and test data

- *Training* data: used to fit parameters of the model
- *Validation* data: used to select the best model and any parameters that need to be manually set (“hyperparameters”)
- *Test* data: used to evaluate the final version of the model

# Introduction to MNIST, a classification benchmark

|     |   |   |   |  |   |   |   |   |   |   |
|-----|---|---|---|--|---|---|---|---|---|---|
| $x$ |  |  |  |  |  |  |  |  |  |  |
| $y$ | 3   | 8   | 7   | 9  | 9   | 0   | 1   | 1   | 5   | 2   |

- 60,000 training samples
- 10,000 test samples
- Each sample has features (pixel intensities)  $x \in R^{784}$ , each value in the range of 0 to 1, and a label (digit)  $y \in [0, 1, \dots, 9]$



# MNIST Processing

- `x_train[0]` is the features of the first training sample
- `y_train[0]` is the label of the first training sample
- `x_train[:1000]` is the features of the first 1000 training samples

```
# initialization code
import numpy as np
from keras.datasets import mnist
%matplotlib inline
from matplotlib import pyplot as plt
from scipy import stats

def load_mnist():
    """
    Loads, reshapes, and normalizes the data
    """
    (x_train, y_train), (x_test, y_test) = mnist.load_data() # loads MNIST data
    x_train = np.reshape(x_train, (len(x_train), 28*28)) # reformat to 784-d vectors
    x_test = np.reshape(x_test, (len(x_test), 28*28))
    maxval = x_train.max()
    x_train = x_train/maxval # normalize values to range from 0 to 1
    x_test = x_test/maxval
    return (x_train, y_train), (x_test, y_test)

def display_mnist(x, subplot_rows=1, subplot_cols=1):
    """
    Displays one or more examples in a row or a grid
    """
    if subplot_rows>1 or subplot_cols>1:
        fig, ax = plt.subplots(subplot_rows, subplot_cols, figsize=(15,15))
        for i in np.arange(len(x)):
            ax[i].imshow(np.reshape(x[i], (28,28)), cmap='gray')
            ax[i].axis('off')
    else:
        plt.imshow(np.reshape(x, (28,28)), cmap='gray')
        plt.axis('off')
    plt.show()
```

# Nearest neighbor algorithm

For given test features, assign the label / target value of the most similar training features

1.  $i^* = \underset{i}{\operatorname{argmin}} \operatorname{distance}(X_{\text{train}}[i], X_{\text{test}})$
2.  $y_{\text{test}} = y_{\text{train}}[i^*]$

Distance is typically sum of squared distance or cosine distance.

There is no training! The accuracy depends on the features, and how much data you have.

# Nearest neighbor algorithm

```
def nearest_neighbor_classifier(X_query, X_train, y_train):
    nearest_i = -1
    mindist = np.Inf
    for i in np.arange(len(X_train)):
        dist = np.sum((X_query-X_train[i])**2)
        if dist < mindist:
            mindist = dist
            nearest_i = i
    nearest_label = y_train[nearest_i]
    return nearest_label
```

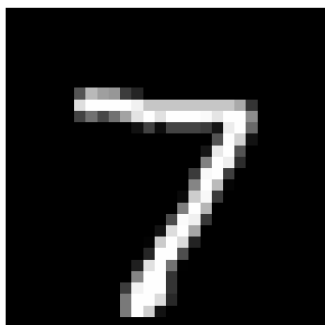
# Nearest neighbor algorithm (concise)

```
def nearest_neighbor_classifier(X_query, X_train, y_train):  
    nearest_i = np.argmin(np.sum((X_query-X_train)**2, axis=1))  
    nearest_label = y_train[nearest_i]  
    return nearest_label
```

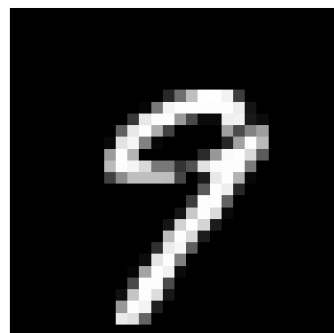
# Example Predictions

Nearest Training Image and Label with Varying Number of Training Samples ( $N$ )

Test image

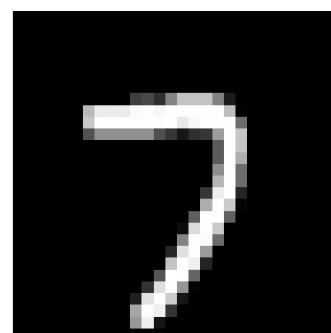


$N = 100$



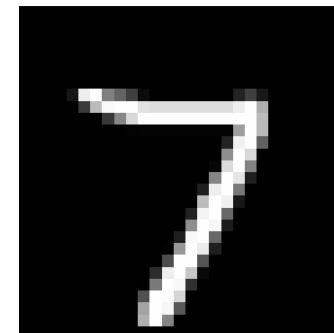
9

$N = 1000$

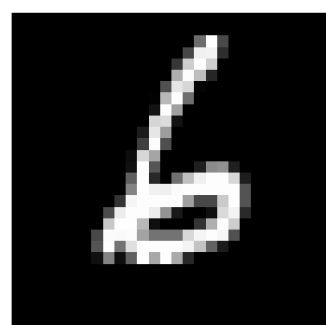


7

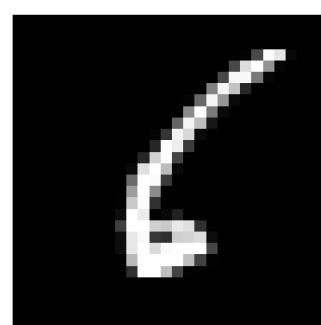
$N = 10000$



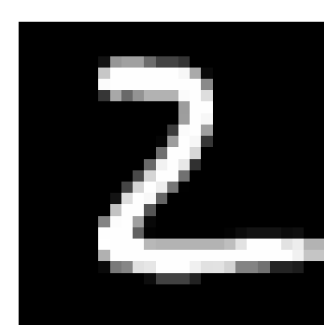
7



6

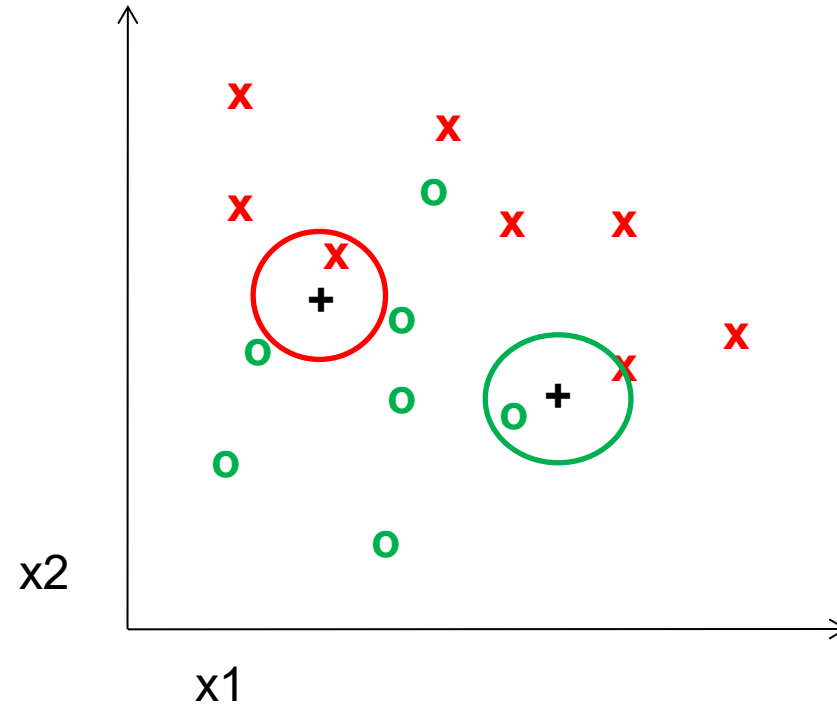


6

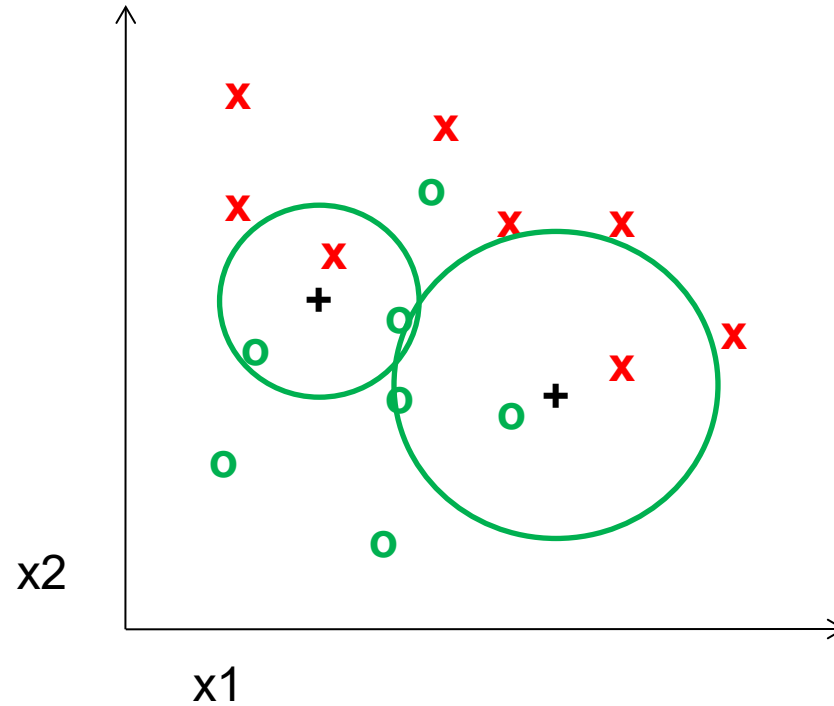


2

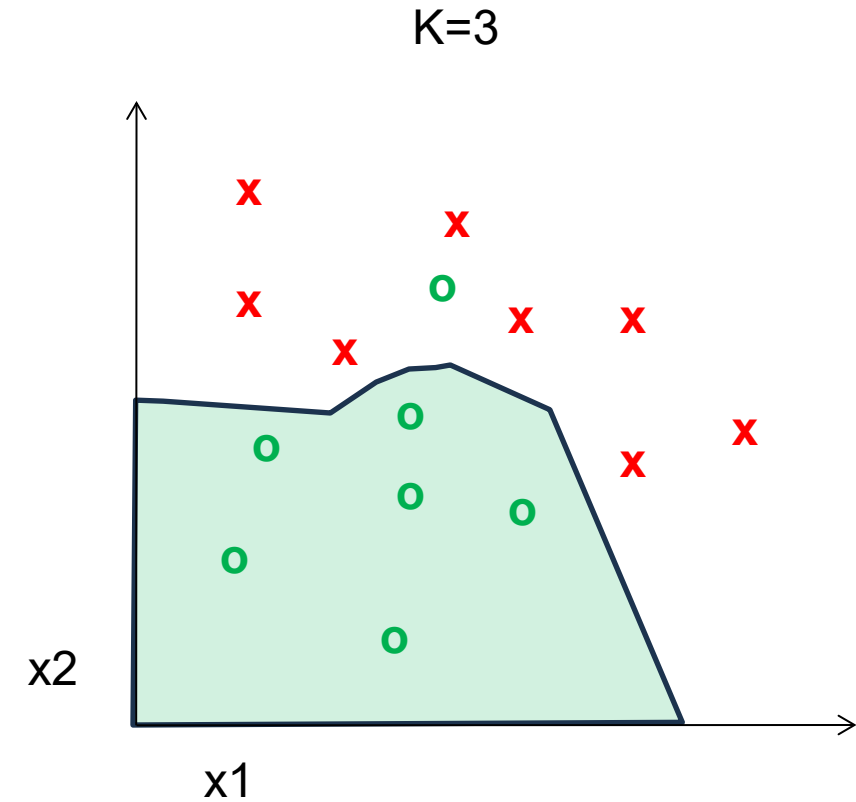
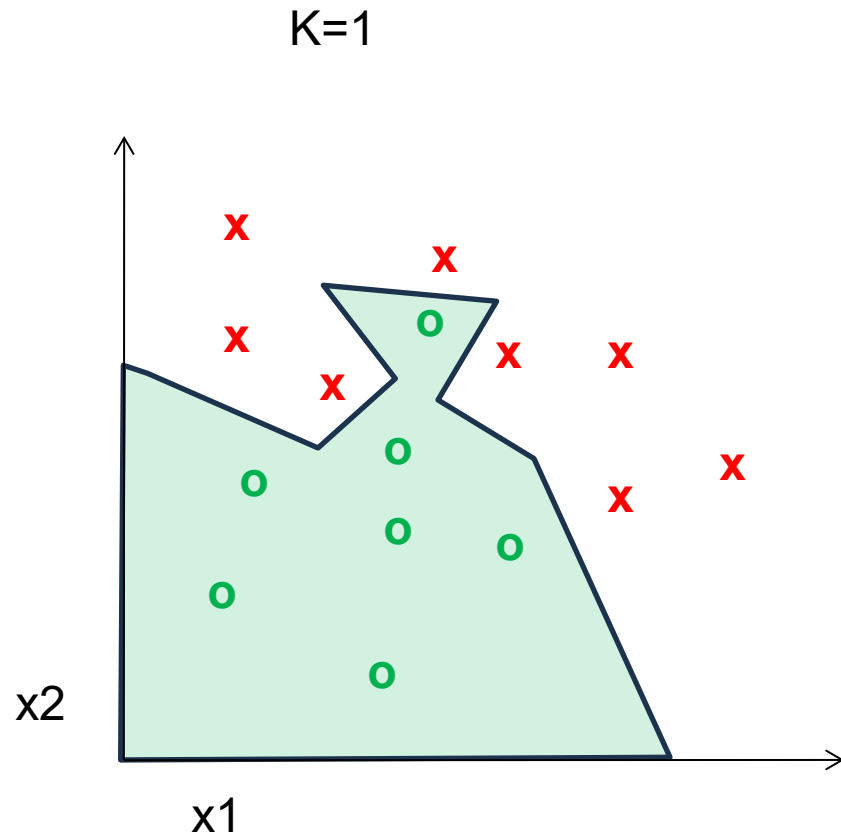
# 1-nearest neighbor (KNN, K=1)



# 3-nearest neighbor (KNN, K=3)

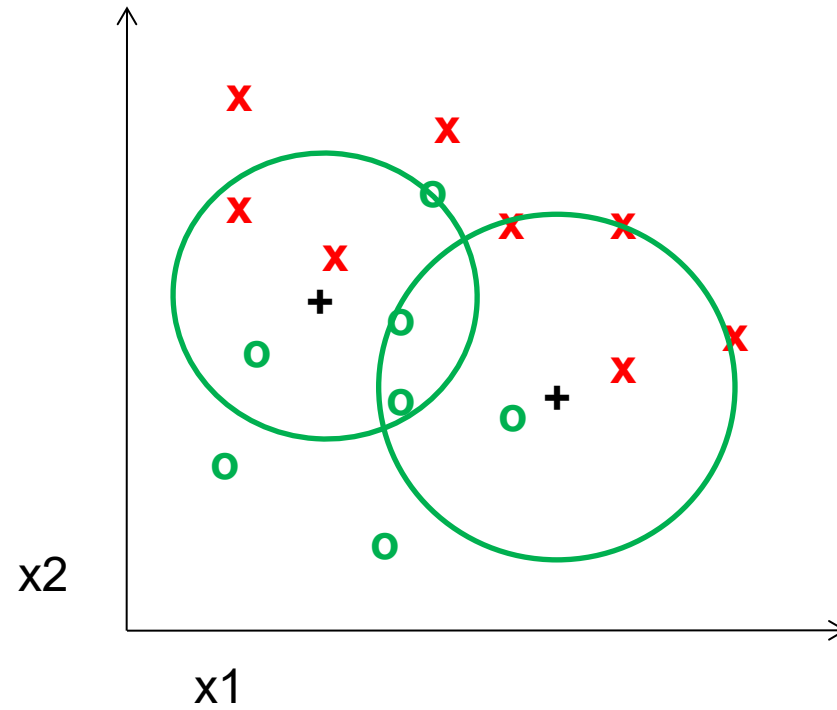


KNN: Increasing K gives a smoother decision boundary  
(less sensitive to unusual data points)





# 5-nearest neighbor (KNN, K=5)



# Answer Q2

<https://tinyurl.com/441-fa24-L2>



# Comments on K-NN

- Simple: an excellent baseline and sometimes hard to beat
  - Naturally scales with data: it may be the only choice when you have one example per class, and is still often achieves good performance when you have many
  - Higher K gives smoother functions
  - Can work with only one example per class
- Naïve implementations are slow, but fast retrieval methods are available (as we'll see in Lecture 4)
- No training time (other than storing/indexing examples)
- With infinite examples, 1-NN provably has error that is at most twice Bayes optimal error (but we never have infinite examples)

# How do we measure and analyze classification performance?

- Evaluate on the test set after all parameters are determined
- Classification error:  $\text{err}(\mathbf{X}) = \frac{1}{N} \sum_i f(X_i) \neq y_i$ 
  - Percent of examples that are wrongly predicted
- Classification accuracy:  $\text{acc}(\mathbf{X}) = 1 - \text{err}(\mathbf{X})$
- Confusion matrix
  - For each true label, what number/fraction of predictions correspond to each label, i.e.  $P(\text{predicted label} \mid \text{true label})$
  - If not normalized, it's a count; otherwise, a fraction

# Example

Answer Q3 and Q4

Classifier predicts whether a team will lose, tie, or win

<https://tinyurl.com/441-fa24-L2>

| True | Predicted |
|------|-----------|
| Win  | Win       |
| Loss | Tie       |
| Tie  | Tie       |
| Win  | Win       |
| Tie  | Loss      |



# Performance measures are an estimate

E.g., suppose we have 1,000 training samples and measure an error rate of 8% on 100 test samples.

- With a different set of test samples, we **expect** to see the same error rate, but it could **vary**
- With a different set of the same number of training samples, we also expect to see the same error rate, but with variance
- If we increase the number of test samples, the expected test error doesn't change, but the variance in the estimate decreases
- If we increase the number of training samples, the expected test error will be lower

# KNN Usage Example: Deep Face

## DeepFace: Closing the Gap to Human-Level Performance in Face Verification

Yaniv Taigman

Ming Yang

Marc'Aurelio Ranzato

Lior Wolf

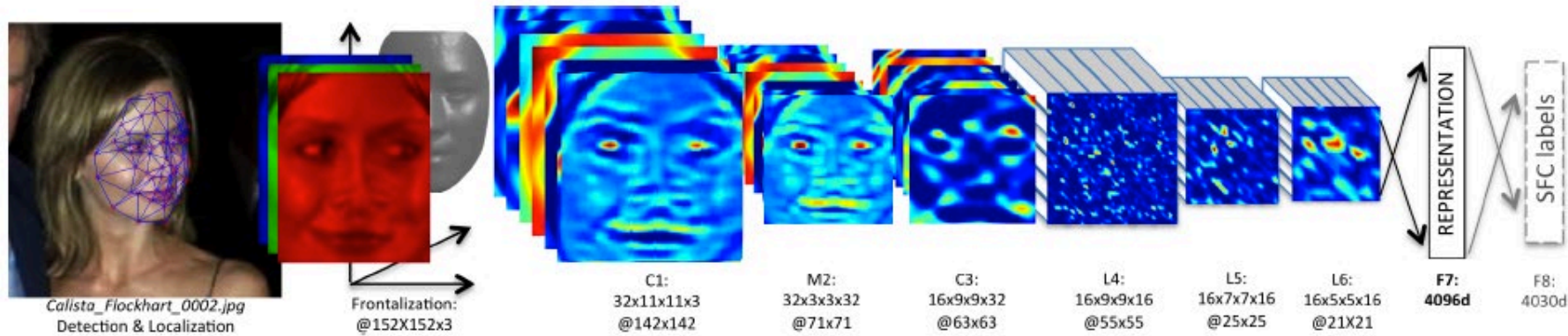
Facebook AI Research  
Menlo Park, CA, USA

{yaniv, mingyang, ranzato}@fb.com

Tel Aviv University  
Tel Aviv, Israel

wolf@cs.tau.ac.il

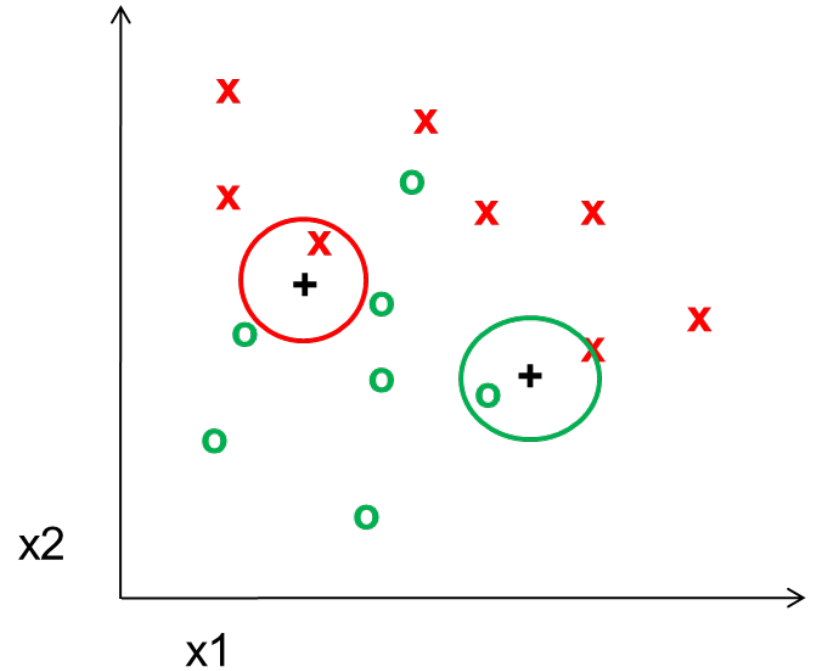
CVPR 2014



1. Detect facial features
  2. Align faces to be frontal
  3. Extract features using deep network while training classifier to label image into person (dataset based on employee faces)
  4. In testing, extract features from deep network and use nearest neighbor classifier to assign identity
- Performs similarly to humans in the LFW dataset (labeled faces in the wild)
  - Can be used to organize photo albums, identifying celebrities, or alert user when someone posts an image of them
  - If this is used in a commercial deployment, what might be some unintended consequences?
  - This algorithm is/was used by Facebook (though with expanded training data)

# Things to remember

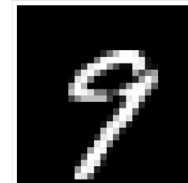
- Foundation of ML: similar features predict similar labels
  - Hard part: How to represent inputs with vectors that reflect the similarity
- KNN is a simple but effective classifier that predicts the label of the most similar training example(s)
  - Accuracy depends on quality of features and number of training samples
- Larger K gives a smoother prediction function
- Measure classification performance with error and confusion matrices



Test image

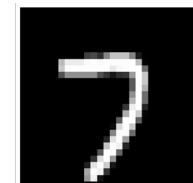


$N = 100$



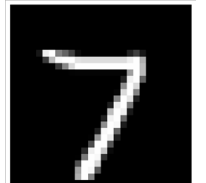
9

$N = 1000$



7

$N = 10000$



7



# Coming up

- Mon: Holiday!
- Tues: Regression with KNN, Generalization
- Thurs: Search and Clustering



# Example

Classifier predicts whether a team will lose, tie, or win

| True | Predicted |
|------|-----------|
| Win  | Win       |
| Loss | Tie       |
| Tie  | Tie       |
| Win  | Win       |
| Tie  | Loss      |

Error = 40%

Normalized Confusion Matrix

|      |   | Predicted |     |   |
|------|---|-----------|-----|---|
|      |   | L         | T   | W |
| True | L | 0         | 1   | 0 |
|      | T | 0.5       | 0.5 | 0 |
|      | W | 0         | 0   | 1 |

# KNN Summary

- Key Assumptions
  - Samples with similar input features will have similar output predictions
  - Depending on distance measure, may assume all dimensions are equally important
- Model Parameters
  - Features and predictions of the training set
- Designs
  - K (number of nearest neighbors to use for prediction)
  - How to combine multiple predictions if  $K > 1$
  - Feature design (selection, transformations)
  - Distance function (e.g. L2, L1, Mahalanobis)
- When to Use
  - Few examples per class, many classes
  - Features are all roughly equally important
  - Training data available for prediction changes frequently
  - Can be applied to classification or regression, with discrete or continuous features
  - Most powerful when combined with feature learning
- When Not to Use
  - Many examples are available per class (feature learning with linear classifier may be better)
  - Limited storage (cannot store many training examples)
  - Limited computation (linear model may be faster to evaluate)