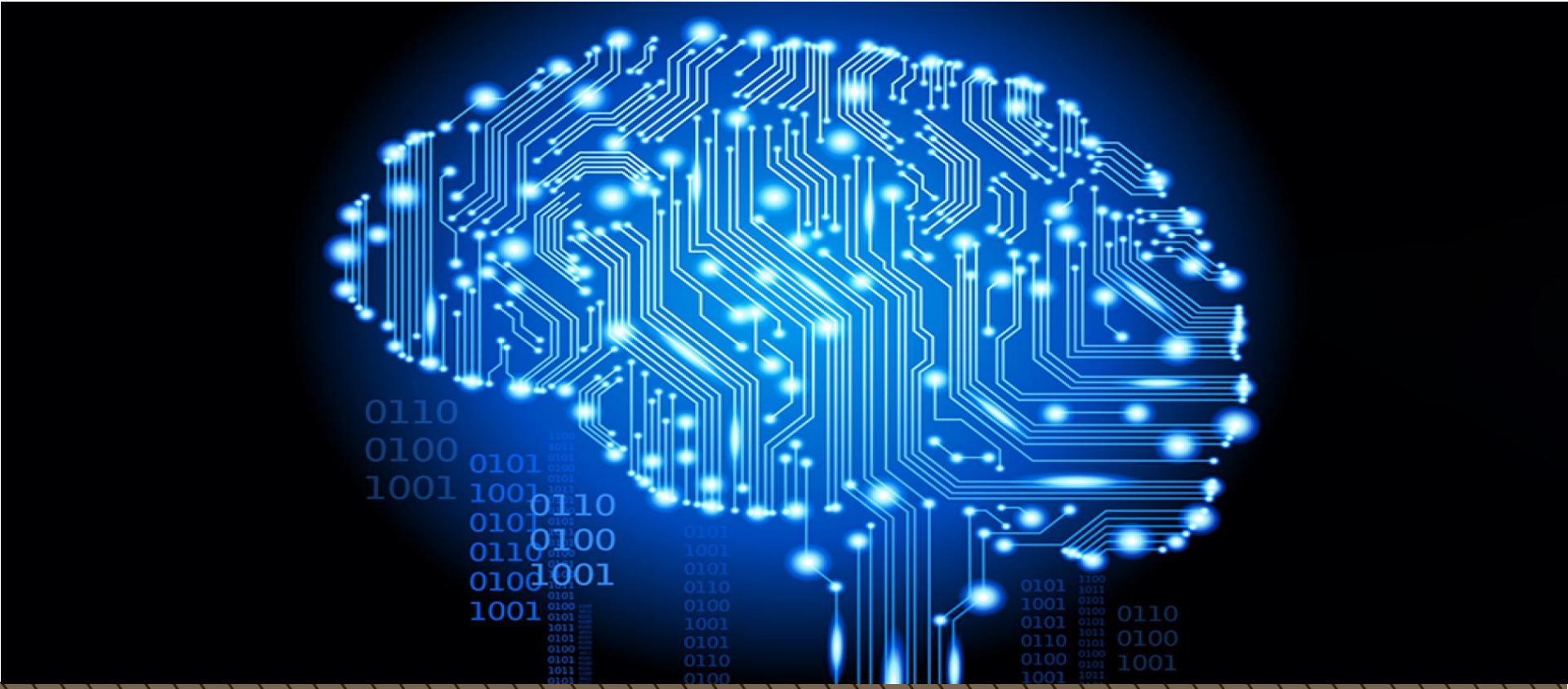


Machine Learning

Lecture 6:
KNN
Image Classification

Image Classification

Multi-class classification

- Direct approaches –
  - k-NN clasifier
  - Bayesian classifier
  - Decision trees
  - Artificial Neural network $\rightarrow$ (e.g., Convolution neural nets)
  - k-class SVM (support vector machines)
- Indirect approaches (i.e., using a binary classifier) –
  - One-vs-All strategy
  - All-vs-All strategy
  - Error correcting codes

Image Classification

A core task in Computer Vision

(assume given set of discrete labels) {dog, cat, truck, plane, ...}

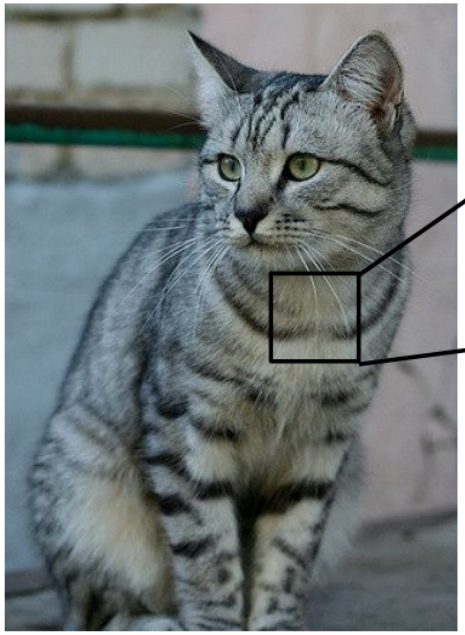
cat

**This slide is from CS231n Stanford University*

Image Classification

A core task in Computer Vision

The Problem: Semantic Gap



This image by Nikita is licensed under CC-BY 2.0

```
[[105 112 108 111 104  99 106  99  96 103 112 119 104  97  93  87]
 [ 91  98 102 106 104  79  98 103  99 105 123 136 110 105  94  85]
 [ 76  85  90 105 128 105  87  96  95  99 115 112 106 103  99  85]
 [ 99  81  81  93 120 131 127 100  95  98 102  99  96  93 101  94]
 [106  91  61  64  69  91  88  85 101 107 109  98  75  84  96  95]
 [114 108  85  55  55  69  64  54  64  87 112 129  98  74  84  91]
 [133 137 147 103  65  81  80  65  52  54  74  84 102  93  85  82]
 [128 137 144 140 109  95  86  70  62  65  63  63  60  73  86 101]
 [125 133 148 137 119 121 117  94  65  79  80  65  54  64  72  98]
 [127 125 131 147 133 127 126 131 111  96  89  75  61  64  72  84]
 [115 114 109 123 150 148 131 118 113 109 100  92  74  65  72  78]
 [ 89  93  90  97 108 147 131 118 113 114 113 109 106  95  77  80]
 [ 63  77  86  81  77  79 102 123 117 115 117 125 125 130 115  87]
 [ 62  65  82  89  78  71  80 101 124 126 119 101 107 114 131 119]
 [ 63  65  75  88  89  71  62  81 120 138 135 105  81  98 110 118]
 [ 87  65  71  87 106  95  69  45  76 130 126 107  92  94 105 112]
 [118  97  82  86 117 123 116  66  41  51  95  93  89  95 102 107]
 [164 146 112  80  82 120 124 104  76  48  45  66  88 101 102 109]
 [157 170 157 120  93  86 114 132 112  97  69  55  70  82  99  94]
 [130 128 134 161 139 100 109 118 121 134 114  87  65  53  69  86]
 [128 112  96 117 150 144 120 115 104 107 102  93  87  81  72  79]
 [123 107  96  86  83 112 153 149 122 109 104  75  80 107 112  99]
 [122 121 102  80  82  86  94 117 145 148 153 102  58  78  92 107]
 [122 164 148 103  71  56  78  83  93 103 119 139 102  61  69  84]]
```

What the computer sees

An image is just a big grid of numbers between [0, 255]:

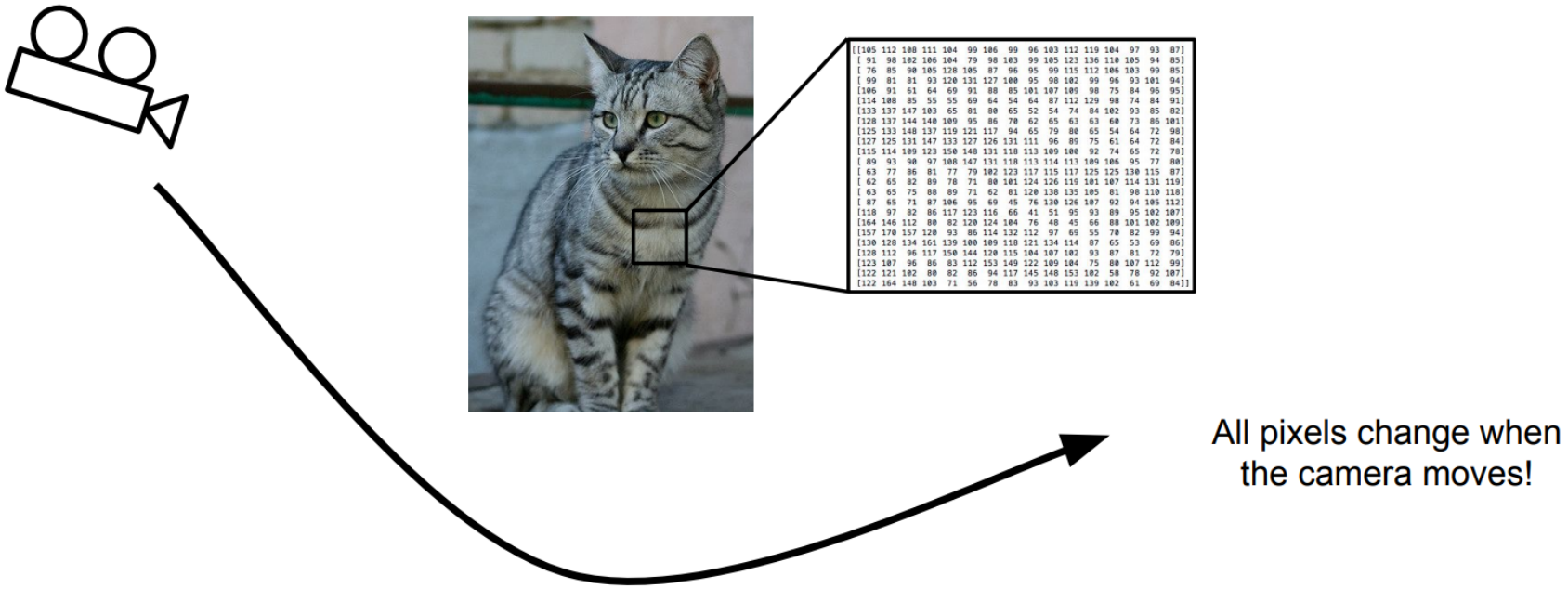
e.g. 800 x 600 x 3
(3 channels RGB)

**This slide is from CS231n Stanford University*

Image Classification

A core task in Computer Vision

Challenges: Viewpoint variation



**This slide is from CS231n Stanford University*

Image Classification

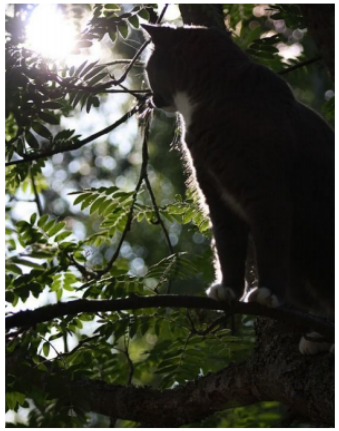
A core task in Computer Vision

Challenges: Illumination

This image is CC0 1.0 public domain

This image is CC0 1.0 public domain

This image is CC0 1.0 public domain



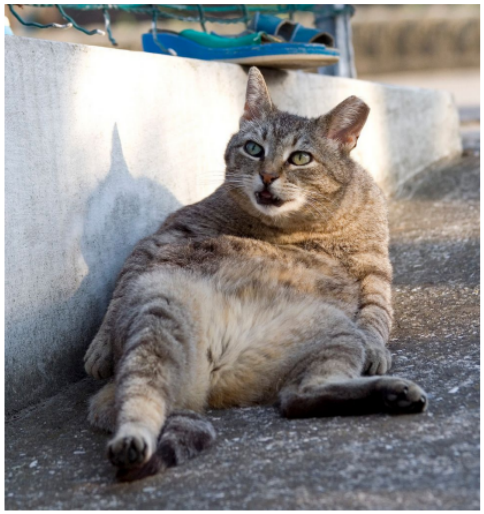
This image is CC0 1.0 public domain

**This slide is from CS231n Stanford University*

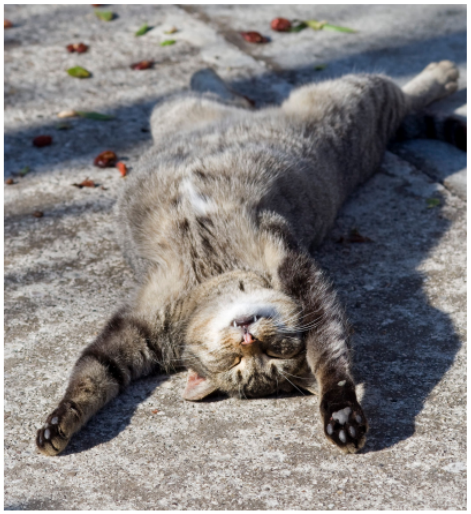
Image Classification

A core task in Computer Vision

Challenges: Deformation

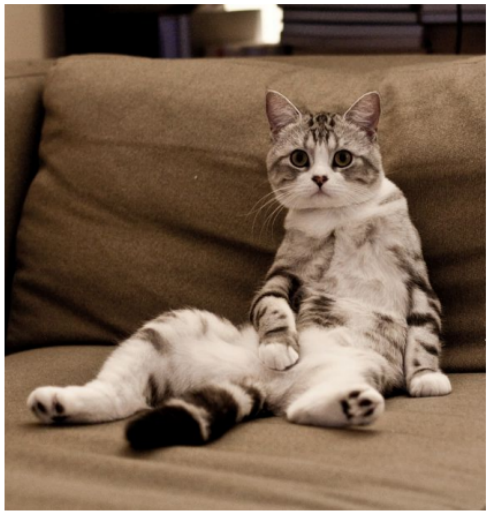


This image by Umberto Salvagnin is licensed under CC-BY 2.0



This image by Umberto Salvagnin is licensed under CC-BY 2.0

This image by sare bear is licensed under CC-BY 2.0



This image by Tom Thai is licensed under CC-BY 2.0

**This slide is from CS231n Stanford University*

Image Classification

A core task in Computer Vision

Challenges: Occlusion

This image is CC0 1.0 public domain

This image is CC0 1.0 public domain

This image by jonsson is licensed under CC-BY 2.0

**This slide is from CS231n Stanford University*

Image Classification

A core task in Computer Vision

Challenges: Background Clutter

This image is CC0 1.0 public domain

This image is CC0 1.0 public domain

**This slide is from CS231n Stanford University*

Image Classification

A core task in Computer Vision

Challenges: Intraclass variation

**This slide is from CS231n Stanford University*

Image Classification

A core task in Computer Vision

An image classifier

```
def classify_image(image):
  # Some magic here?
  return class_label
```

Unlike e.g. sorting a list of numbers,

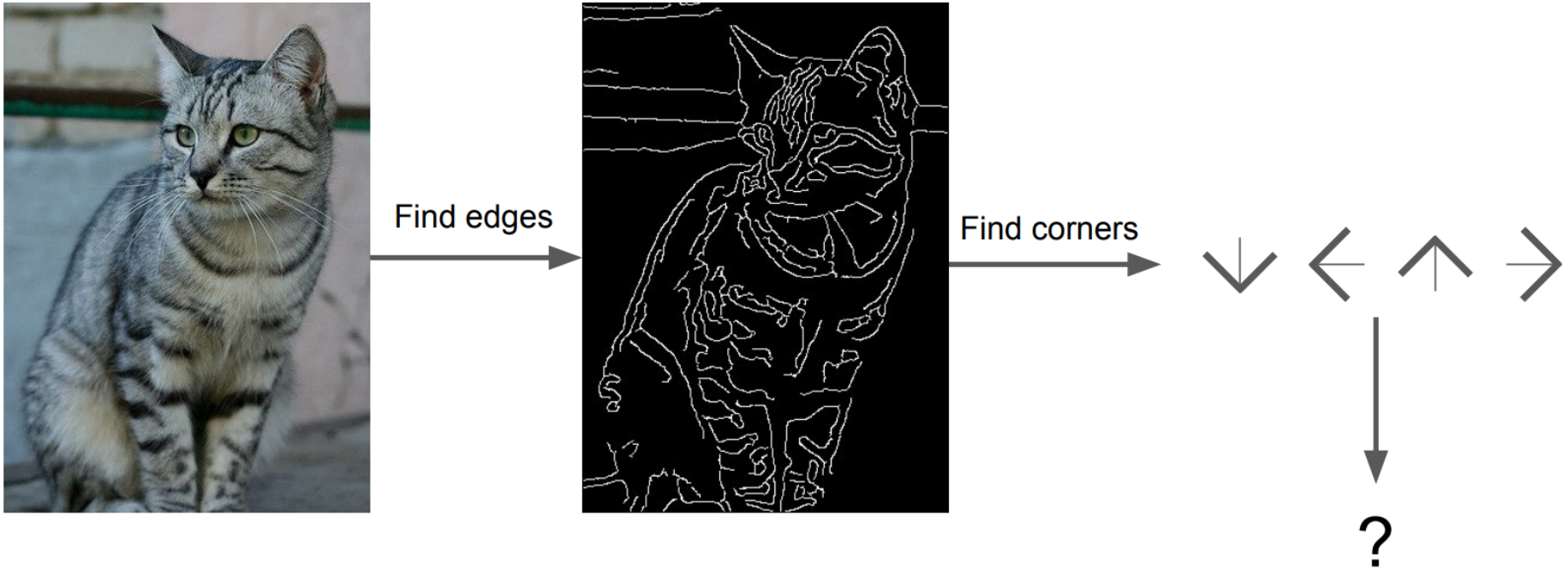
no obvious way to hard-code the algorithm for recognizing a cat, or other classes.

**This slide is from CS231n Stanford University*

Image Classification

A core task in Computer Vision

Attempts have been made



**This slide is from CS231n Stanford University*

Image Classification

A core task in Computer Vision

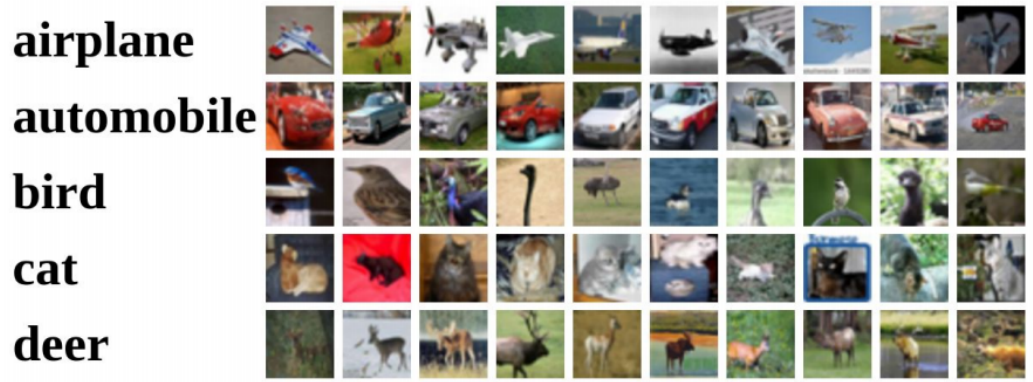
Machine Learning: Data-Driven Approach

1. Collect a dataset of images and labels
2. Use Machine Learning to train a classifier
3. Evaluate the classifier on new images

```
def train(images, labels):
  # Machine learning!
  return model
```

```
def predict(model, test_images):
  # Use model to predict labels
  return test_labels
```

Example training set



**This slide is from CS231n Stanford University*

Image Classification

Nearest Neighbor classifier
K-Nearest Neighbors classifier
Assignment

Image Classification

Nearest Neighbor classifier

Instance based classifier

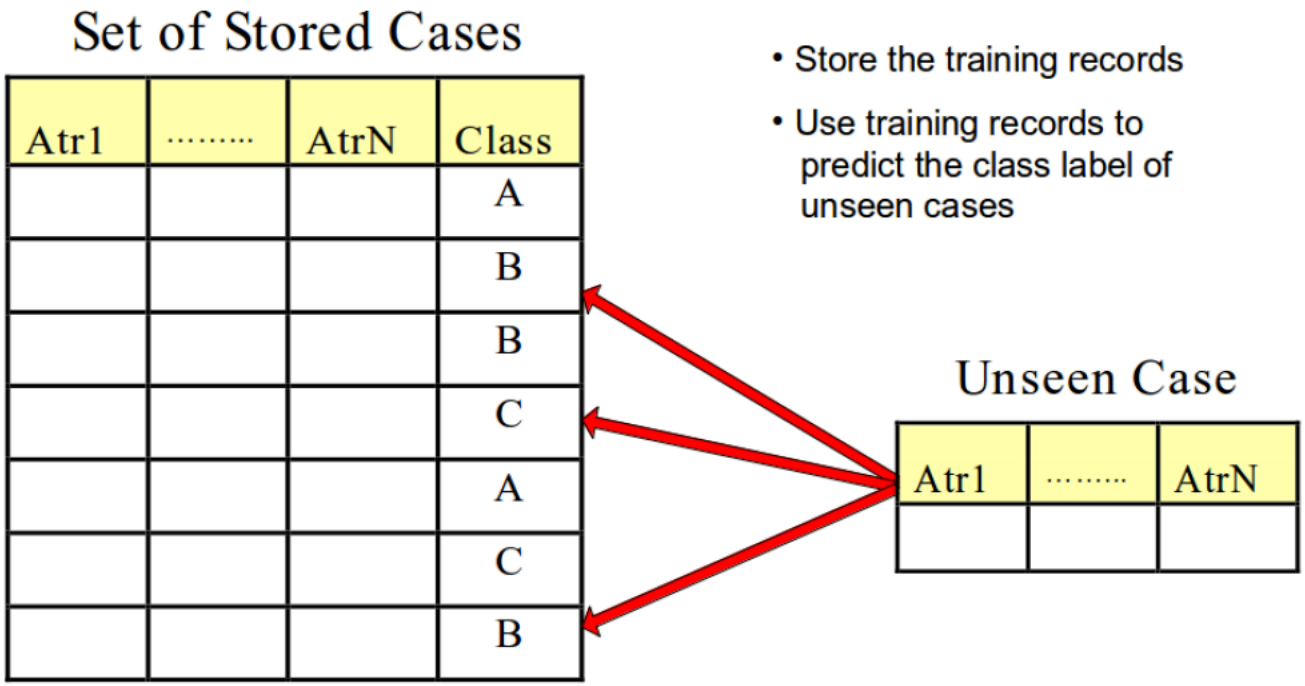


Image Classification

Nearest Neighbor classifier

Training: Nothing just records

Testing: Compare (distance)

Question:

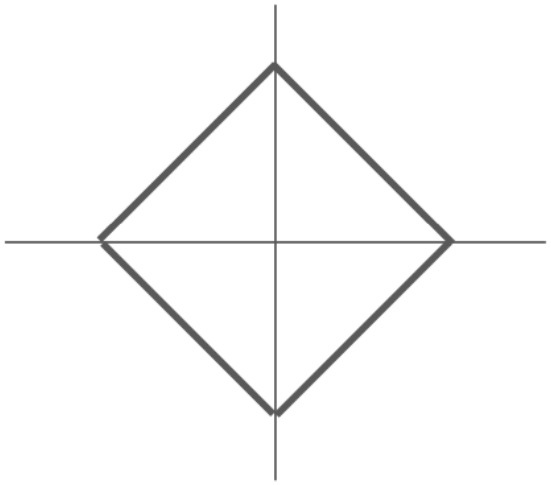
How to compare ?
How to compute the distance, difference?

Image Classification

Nearest Neighbor classifier

L1 (Manhattan) distance

$$d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$$



L2 (Euclidean) distance

$$d_2(I_1, I_2) = \sqrt{\sum_p \left(I_1^p - I_2^p\right)^2}$$

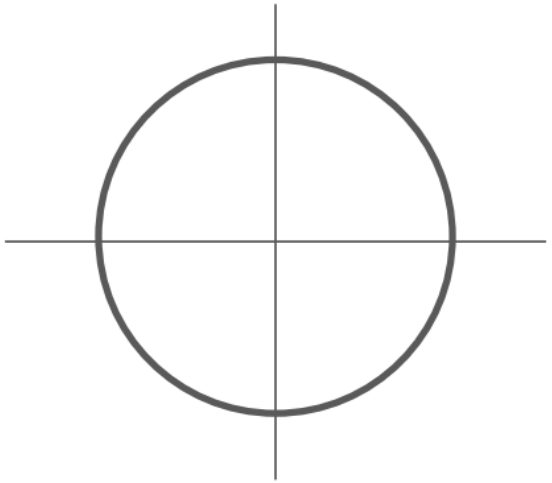


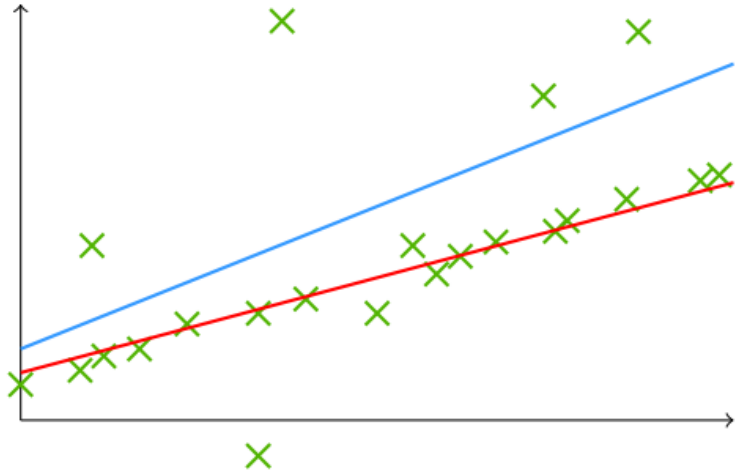
Image Classification

Nearest Neighbor classifier

Given : A set of points in 2-dimension

Goal : Find a line to fit those points

Output : ℓ_2 minimizer line, ℓ_1 minimizer line



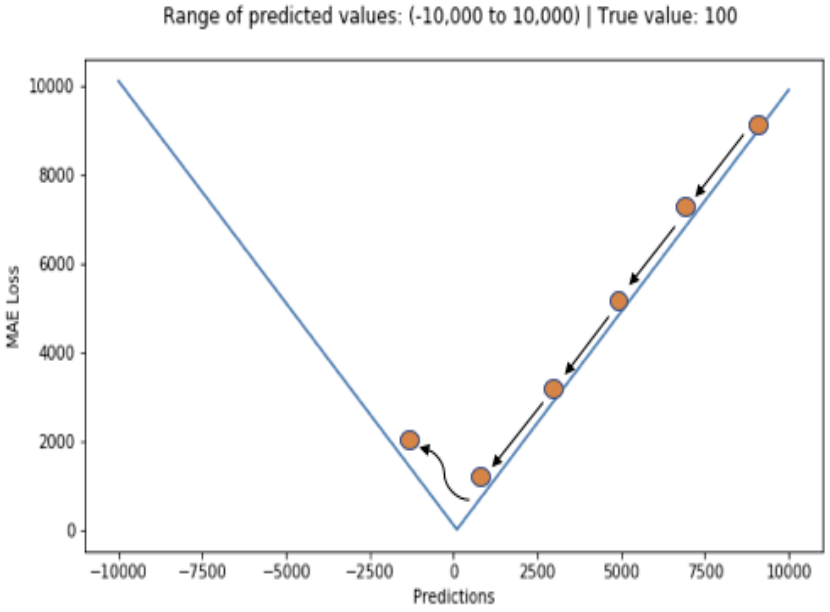
L1 is more robust to outlier

http://www.chioka.in/differences-between-the-l1-norm-and-the-l2-norm-least-absolute-deviations-and-least-squares/

Image Classification

Nearest Neighbor classifier

L2 loss is sensitive to outliers, but gives a more stable and closed form solution (by setting its derivative to 0.)



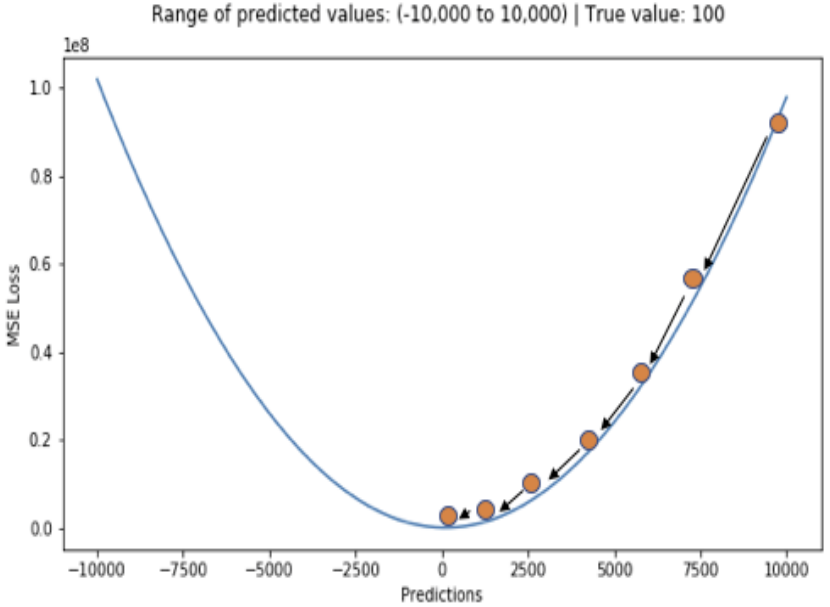


Image Classification

Nearest Neighbor classifier

```python
import numpy as np

class NearestNeighbor:
  def __init__(self):
    pass

  def train(self, X, y):
    """ X is N x D where each row is an example. Y is 1-dimension of size N """
    # the nearest neighbor classifier simply remembers all the training data
    self.Xtr = X
    self.ytr = y

  def predict(self, X):
    """ X is N x D where each row is an example we wish to predict label for """
    num_test = X.shape[0]
    # lets make sure that the output type matches the input type
    Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

    # loop over all test rows
    for i in xrange(num_test):
      # find the nearest training image to the i'th test image
      # using the L1 distance (sum of absolute value differences)
      distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
      min_index = np.argmin(distances) # get the index with smallest distance
      Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

    return Ypred
```

Memorize training data

For each test image:
Find closest train image
Predict label of nearest image

Image Classification

Nearest Neighbor classifier

```python
import numpy as np

class NearestNeighbor:
  def __init__(self):
    pass

  def train(self, X, y):
    """ X is N x D where each row is an example. Y is 1-dimension of size N """
    # the nearest neighbor classifier simply remembers all the training data
    self.Xtr = X
    self.ytr = y

  def predict(self, X):
    """ X is N x D where each row is an example we wish to predict label for """
    num_test = X.shape[0]
    # lets make sure that the output type matches the input type
    Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

    # loop over all test rows
    for i in xrange(num_test):
      # find the nearest training image to the i'th test image
      # using the L1 distance (sum of absolute value differences)
      distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
      min_index = np.argmin(distances) # get the index with smallest distance
      Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

    return Ypred
```

Q: With N examples, how fast are training and prediction?

A: Train O(1),
predict O(N)

This is bad: we want classifiers that are **fast** at prediction; **slow** for training is ok

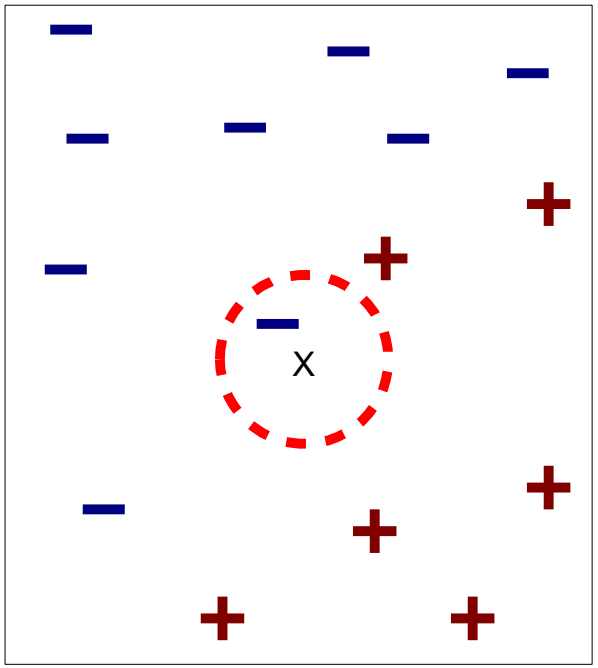
Image Classification

K-Nearest Neighbor classifier

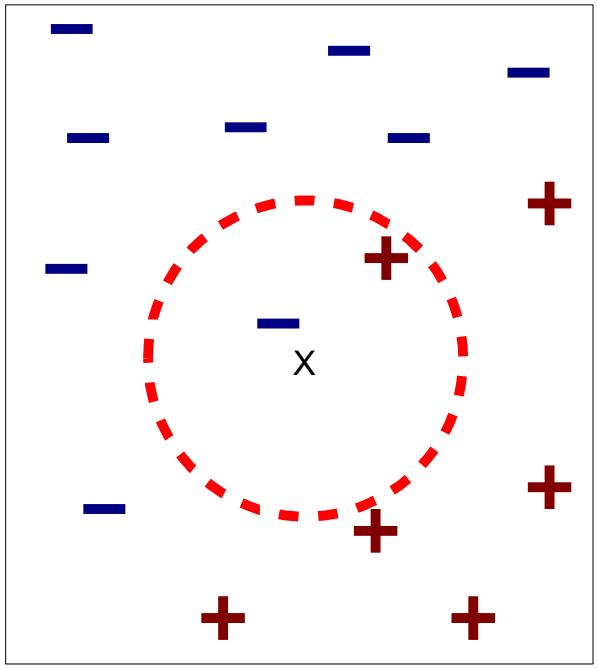
You may have noticed that it is strange to only use the label of the nearest image when we wish to make a prediction. Indeed, it is almost always the case that one can do better by using what's called a **k-Nearest Neighbor Classifier**. The idea is very simple: instead of finding the single closest image in the training set, we will find the top **k** closest images, and have them vote on the label of the test image. In particular, when $k = 1$, we recover the Nearest Neighbor classifier.

Image Classification

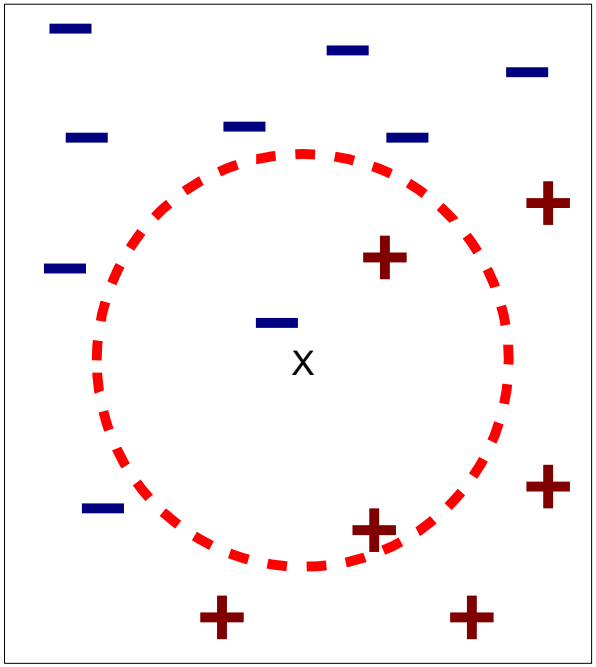
K-Nearest Neighbor classifier



(a) 1-nearest neighbor



(b) 2-nearest neighbor



(c) 3-nearest neighbor

Image Classification

K-Nearest Neighbor classifier

http://vision.stanford.edu/teaching/cs231n-demos/knn/

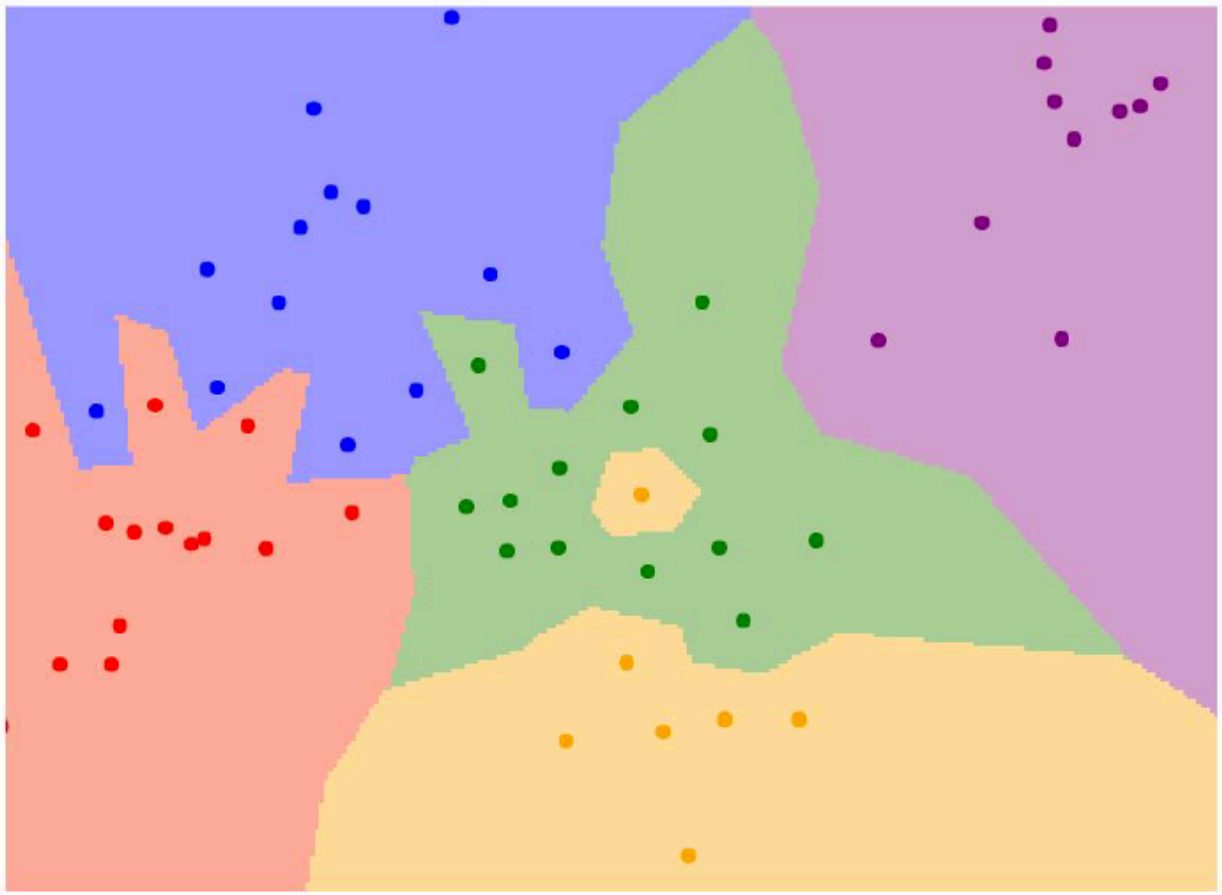
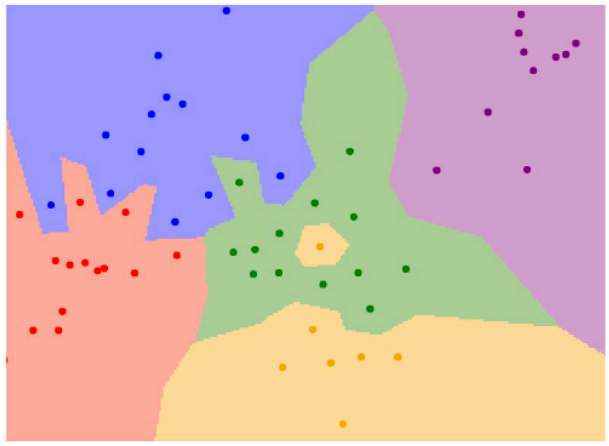


Image Classification

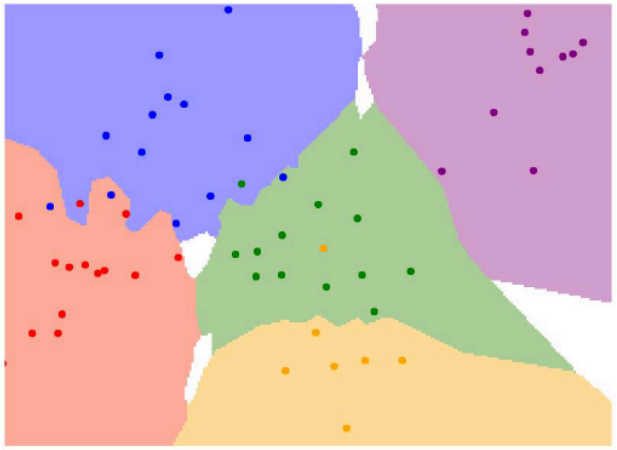
K-Nearest Neighbor classifier

http://vision.stanford.edu/teaching/cs231n-demos/knn/

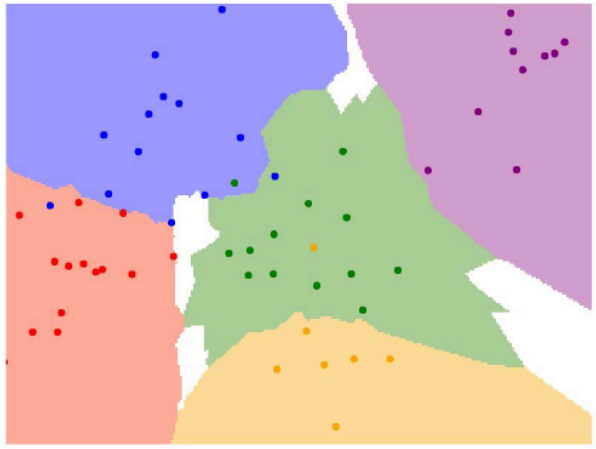
Instead of copying label from nearest neighbor,
take **majority vote** from K closest points



K = 1



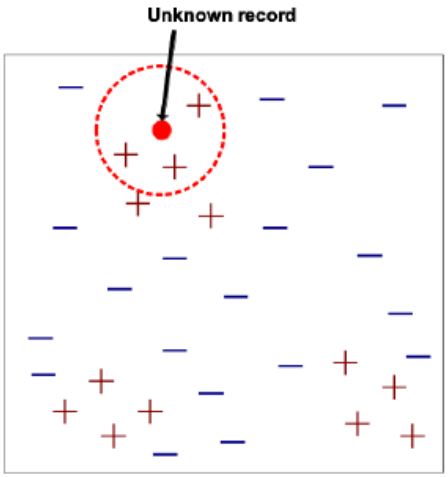
K = 3



K = 5

Image Classification

K-Nearest Neighbor classifier



Requires 3 things:

- The set of stored samples
- A Distance metric to compute distance between two samples
- The value of k, the number of nearest neighbors to retrieve.

To classify an unknown record:

- Compute distance to other training samples.
- Identify its k nearest neighbors
- Use class labels of nearest neighbors to determine the class label of the unknown sample. (e.g., by taking majority vote).

Image Classification

K-Nearest Neighbor classifier

- Compute distance between two data points:
  - Given $\vec{p}$, $\vec{q}$ be two n dimensional data points,
  - Euclidean distance between the two is defined as

$$d(\mathbf{p}, \mathbf{q}) = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2}$$

- Determine the class from the nearest neighbor list
  - take the majority vote of class labels among the k nearest neighbors
  - Weight the vote according to distance, $w = \frac{1}{d^2}$

Image Classification

K-Nearest Neighbor classifier

Consider the following 10 points in the 2-D space and their corresponding class labels. Predict the class label for point (4,5), using k-nearest neighbors, $k \in \{1, 3, 5, 9\}$

x	y	class	d	1NN	3NN	5NN	9NN
1	5	B	3.00		B	B	B
1	6	B	3.16		B	B	B
2	1	C	4.47				C
3	9	C	4.12				C
4	0	C	5.00				C
5	4	A	1.41	A	A	A	A
6	8	B	3.61			B	B
7	7	C	3.61			C	C
8	2	C	5.00				
0	5	C	4.00				C
4	5	?		A	B	B	C

Image Classification

K-Nearest Neighbor classifier (apply to images)

L1 distance: $$d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$$

test image			
56	32	10	18
90	23	128	133
24	26	178	200
2	0	255	220

\-

training image			
10	20	24	17
8	10	89	100
12	16	178	170
4	32	233	112

=

pixel-wise absolute value differences			
46	12	14	1
82	13	39	33
12	10	0	30
2	32	22	108

add → 456

Image Classification

K-Nearest Neighbor classifier (apply to images)

Example Dataset: **CIFAR10**

10 classes
50,000 training images
10,000 testing images

Test images and nearest neighbors

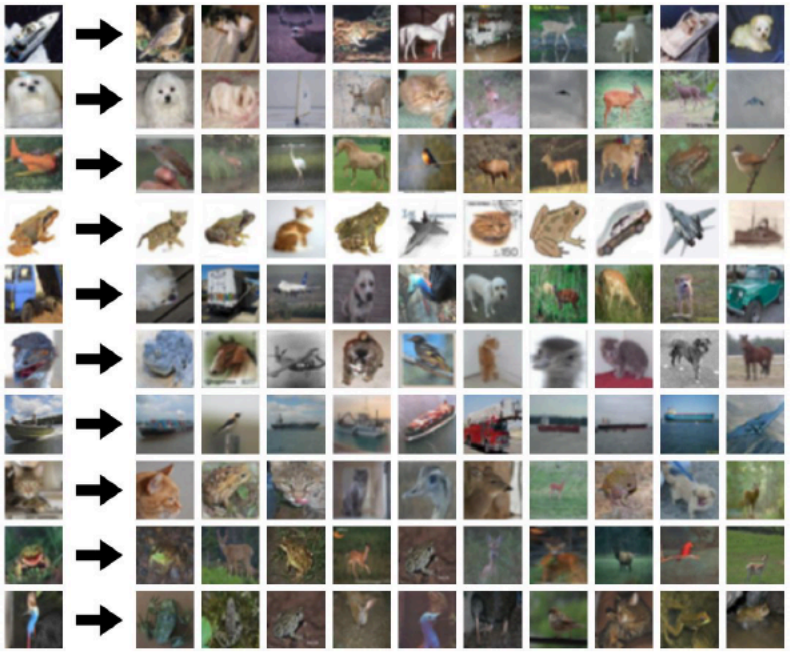


Image Classification

K-Nearest Neighbor classifier (apply to images)

Image Classification

K-Nearest Neighbor classifier

- Choosing the value of k:
  - If k is too small, sensitive to noise points
  - If k is too large, neighborhood may include points from other classes

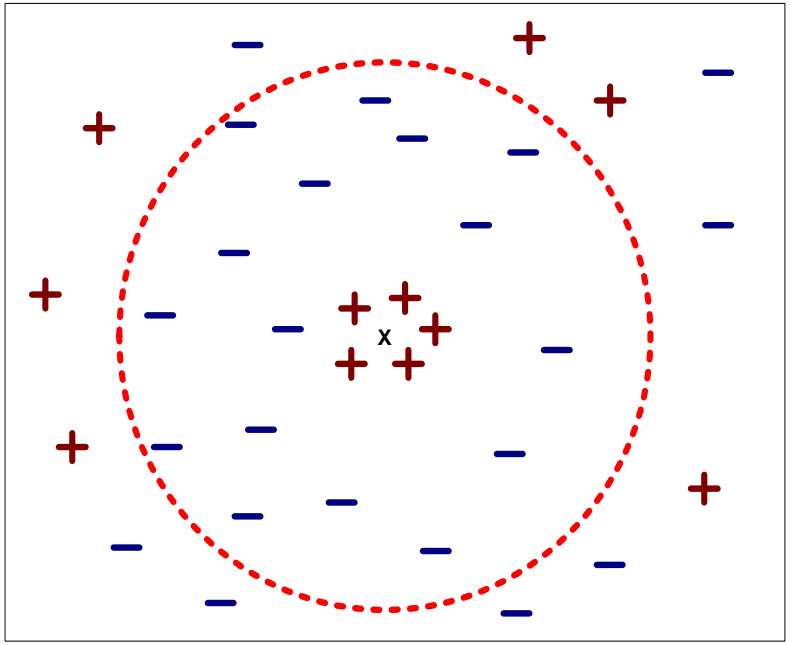


Image Classification

K-Nearest Neighbor classifier

Hyperparameters

What is the best value of **k** to use?
What is the best **distance** to use?

These are **hyperparameters**: choices about the algorithm that we set rather than learn

Very problem-dependent.
Must try them all out and see what works best.

Image Classification

K- Nearest Neighbor Classifier

Hyperparameters

What is the best value of **k** to use?
What is the best **distance** to use?

These are **hyperparameters**: choices about the algorithm that we set rather than learn

Very problem-dependent.
Must try them all out and see what works best.

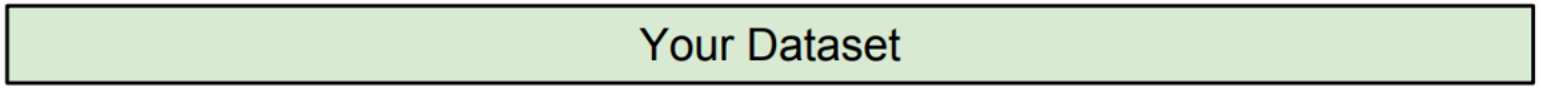
**This slide is from CS231n Stanford University*

Image Classification

K- Nearest Neighbor Classifier

Setting Hyperparameters

Idea #1: Choose hyperparameters that work best on the data



Idea #2: Split data into **train** and **test**, choose hyperparameters that work best on test data

Idea #3: Split data into **train**, **val**, and **test**; choose hyperparameters on val and evaluate on test

Better!

**This slide is from CS231n Stanford University*

Image Classification

K- Nearest Neighbor Classifier

Setting Hyperparameters

Your Dataset

Idea #4: **Cross-Validation**: Split data into **folds**,
try each fold as validation and average the results

fold 1	fold 2	fold 3	fold 4	fold 5	test
fold 1	fold 2	fold 3	fold 4	fold 5	test
fold 1	fold 2	fold 3	fold 4	fold 5	test

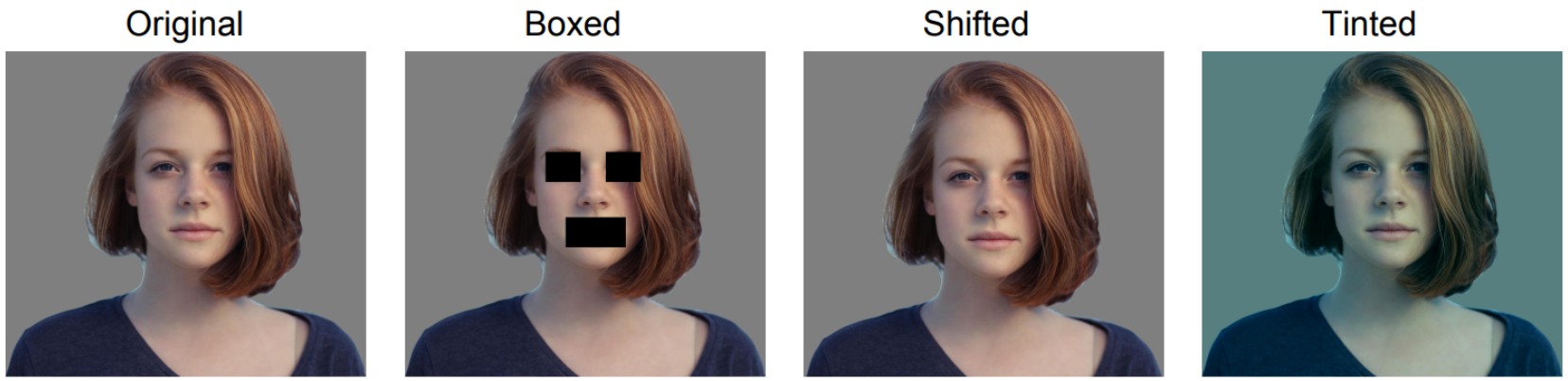
**This slide is from CS231n Stanford University*

Image Classification

K- Nearest Neighbor Classifier

k-Nearest Neighbor on images **never used.**

- Very slow at test time
- Distance metrics on pixels are not informative



(all 3 images have same L2 distance to the one on the left)

**This slide is from CS231n Stanford University*

Image Classification

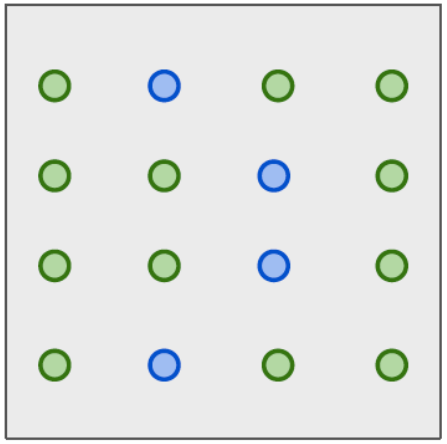
K- Nearest Neighbor Classifier

k-Nearest Neighbor on images **never used.**

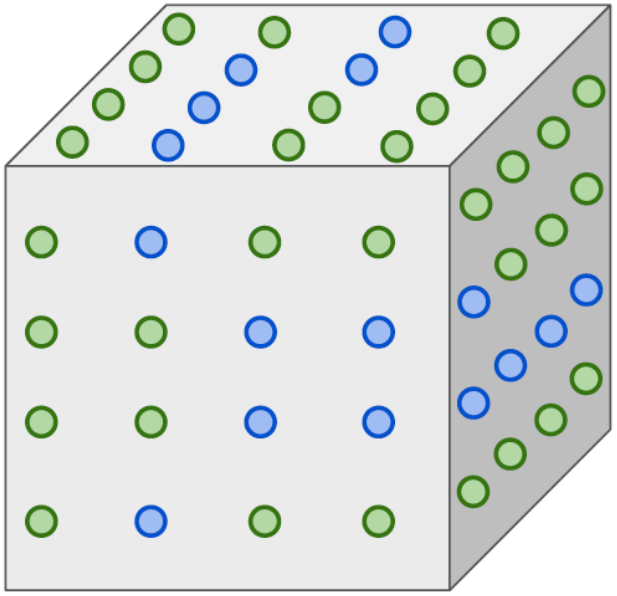
- Curse of dimensionality

Dimensions = 1
Points = 4

Dimensions = 2
Points = 4^2



Dimensions = 3
Points = 4^3



**This slide is from CS231n Stanford University*

Image Classification

K- Nearest Neighbor Classifier

- One main advantage of the k-NN algorithm is that it's **extremely simple** to implement and understand.
- Furthermore, the classifier takes absolutely **no time to train**, since all we need to do is store our data points for the purpose of later computing distances to them and obtaining our final classification.
- However, we pay for this simplicity at classification time. Classifying a new testing point requires a comparison to every single data point in our training data, which scales O(N), making working with larger datasets computationally prohibitive.

Image Classification

K- Nearest Neighbor Classifier

K-Nearest Neighbors: Summary

In **Image classification** we start with a **training set** of images and labels, and must predict labels on the **test set**

The **K-Nearest Neighbors** classifier predicts labels based on nearest training examples

Distance metric and K are **hyperparameters**

Choose hyperparameters using the **validation set**; only run on the test set once at the very end!

**This slide is from CS231n Stanford University*

Image Classification

K- Nearest Neighbor Classifier

HW : 3-class animals classification using K-nearest neighbor classifier

Image Classification

K- Nearest Neighbor Classifier

HW : 3-class animals classification using K-nearest neighbor classifier

1. **Gather the Dataset**: The Animals datasets consists of 3,000 images with 1,000 images per dog, cat, and panda class, respectively. Each image is represented in the RGB color space. You will preprocess each image by resizing it to 32x32 pixels. Taking into account the three RGB channels, the resized image dimensions imply that each image in the dataset is represented by 32x32x3 = 3,072 integers.

Image Classification

K- Nearest Neighbor Classifier

HW : 3-class animals classification using K-nearest neighbor classifier

2. Split the Dataset: You'll be using three splits of the data. One split for training, one split for validation and the other for testing. Please **randomly** partition the data into these three splits. For example, 70% for training, 10% for validation and 20% for testing.

Image Classification

K- Nearest Neighbor Classifier

HW : 3-class animals classification using K-nearest neighbor classifier

3. Train the Classifier: Your k-NN classifier will be trained on the raw pixel intensities of the images in the training set. You need to convert the images to data vectors with label (how?). What is the best value of K to use? What is the best distance to use?

Image Classification

K- Nearest Neighbor Classifier

HW : 3-class animals classification using K-nearest neighbor classifier

4. Evaluate: Once your k-NN classifier is trained, you should evaluate performance (accuracy, precision, recall, F-measure) on the test set. You need to provide the details of your evaluation steps.

Image Classification

K- Nearest Neighbor Classifier

HW: 3-class animals classification using K-nearest neighbor classifier

Submission:

- Write a report to describe the detailed steps of your design and include your evaluation results.
- Upload your code with comments as a separate .zip file

Image Classification

K- Nearest Neighbor Classifier

Example:

https://www.datacamp.com/community/tutorials/k-nearest-neighbor-classification-scikit-learn

```
# Assigning features and label variables
# First Feature

weather=['Sunny','Sunny','Overcast','Rainy','Rainy','Rainy','Overcast','Sunny','Sunny',
'Rainy','Sunny','Overcast','Overcast','Rainy'] # Second Feature
temp=['Hot','Hot','Hot','Mild','Cool','Cool','Cool','Mild','Cool','Mild','Mild','Mild','Hot'
,'Mild'] # Label or target varible
play=['No','No','Yes','Yes','Yes','No','Yes','No','Yes','Yes','Yes','Yes','Yes','No']
```

Image Classification

K- Nearest Neighbor Classifier

Example:

https://www.datacamp.com/community/tutorials/k-nearest-neighbor-classification-scikit-learn

```
#Import LabelEncoder from sklearn import preprocessing
#creating labelEncoder
le = preprocessing.LabelEncoder()

# Converting string labels into numbers.
weather_encoded=le.fit_transform(weather)
print(weather_encoded)
```

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

Image Classification

K- Nearest Neighbor Classifier

Example:

https://www.datacamp.com/community/tutorials/k-nearest-neighbor-classification-scikit-learn

```
# converting string labels into numbers
temp_encoded=le.fit_transform(temp)
label=le.fit_transform(play)
```

```
#combinig weather and temp into single list of tuples
features=list(zip(weather_encoded,temp_encoded))
```

Image Classification

K- Nearest Neighbor Classifier

Example:

https://www.datacamp.com/community/tutorials/k-nearest-neighbor-classification-scikit-learn

```
from sklearn.neighbors import KNeighborsClassifier
model = KNeighborsClassifier(n_neighbors=3)

# Train the model using the training sets
model.fit(features,label)

#Predict Output
predicted= model.predict([[0,2]]) # 0:Overcast, 2:Mild
print(predicted)
```

```
>> [1]
```

Image Classification

K- Nearest Neighbor Classifier

HW:

Pip3 install opencv-python
Import cv2
https://pypi.org/project/opencv-python/

```
Import os
Import cv2

Dataset_path = "../Data/"

image_list = os.listdir(Dataset_path)
      for image_name in image_list:
         image = cv2.imread(Dataset_path+image_name)
         image = cv2.resize(image, (32, 32),interpolation=cv2.INTER_CUBIC)
```

Image Classification

K- Nearest Neighbor Classifier

HW:

Pip3 install opencv-python
Import cv2
https://pypi.org/project/opencv-python/

```
Import os
Import cv2

Dataset_path = "../Data/"
 data=[]
 #labels=[]

image_list = os.listdir(Dataset_path)
      for image_name in image_list:
            …
            # read image append as list
            data.append(image)

mydata=np.array(data)
```

Image Classification

K- Nearest Neighbor Classifier

HW:

Pip3 install opencv-python
Import cv2
https://pypi.org/project/opencv-python/

```
Import os
Import cv2

Dataset_path = "../Data/KNN/animals/"

class_folders = os.listdir(Dataset_path)
 for class_name in class_folders:
      image_list = os.listdir(Dataset_path+class_name)
      for image_name in image_list:

      print("Folder Done")
```

Image Classification

K- Nearest Neighbor Classifier

HW-V1:

```
def load(Dataset_path):
    return (np.array(data),np.array(labels))

def KNN_pred(trainX, trainY, testX, k)
     return Ypred

#if your data is (3000, 32,32,3) you need data.reshape((data.shape[0],3072))

Le=LabelEncode()
labels=le.fit_transform(labels)

(trainX, testX,
trainY,testY)=Train_test_split(data,labels,test_size=0.3,random_state=number)

print(classification_report(testY, Ypred, target_names=le.classes_))
```

Image Classification

K- Nearest Neighbor Classifier

HW_V2:

```
def load(Dataset_path):
    return (np.array(data),np.array(labels))

#if your data is (3000, 32,32,3) you need data.reshape((data.shape[0],3072))
Le=LabelEncode()
labels=le.fit_transform(labels)

(trainX, testX,
trainY,testY)=Train_test_split(data,labels,test_size=0.3,random_state=number)

print(classification_report(testY, Ypred, target_names=le.classes_))

model = KNeighborsClassifier(n_neighbors=number)
model.fit(data,labels)
Ypred=model.predict(testX)
```

Image Classification

K- Nearest Neighbor Classifier

HW:

for a selected K, L#:

```
X_train, X_tv, y_train, y_tv = train_test_split(X, y, test_size=0.30, random_state=42)

X_test, X_valid, y_test, y_valid = train_test_split(X_tv, y_tv, test_size=0.33, random_state=42)

print ('Train Test Validate Split Done')
```

Image Classification

K- Nearest Neighbor Classifier

HW:

for k in Krange:

```
    plot_report (k, L1)
    plot_report (k, L2)
```

Analysis :

What K

What L

Image Classification

K- Nearest Neighbor Classifier

HW:

Your own KNN

```
def predict(trainX,trainY,testX,k):

        num_test = testX.shape[0]
      # lets make sure that the output type matches the input type
        Ypred = np.zeros(num_test, dtype=trainY.dtype)


        for i in range(num_test):
                L1_distances = np.sum(np.abs(trainX - testX[i, :]), axis=-1)
            #L2_distances = np.sqrt(np.sum(np.square(trainX - testX[i,:]), axis = 1))
            #min_index = np.argmin(L1_distances)  # get the index with smallest distance
                K_index = np.argsort(L1_distances)[:k]
                #K_labels=np.zeros(len(K_index))
                K_labels=[]
                for j in range(len(K_index)):
                    K_labels.append(trainY[K_index[j]])

                Ypred[i] = max(set(K_labels), key = K_labels.count)

        return Ypred
```