Computer Vision

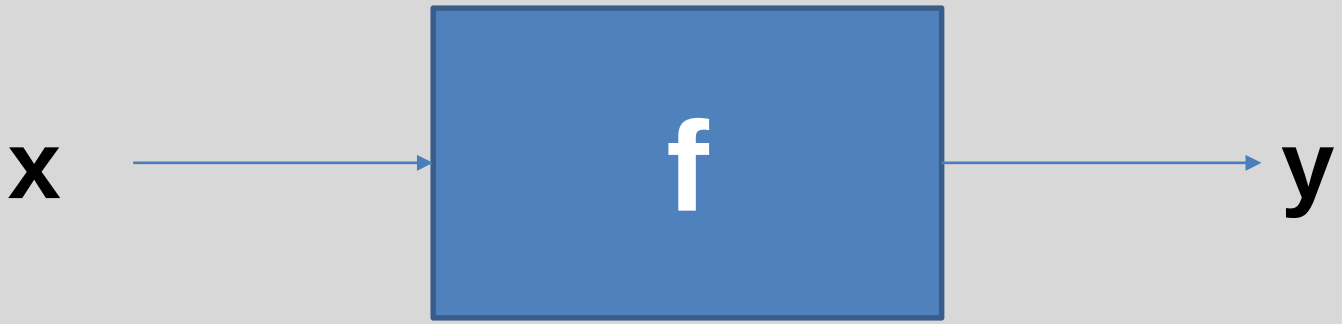
Lecture 02: Machine Learning Basics-1

Machine Learning Basics

- Take 'CSE46302 machine learning' class for more fundamental details.
- The focus of this class (CSE472) is reviewing machine learning algorithms in the context of solving the computer vision applications.

Machine learning

- **x**: data, **y**: ground-truth, **f**: function (ie. machine)



- Find **f** from data=learning a machine **f** from many (**x**, **y**) pairs. ie. machine learning

Machine learning

- **x***: new data, **y***: prediction, **f**: function

- After finding f, we will predict **y*=f**(**x***) for the new sample **x***.

Types of machine learning

- Supervised learning
  - Data **x**, Ground-truth **y**, machine **f**:**x**→**y**.
  - Use (**x**, **y**) pairs to find a proper function **f**.
  - Normally, **f** is parameterized by the **w**.
  - We find **w** that is able to minimize the loss function **L**: normally the difference between **y*=f**(**x**; **w**) and **y.**
  - For example, **L=||y*-y||**.

Types of machine learning

- Un-supervised learning
  - Data **x**, Category **y**, machine **f**:**x**→**y**.
  - Does not use **y** when deciding **f**, relying solely on **x**.
  - Reveal categories **y** of data **x**, by seeing only geometry of **x** space (similarities/dissimilarities).

Types of machine learning

- Clustering
  - Representative tasks for the un-supervised learning.
  - Categorize input **x** according to their geometric structures.

Types of machine learning

- Classification
  - Representative computer vision tasks for supervised learning.
  - In classification, **x**: image, **y**: semantic class.

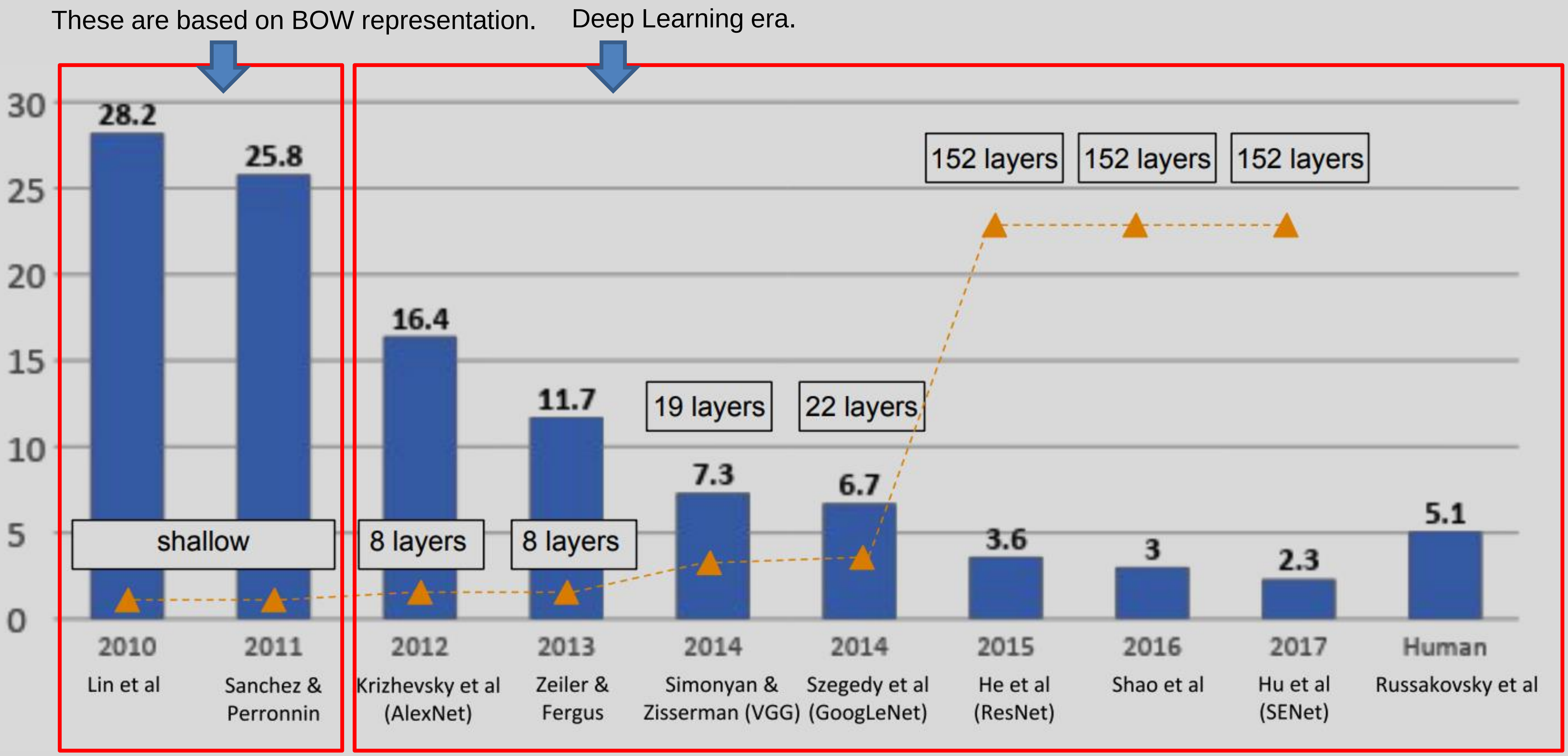
Types of machine learning

- Regression
  - Data **x**, ground-truth **y**.
  - The ground-truth **y** is continuous.
  - It is trained in the supervised way.

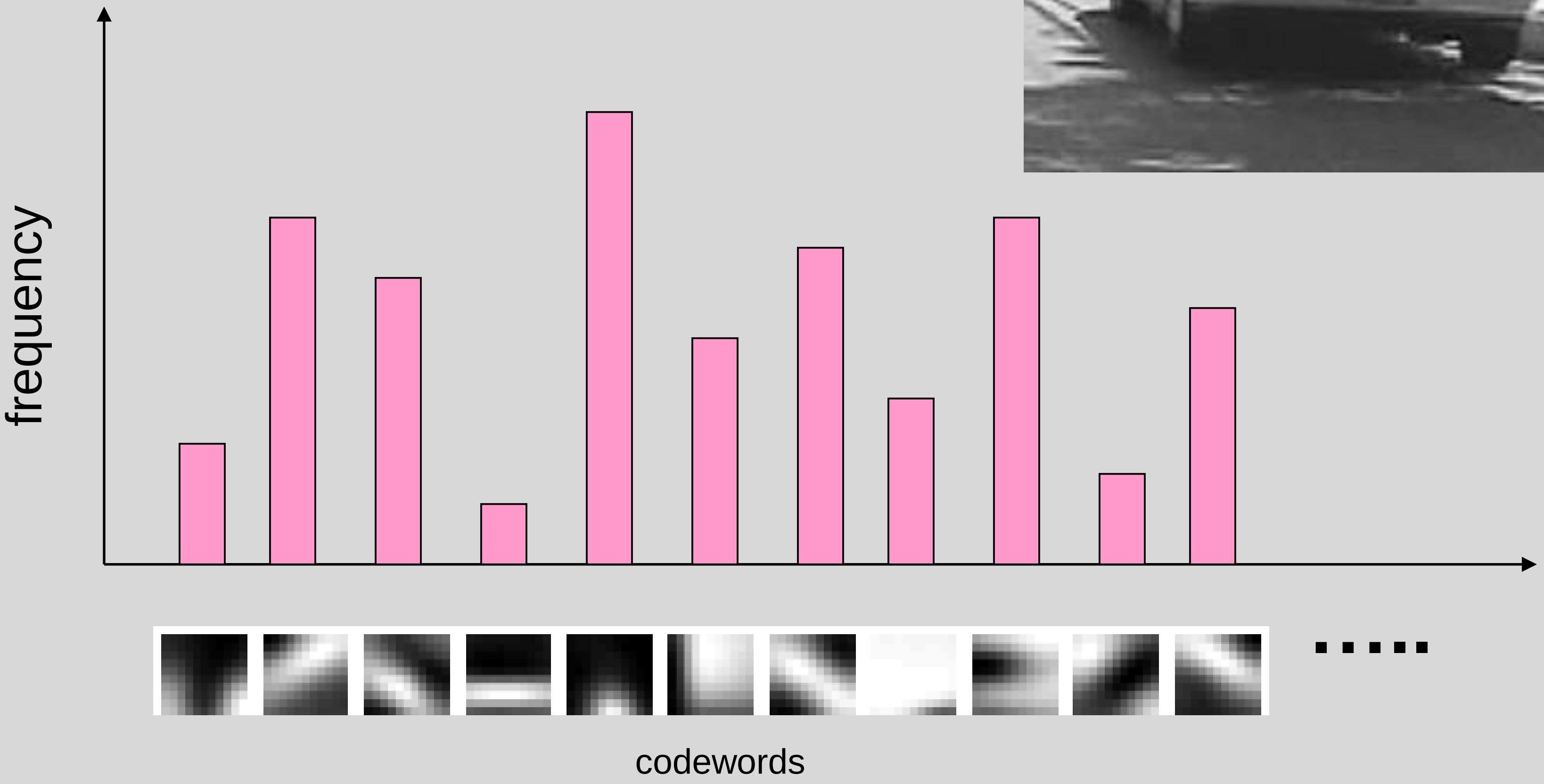
Types of machine learning

- Detection/Segmentation
  - In detection, **x**: image, **y**: bounding box location, class.
  - In segmentation, **x**: image pixels, **y**: class.

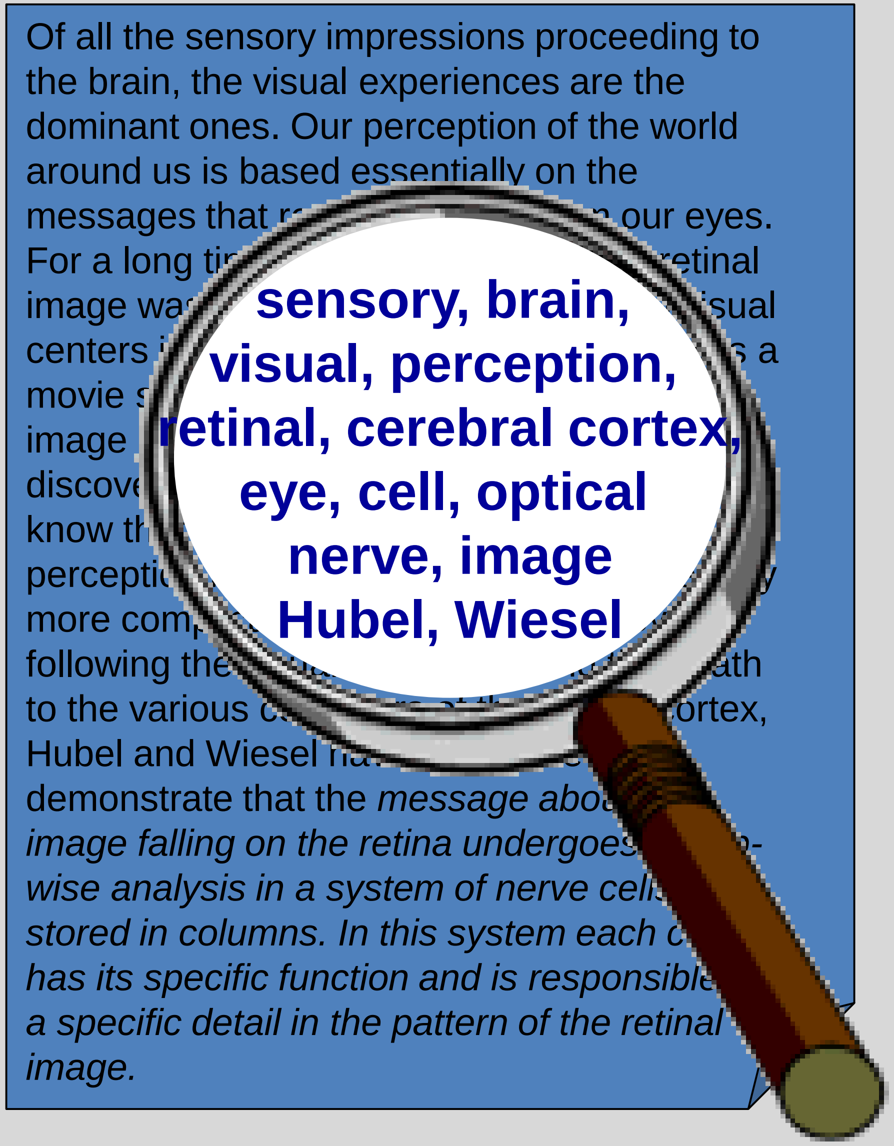
ImageNet challenge result



Bag-of-words (BOW) model



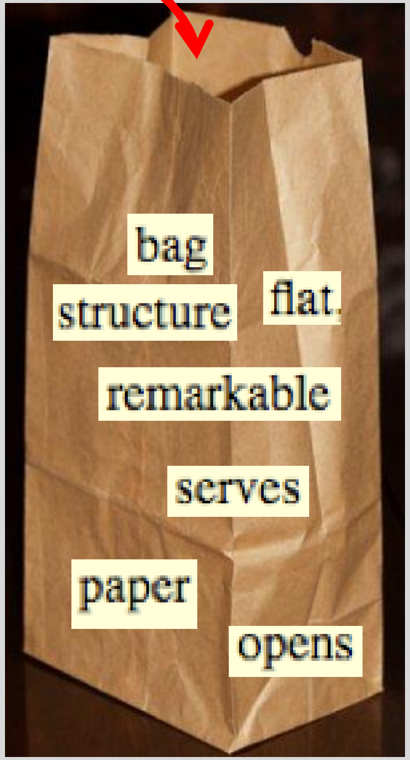
Bag-of-words (BOW) model for texts



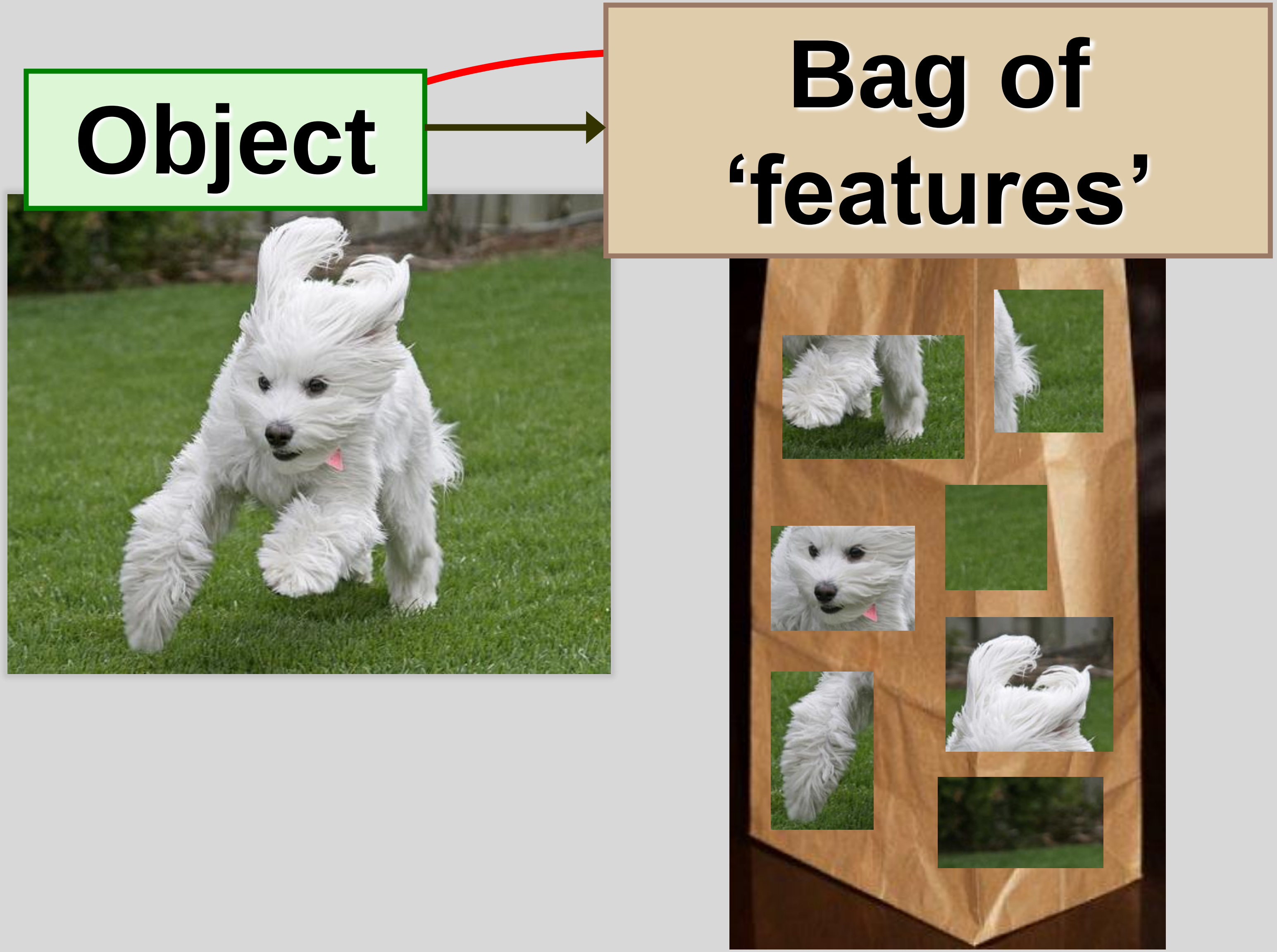
Bag-of-words (BOW) model for texts

The paper bag is a remarkable contrivance. It serves us constantly and inconspicuously. It folds flat, yet opens into a structure that can stand open upon the table while we eat our sandwiches from it and chat with friends.

If we take the bag apart, we find it's made from a single paper cylinder. One end of the cylinder has been folded into a complex 3-dimensional pattern and finished off with a bit of paste. It would be, and once was, costly to make, because each fragile cylinder had to be folded manually into that hardy sack.

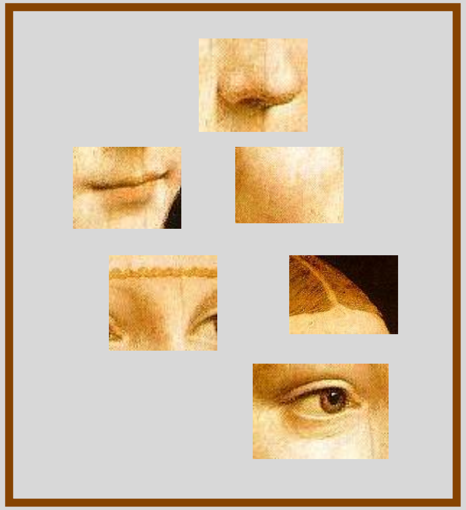


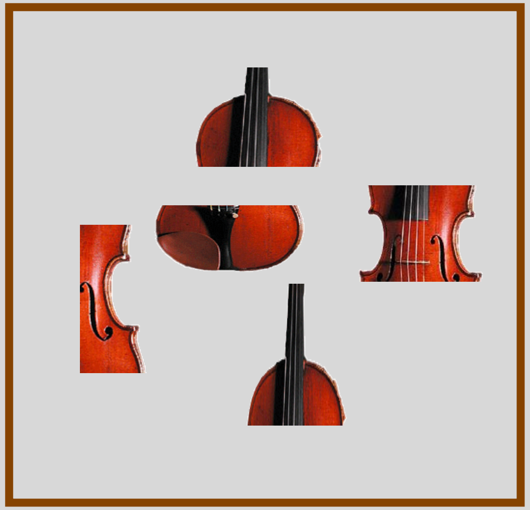
Bag-of-words (BOW) model



Generating Bag-of-words (BOW)

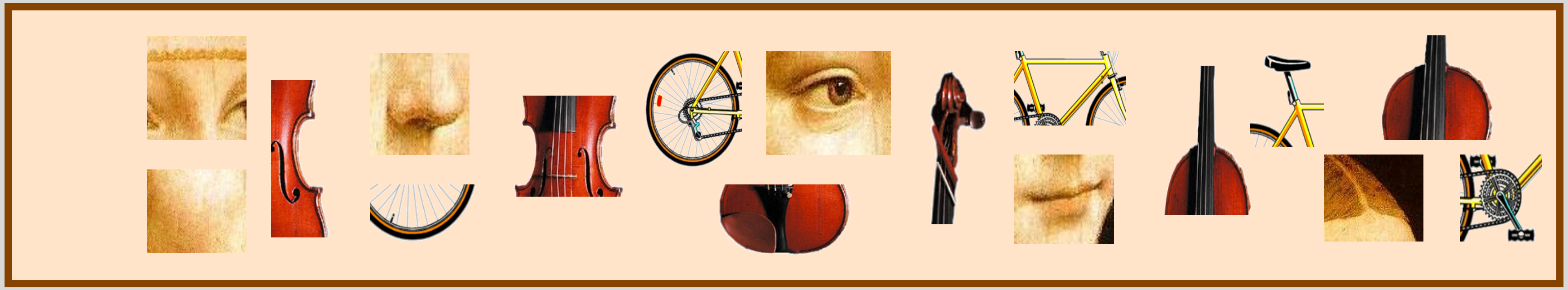
1. Extract features from all training images.





Generating Bag-of-words (BOW)

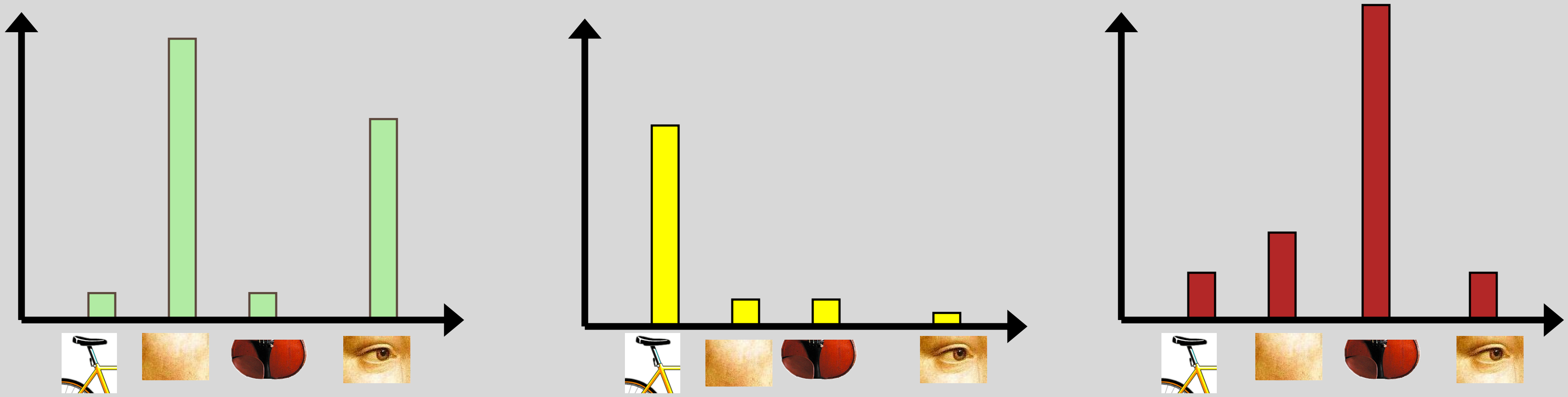
2. Learn a ''visual dictionary''



Generating Bag-of-words (BOW)

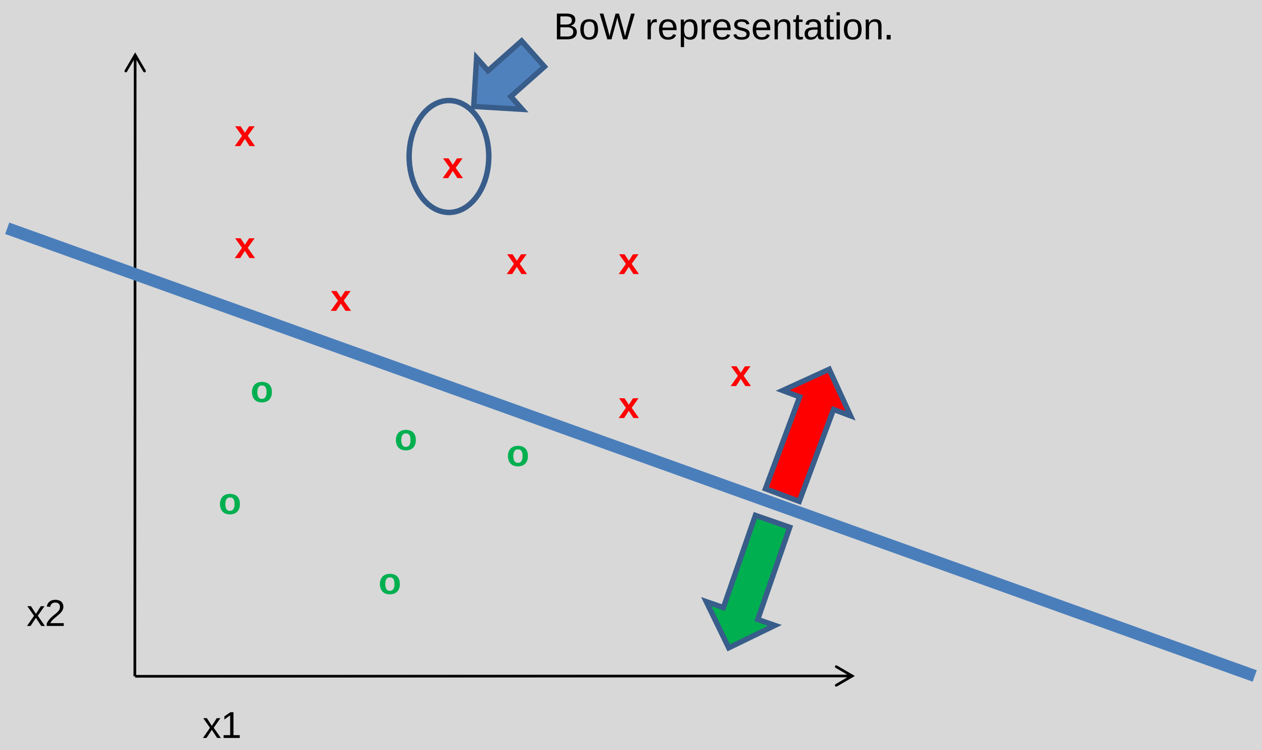
3. Represent images by frequencies of "visual words"

→ Histogram of ''visual words''.

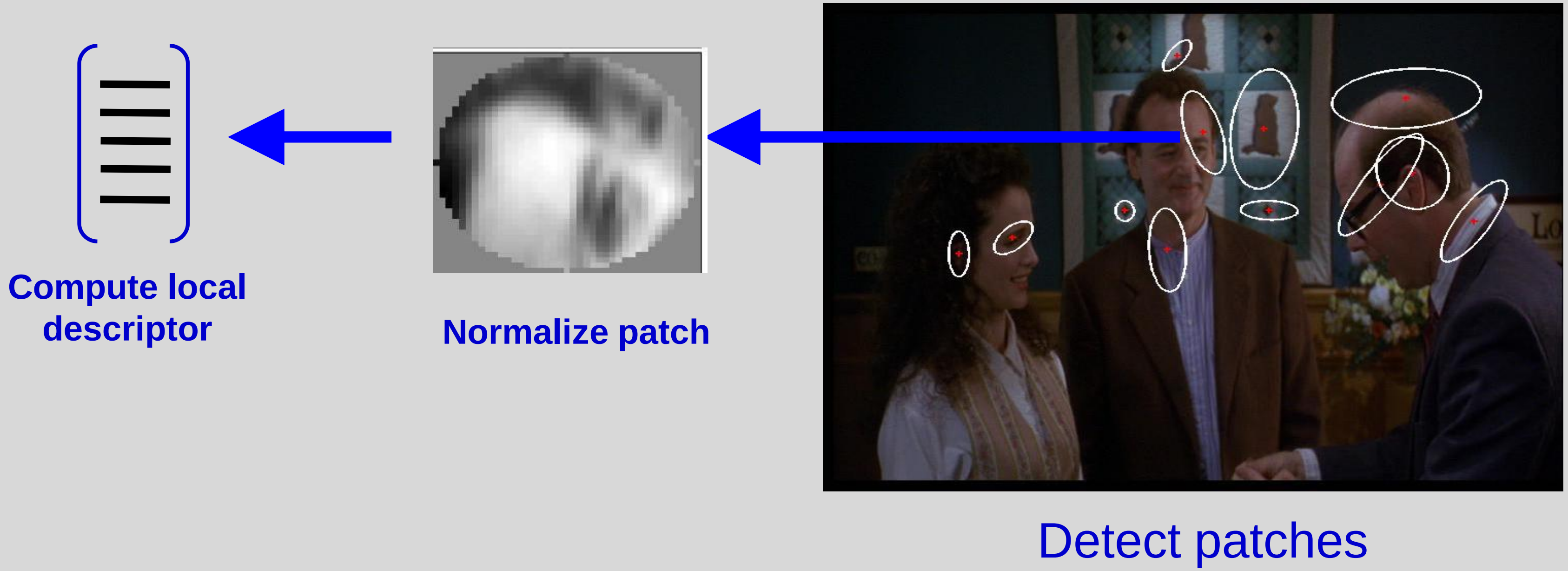


Generating Bag-of-words (BOW)

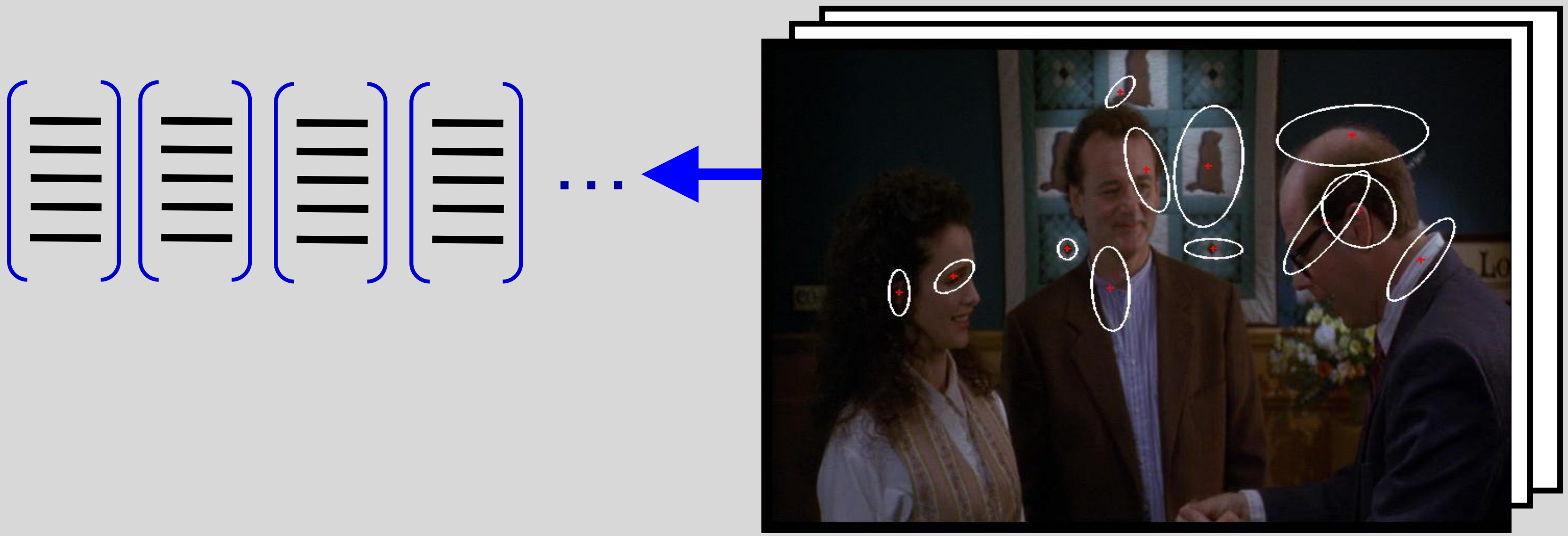
4. Apply machine learning algorithms to discriminate histograms.



Feature extraction

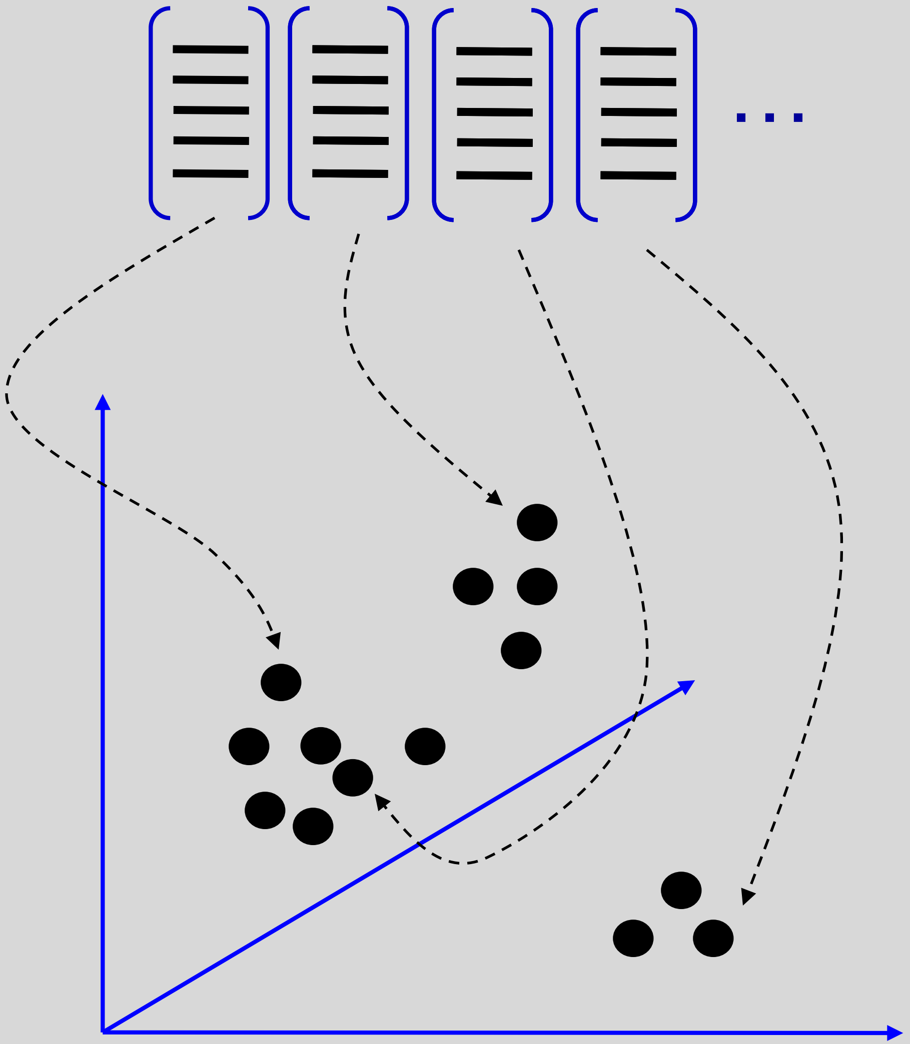


Feature extraction

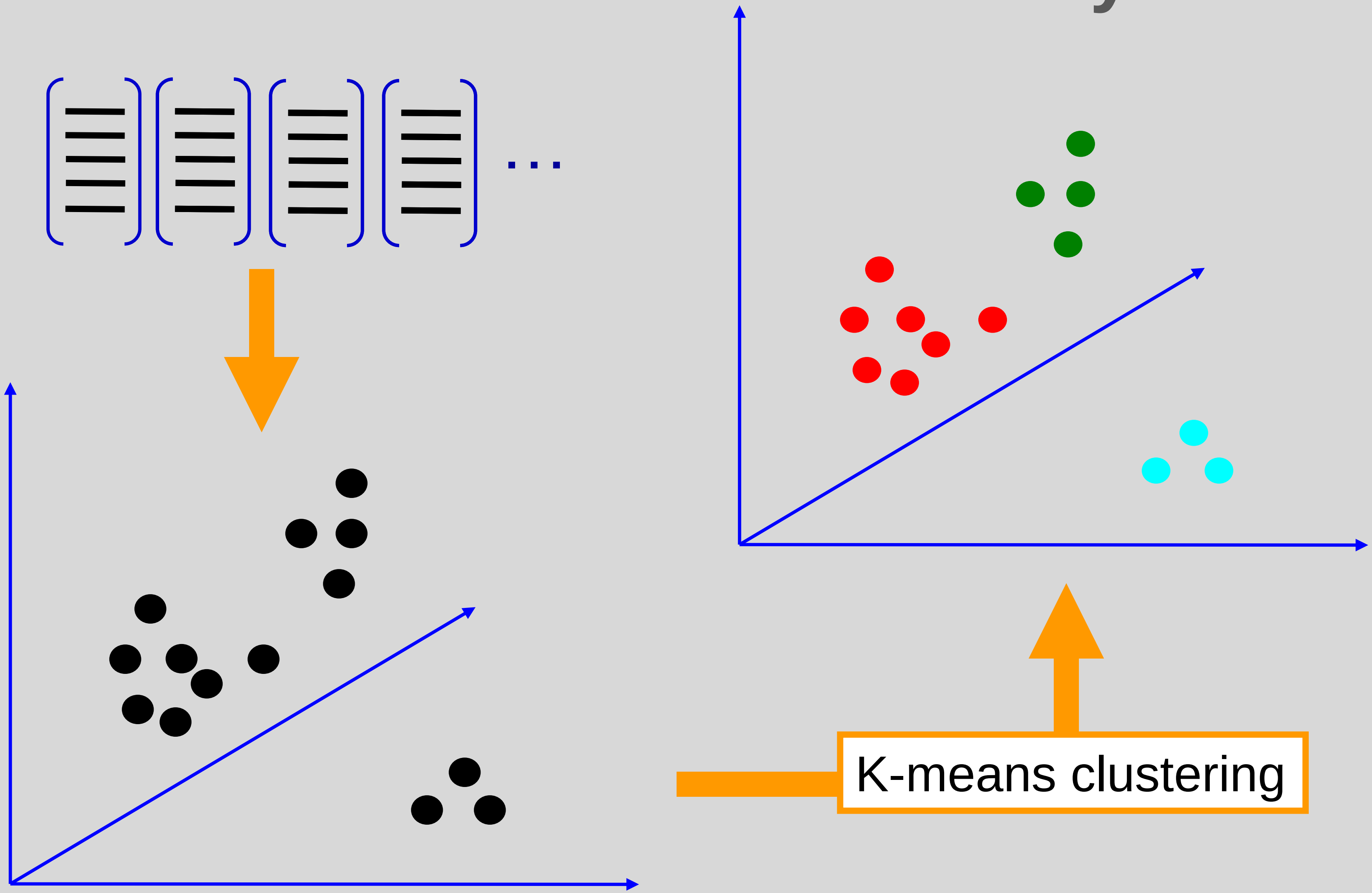


Training images

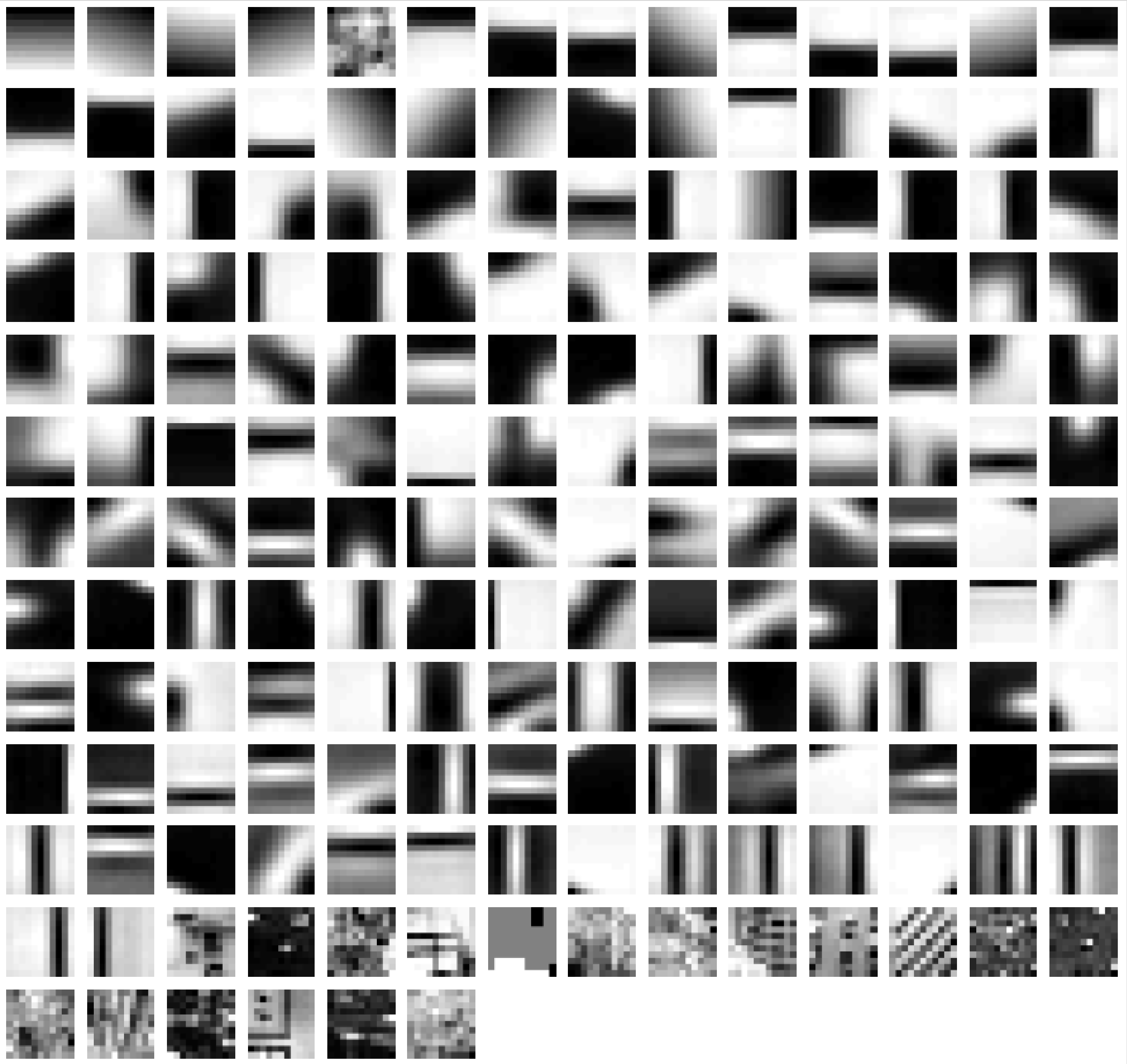
Learn visual dictionary



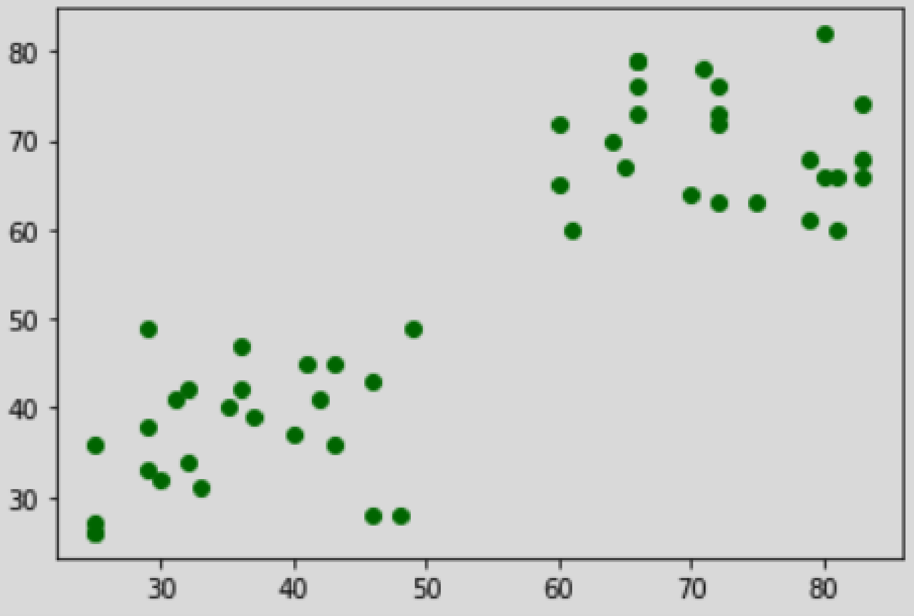
Learn visual dictionary



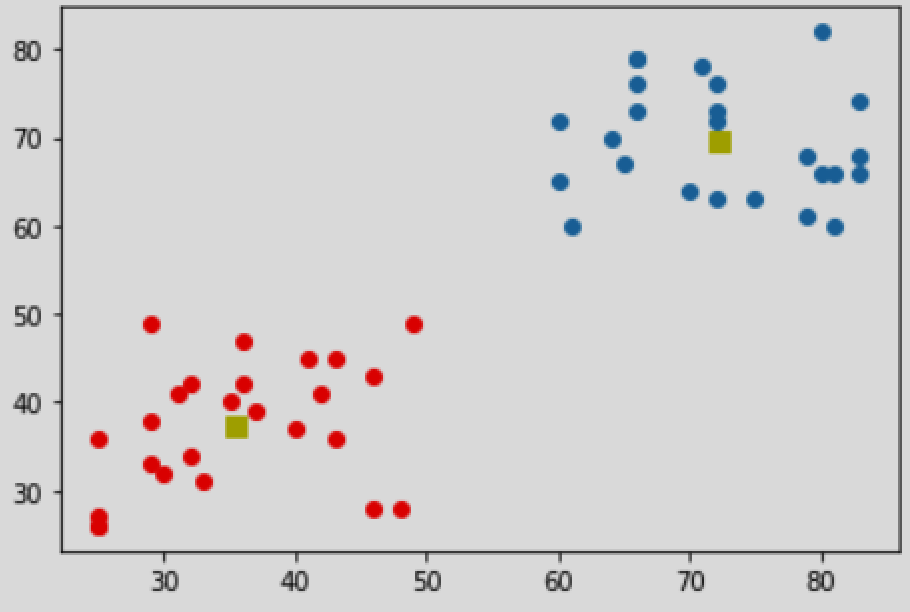
Visual dictionary



K-means Clustering



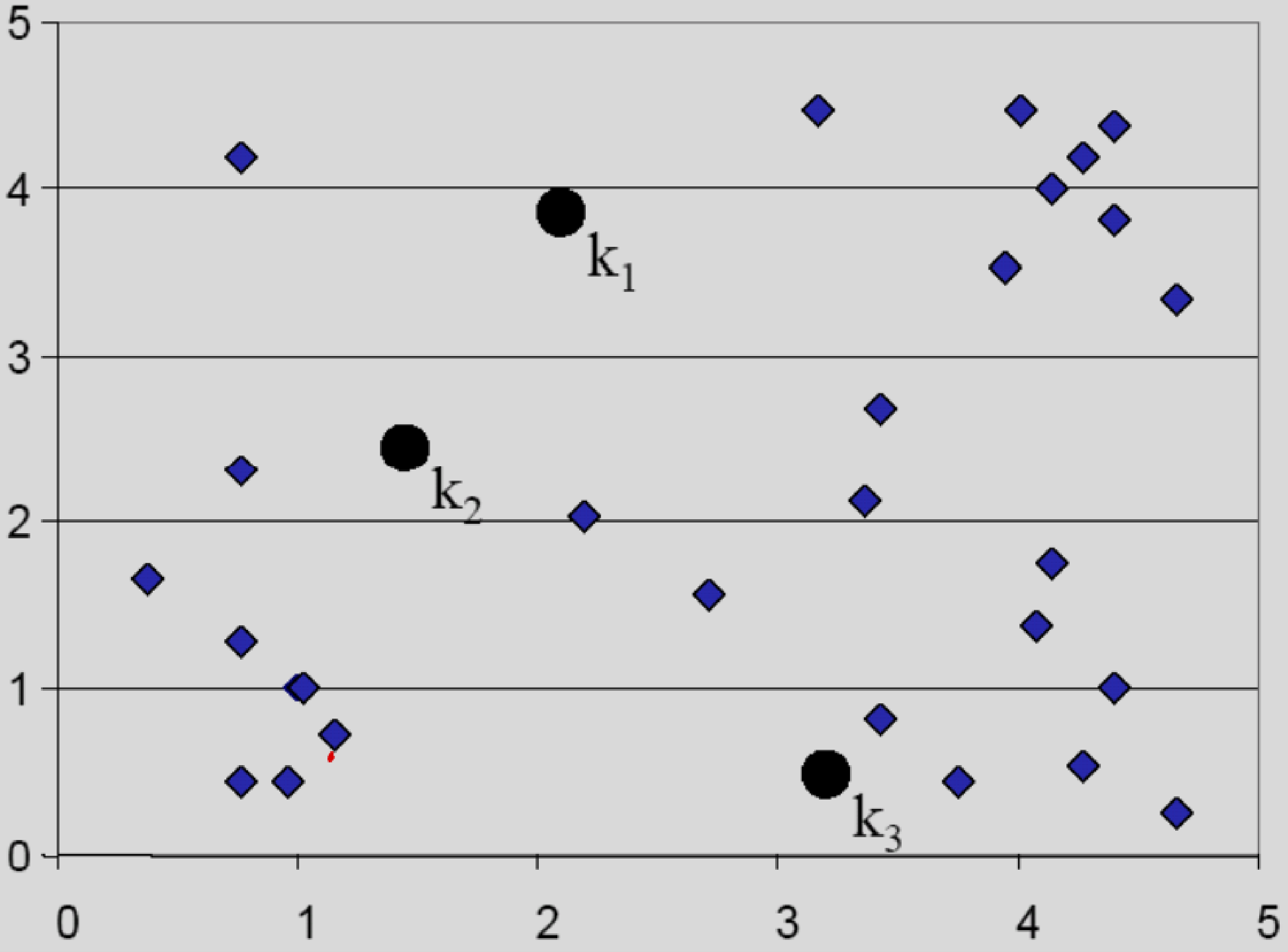
2-dimensional input data



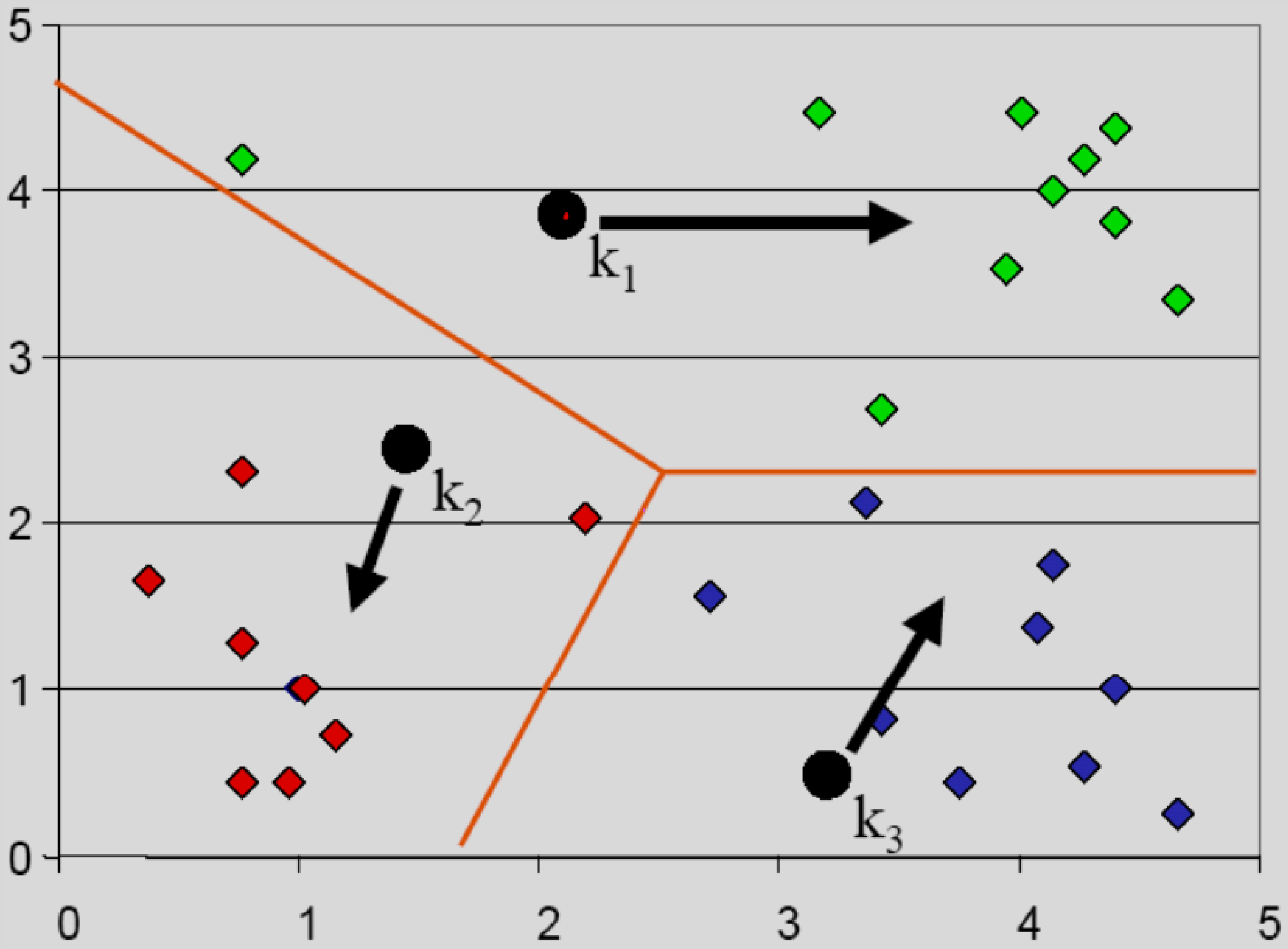
Categorized data

K=2 for this setting, we want to categorize input data into K=2 categories.

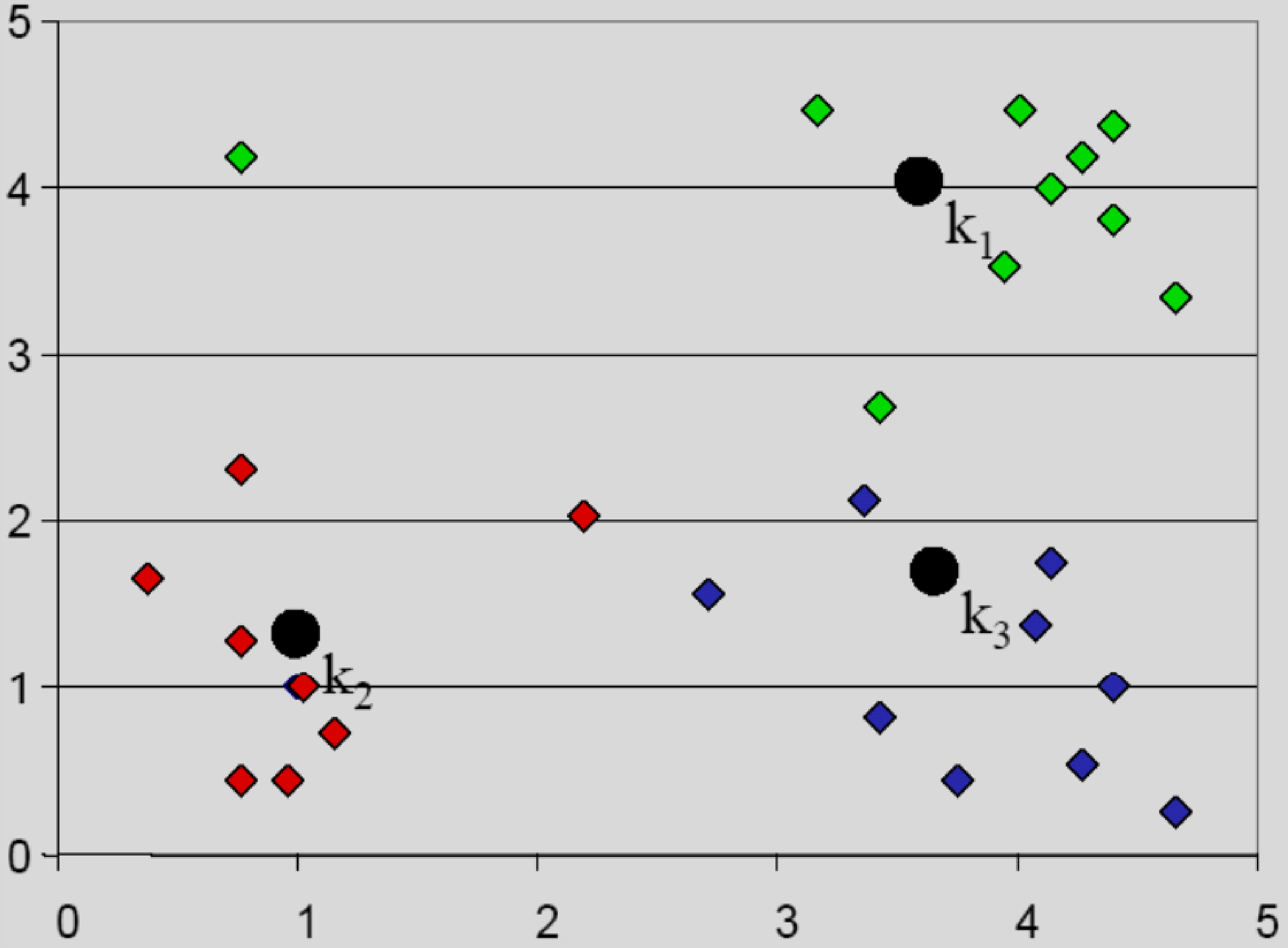
K-means Clustering



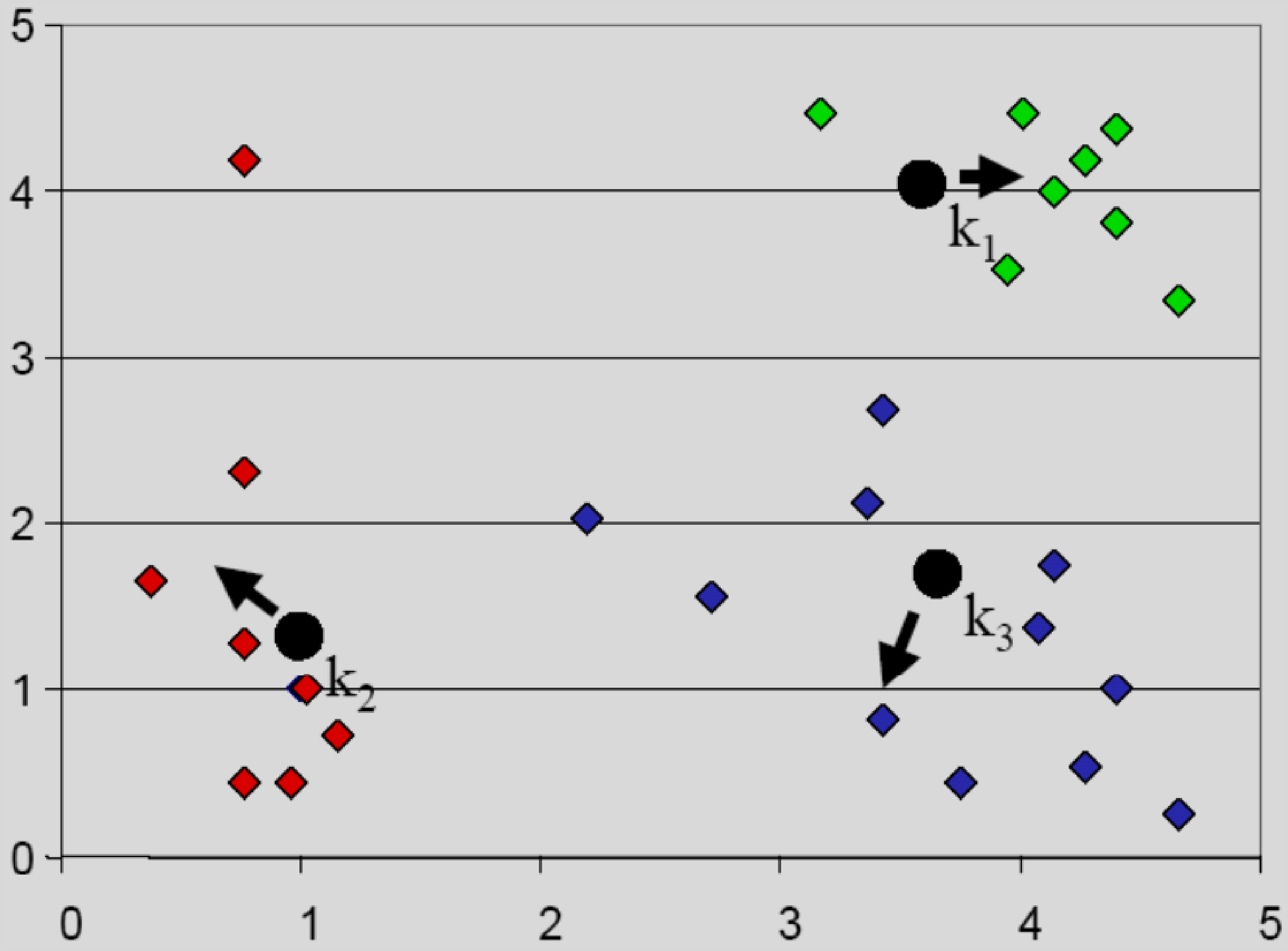
K-means Clustering



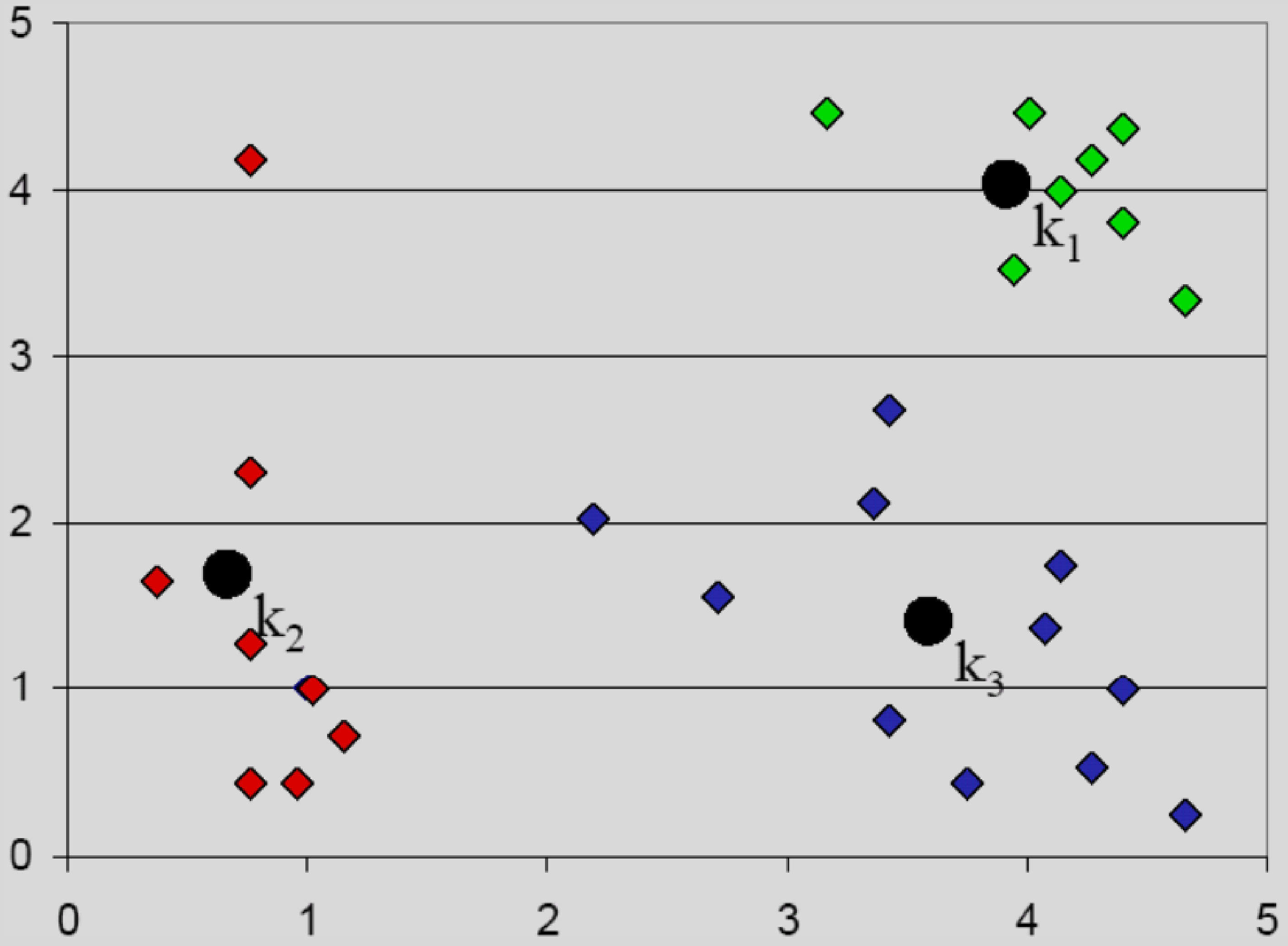
K-means Clustering



K-means Clustering



K-means Clustering



K-means Clustering

Input: cluster size K, instances $\{x_i\}_{i=1}^N$, distance metric $d(\cdot,\cdot)$
Output: cluster membership assignments $\{y_i\}_{i=1}^N$

1. Initialize K cluster centroids $\{c_i\}_{i=1}^k$ (randomly if no domain knowledge available)
2. Repeat 1) and 2) until no instance changes its cluster membership:

1) Decide the cluster membership of instances by assigning them to the nearest cluster centroid

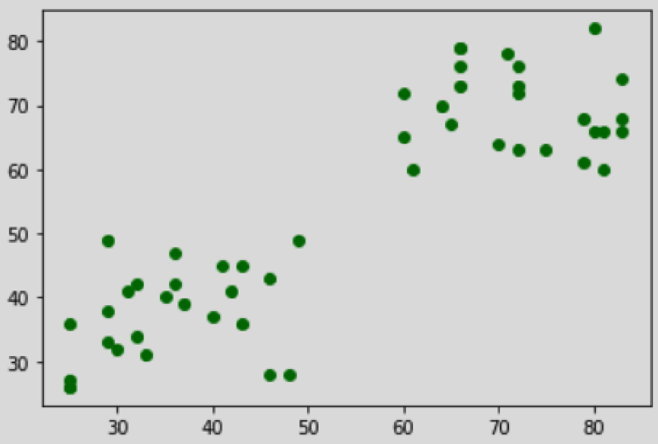
$$y_i = argmin_k d(c_k, x_i)$$

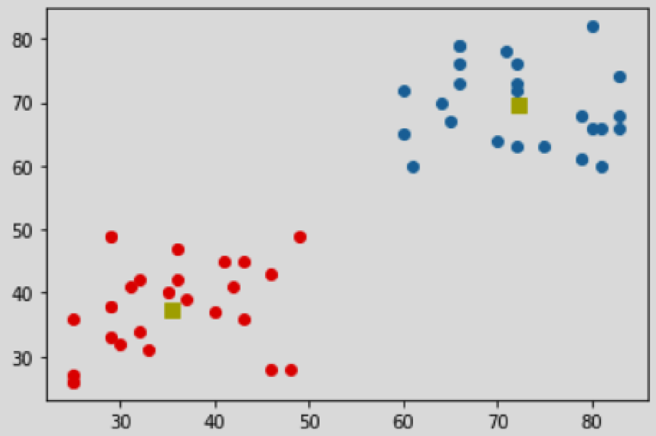
2) Update the K cluster centroids based on the assigned cluster membership

$$c_k = \frac{\sum_i \delta(y_i = k) x_i}{\sum_i \delta(y_i = k)}$$

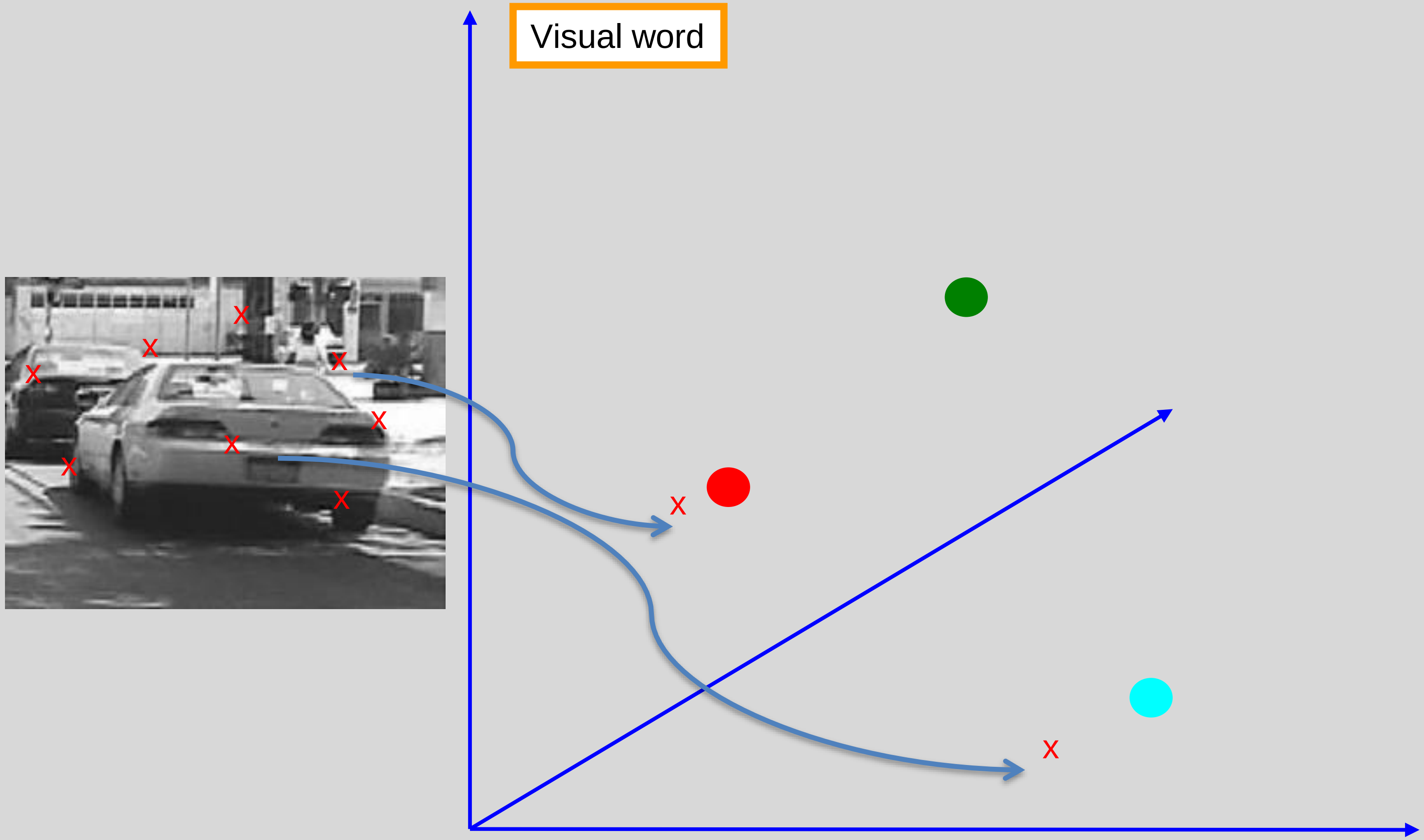
K-means Clustering

```
import cv2
from matplotlib import pyplot as plt
import numpy as np

X = np.random.randint(25,50,(25,2))
Y = np.random.randint(60,85,(25,2))
Z = np.vstack((X,Y))
Z = np.float32(Z)

criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 10, 1.0)
ret,label,center=cv2.kmeans(Z,2,None,criteria,10,cv2.KMEANS_RANDOM_CENTERS)
A = Z[label.ravel()==0]
B = Z[label.ravel()==1]

plt.scatter(Z[:,0],Z[:,1],c = 'g')
plt.show()

plt.scatter(A[:,0],A[:,1],c = ‘b’)
plt.scatter(B[:,0],B[:,1],c = 'r')
plt.scatter(center[:,0],center[:,1],s = 80,c = 'y', marker = 's')
plt.show()
```

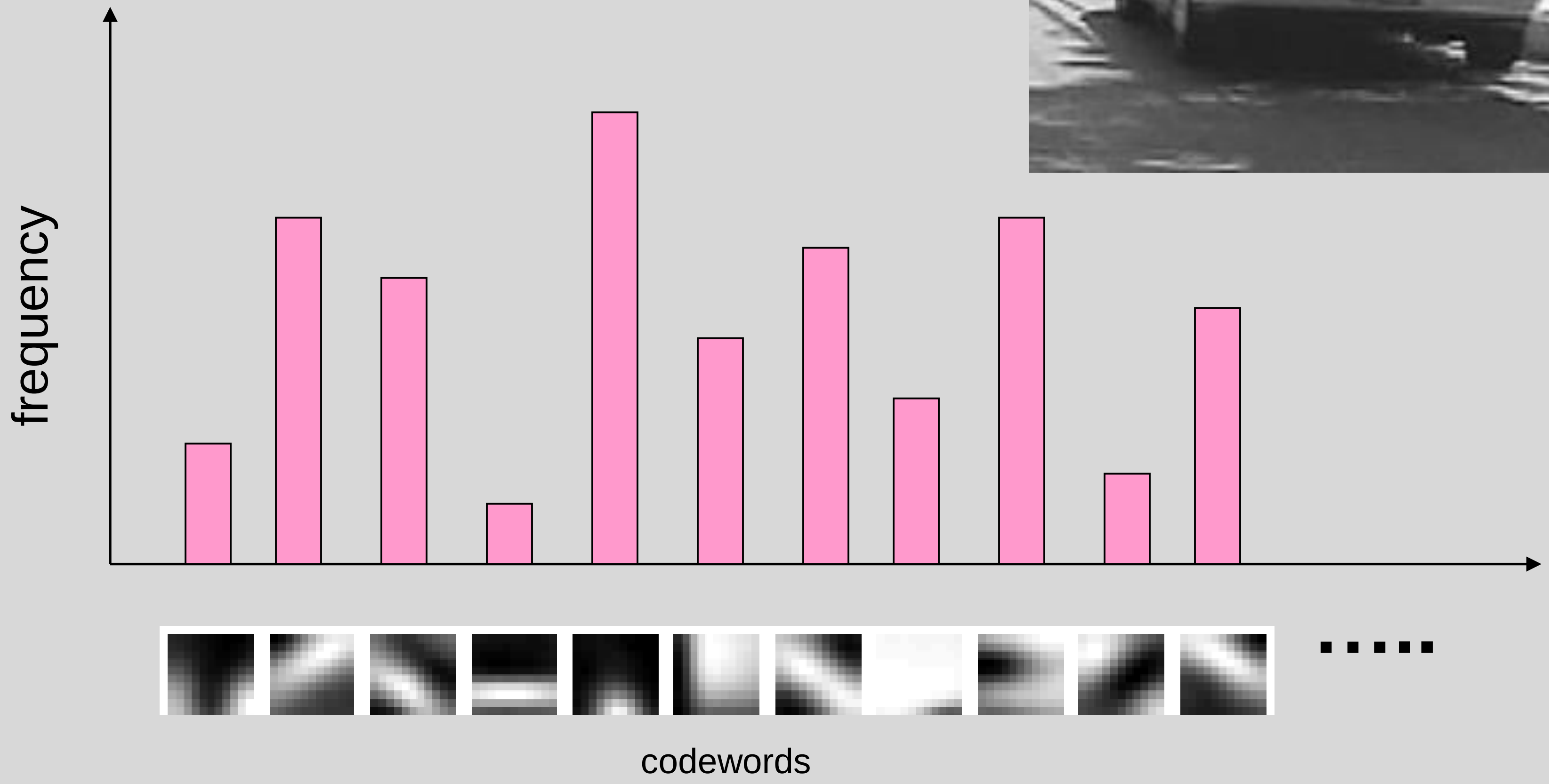




Representing images

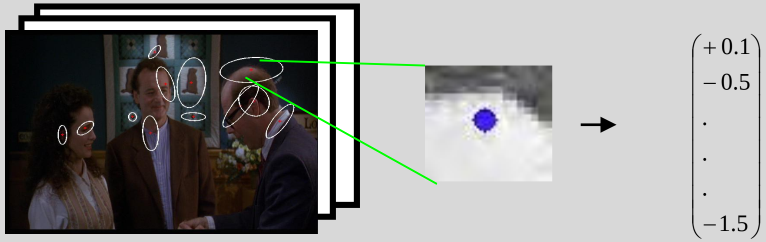


Representing images



Bag of words pipeline

Training stage

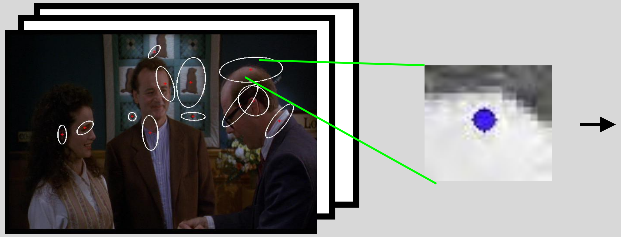


Training images

Extract features

Bag of words pipeline

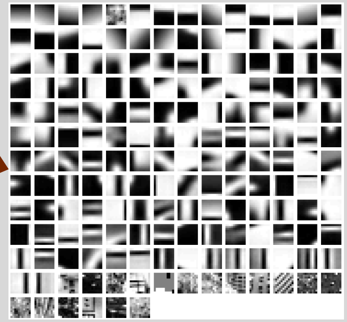
Training stage



$$\begin{pmatrix} +0.1 \\ -0.5 \\ \cdot \\ \cdot \\ \cdot \\ -1.5 \end{pmatrix}$$

Training images

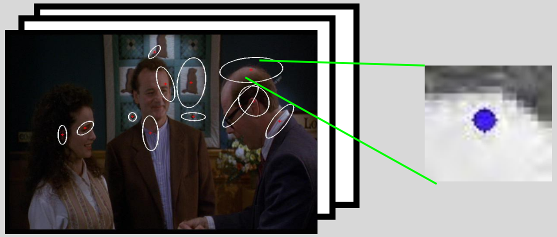
Extract features



Visual dictionary

Bag of words pipeline

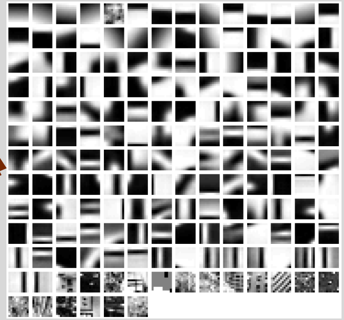
Training stage



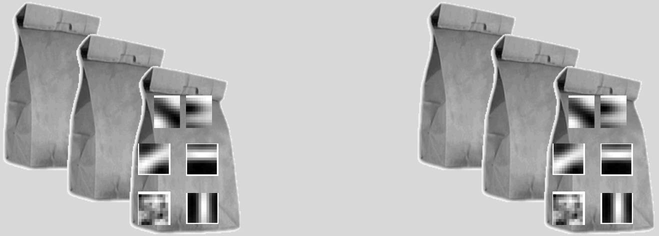
$$\rightarrow \begin{pmatrix} +0.1 \\ -0.5 \\ \cdot \\ \cdot \\ \cdot \\ -1.5 \end{pmatrix}$$

Training images

Extract features



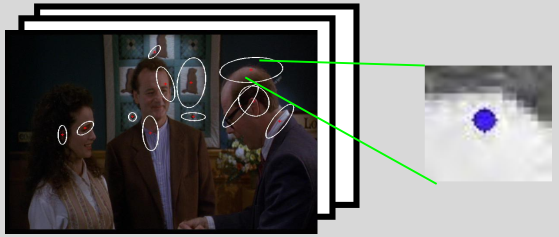
Visual dictionary



Train machine learning classifier

Bag of words pipeline

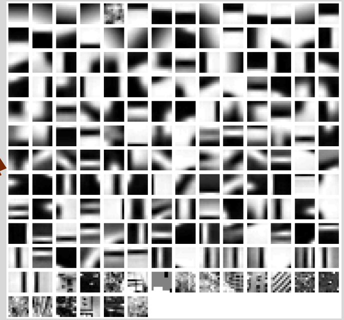
Training stage



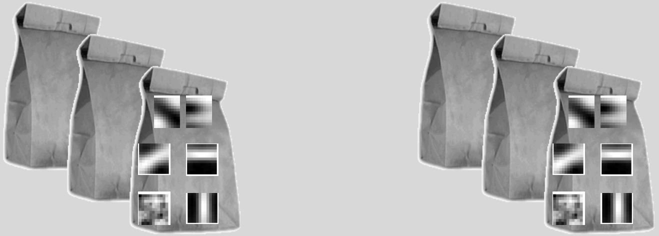
$$\begin{pmatrix} +0.1 \\ -0.5 \\ \cdot \\ \cdot \\ \cdot \\ -1.5 \end{pmatrix}$$

Training images

Extract features



Visual dictionary



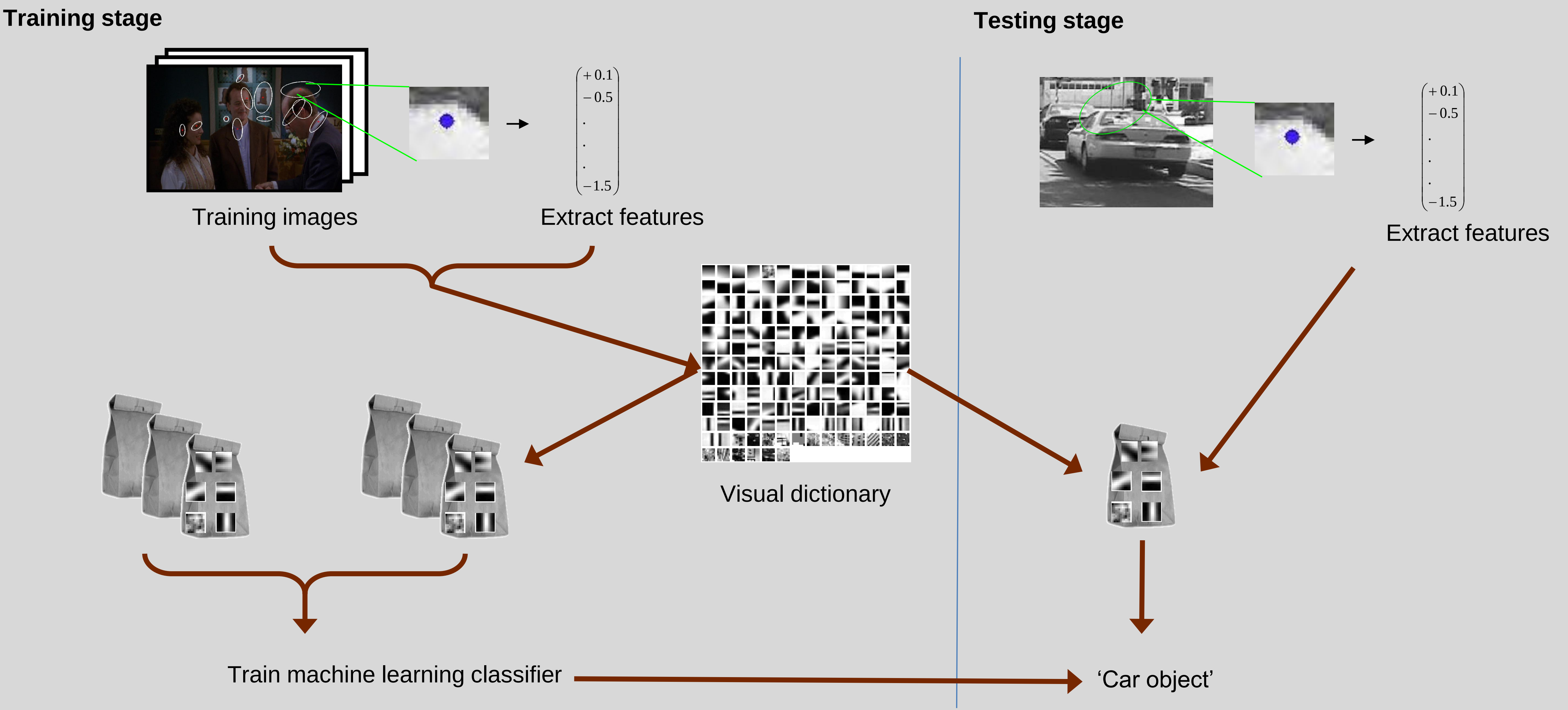
Train machine learning classifier

Testing stage

$$\begin{pmatrix} +0.1 \\ -0.5 \\ \cdot \\ \cdot \\ \cdot \\ -1.5 \end{pmatrix}$$

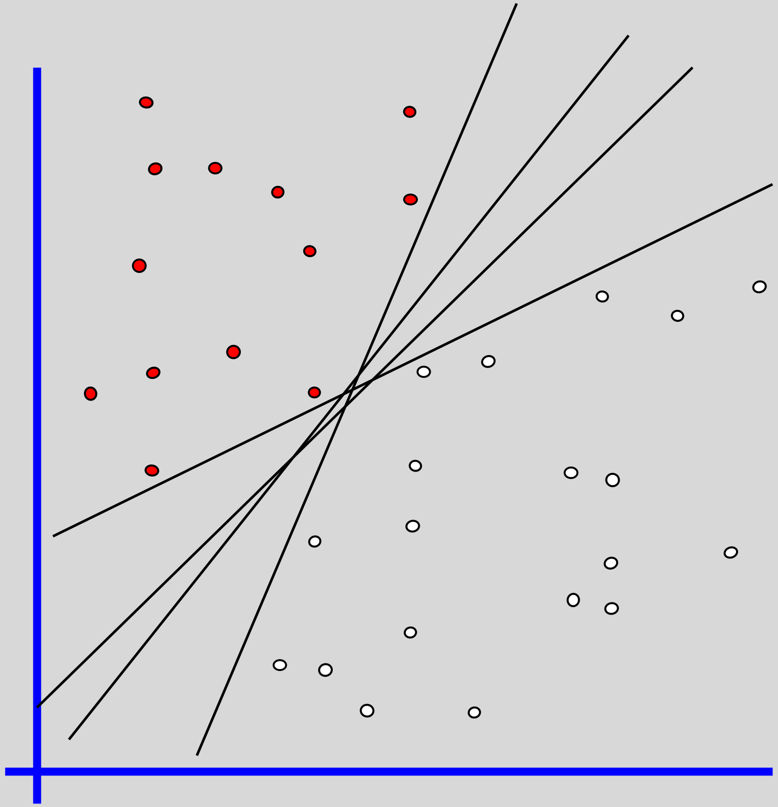
Extract features

Bag of words pipeline



Support vector machine (SVM)

- denotes +1
- denotes -1

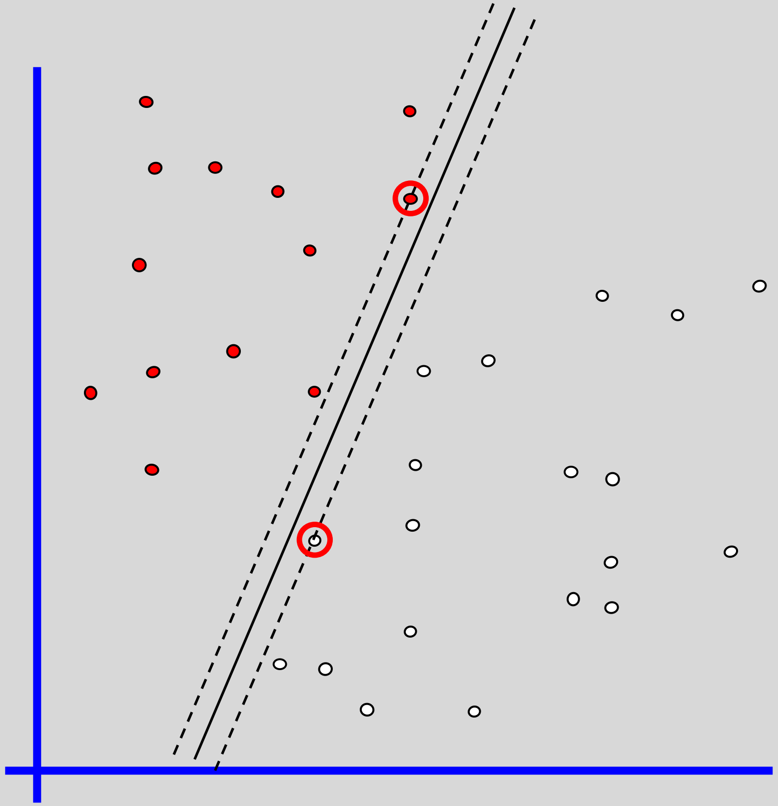


Any of these would be fine..

..but which is best?

Support vector machine (SVM)

- denotes +1
- denotes -1

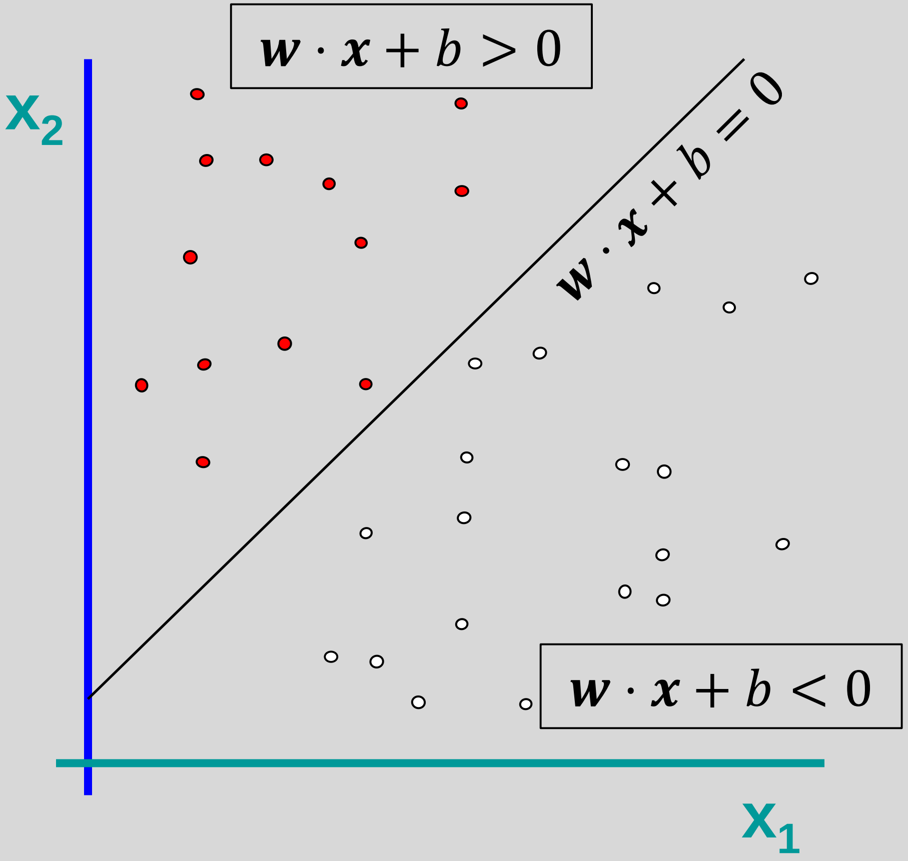


Define the **margin** of a linear classifier as the width that the boundary could be increased by before hitting a datapoint.

Support vector machine (SVM)

$$y_i = \mathrm{sign}(\boldsymbol{w} \cdot \boldsymbol{x} + b)$$

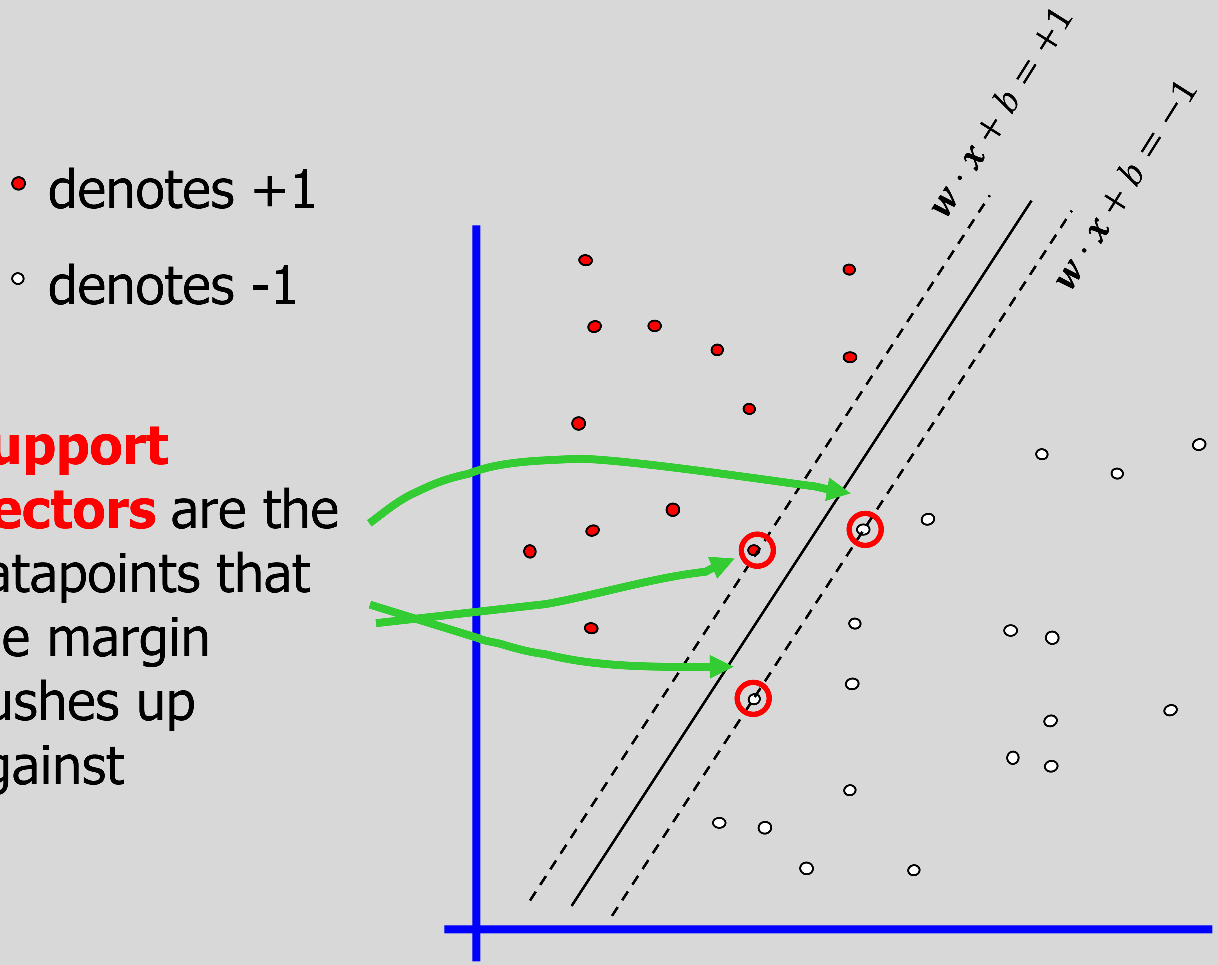
- denotes +1
- denotes -1



$\boldsymbol{x} = [x_1, x_2]$
$\boldsymbol{w} = [w_1, w_2]$

How would you classify this data?

Support vector machine (SVM)

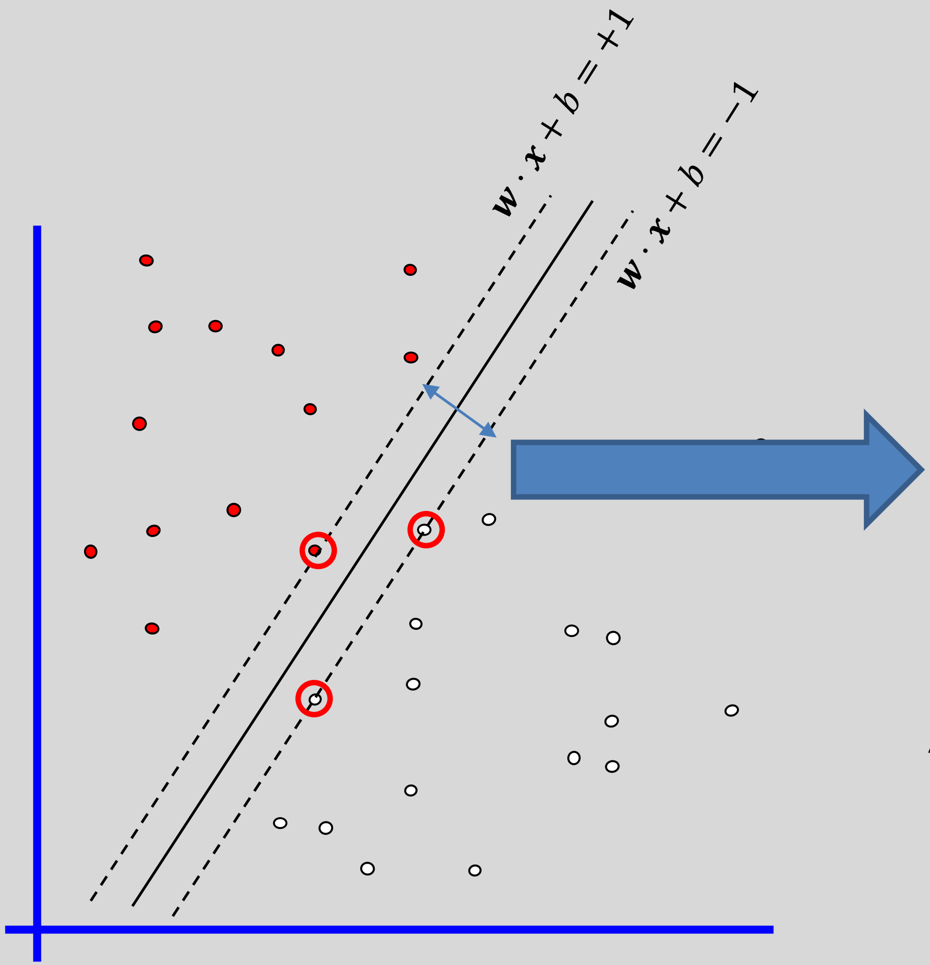


Support Vectors are the datapoints that the margin pushes up against

The **maximum margin linear classifier** is the linear classifier with the maximum margin.

This is the simplest kind of linear SVM.

Support vector machine (SVM)



Margin = 2/||w||

Distance between a point and a line:

Ax+By+C =0, (x0, y0) on wx+b=1

$$\rightarrow \frac{AX0+By0+C}{\sqrt{A^2+B^2}} = \frac{1}{||w||}$$

Support vector machine (SVM)

Margin width, can be shown to be $m = \frac{2}{\|\boldsymbol{w}\|}$.

We want to find maximum margin, i.e. we want to maximize m.

This is equivalent to minimizing $\frac{\|\boldsymbol{w}\|}{2}$.

However not every line with high margin is the solution.

The line has to have maximum margin, but it also must classify the data.

Support vector machine (SVM)

This leads to the following quadratic constrained optimization problem:

$$\text{minimize}_{\boldsymbol{w},b}\left(\frac{1}{2}\|\boldsymbol{w}\|\right)$$

$$\text{subject to} \quad y_i(\boldsymbol{w}\cdot\boldsymbol{x_i}+b)\geq 1 \quad i=1,\ldots,n$$

Constrained quadratic optimization is a standard problem in mathematical optimization.

Support vector machine (SVM)

- Constrained quadratic optimization leads to the following expansion of the weight vector $\boldsymbol{w}$ in terms of the input examples $\boldsymbol{x_i}$: (y_i is the output variable, i.e. +1 or -1)

$$\boxed{\boldsymbol{w} = \sum_{i=1}^{n} y_i \alpha_i \boldsymbol{x_i}}$$

- Only points on the margin (i.e. support vectors $\mathbf{x_i}$) have $\alpha_i > 0$.

$$\boldsymbol{w} \cdot \boldsymbol{x} + b = \sum_{i=1}^{n} y_i \alpha_i \boxed{\boldsymbol{x_i} \cdot \boldsymbol{x}} + b$$

dot product

Support vector machine (SVM)

Training SVM: find the sets of the parameters α_i and b.
Classification with SVM:

$$\text{class}(x_{unknown}) = \text{sign}\left(\sum_{i=1}^{n} y_i \alpha_i \boldsymbol{x_i} \cdot \boldsymbol{x_{uknown}} + b\right)$$

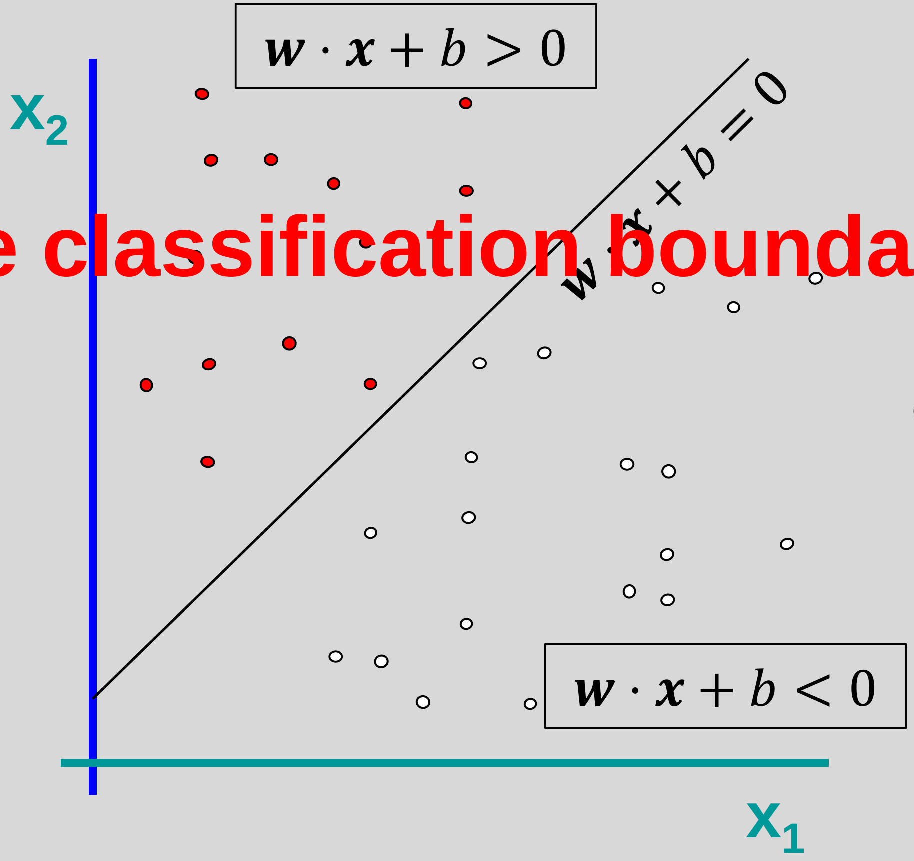
To classify a new pattern $\boldsymbol{x_{unknown}}$, it is only necessary to calculate the dot product between $\boldsymbol{x_{unknown}}$ and every support vector $\boldsymbol{x_i}$.
If the number of support vectors is small, computation time is significantly reduced.

Non-linear SVM

$$y_i = \mathrm{sign}(\boldsymbol{w} \cdot \boldsymbol{x} + b)$$

- denotes +1
- denotes -1

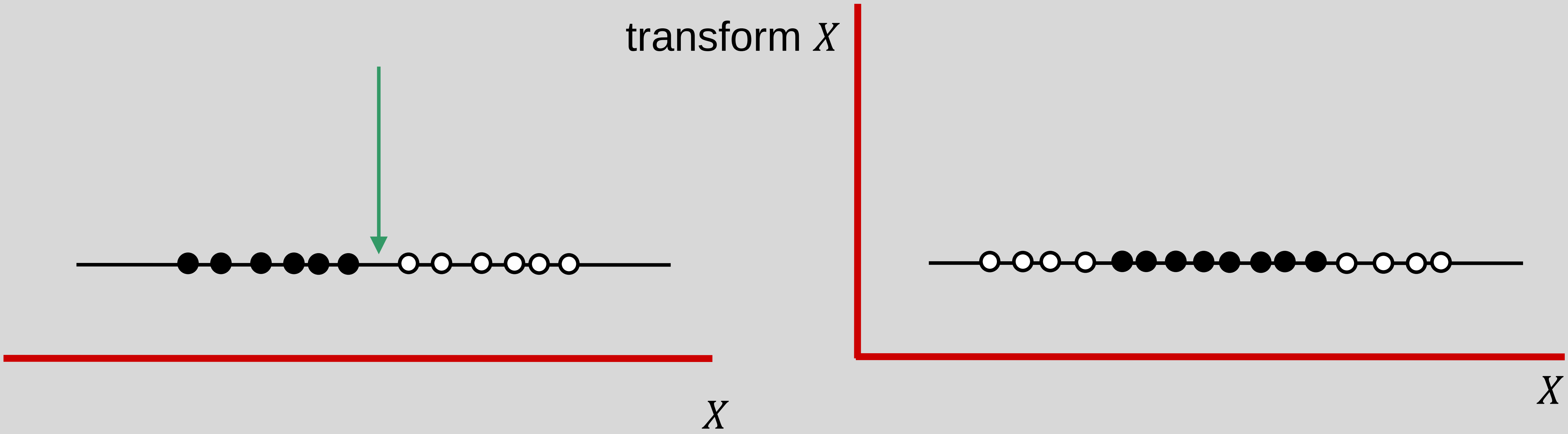
What to do if the classification boundary is non-linear?



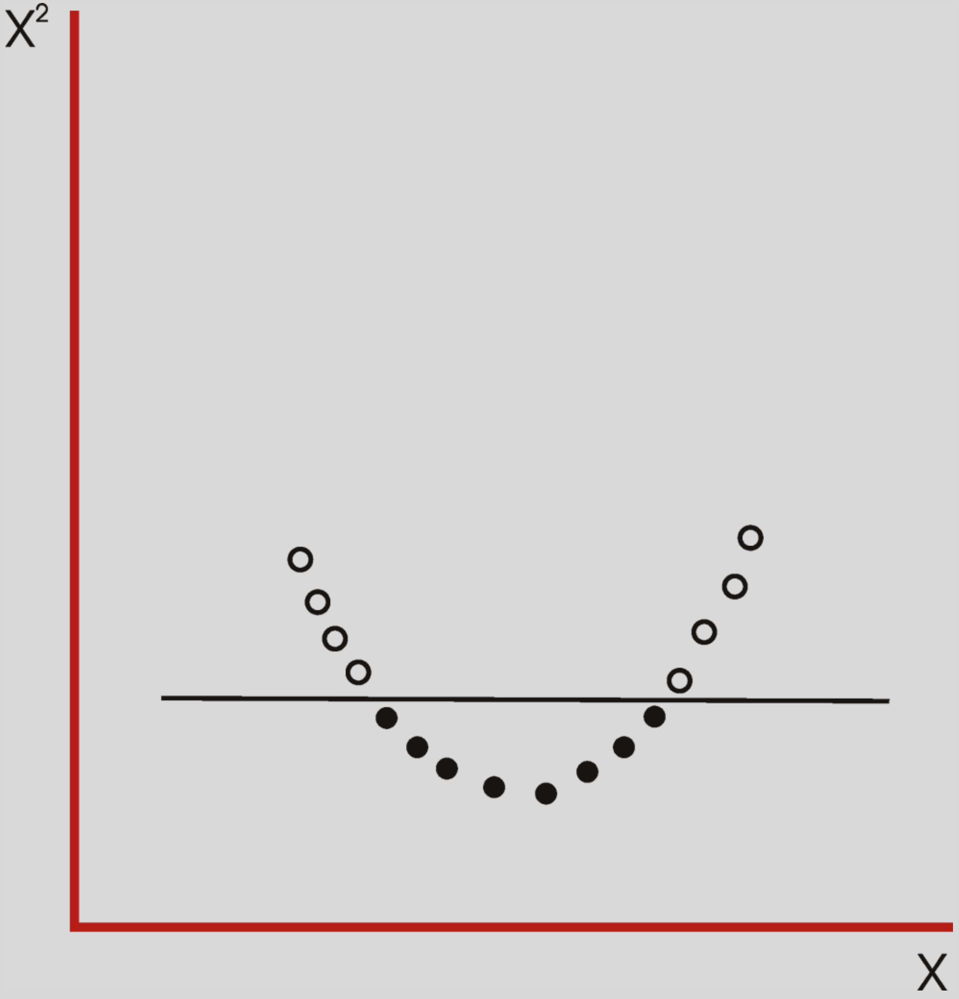
$\boldsymbol{x} = [x_1, x_2]$
$\boldsymbol{w} = [w_1, w_2]$

How would you classify this data?

Non-linear SVM



Non-linear SVM



Non-linear SVM

We know that the discriminant function is given by:

$$f(x) = \sum_{i=1}^{n} y_i \alpha_i \boldsymbol{x_i} \cdot \boldsymbol{x} + b$$

In the feature space $\mathcal{F}$ it becomes:

$$f(x) = \sum_{i=1}^{n} y_i \alpha_i \phi(\boldsymbol{x_i}) \cdot \phi(\boldsymbol{x}) + b$$

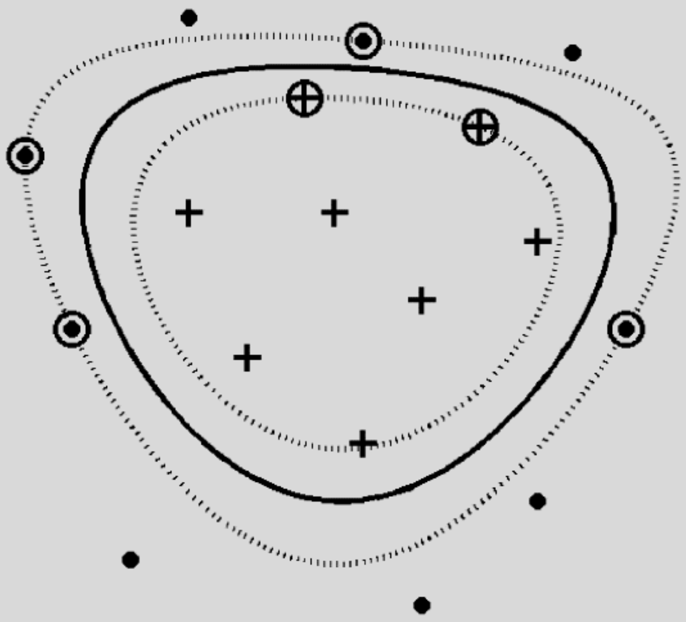
And now we use the so called **kernel trick**. We define kernel function:

$$k(\boldsymbol{x}, \boldsymbol{z}) = \phi(\boldsymbol{x}) \cdot \phi(\boldsymbol{z})$$

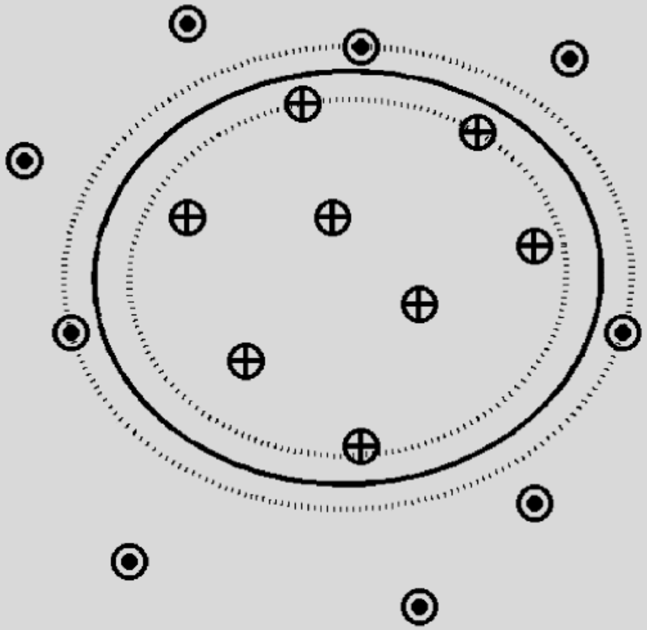
Non-linear SVM

Gaussian RBF Kernel

$$k(\mathbf{x},\mathbf{z}) = \exp\left(-\frac{\|\mathbf{x}-\mathbf{z}\|^2}{2\sigma^2}\right)$$



σ = 1



σ = 10

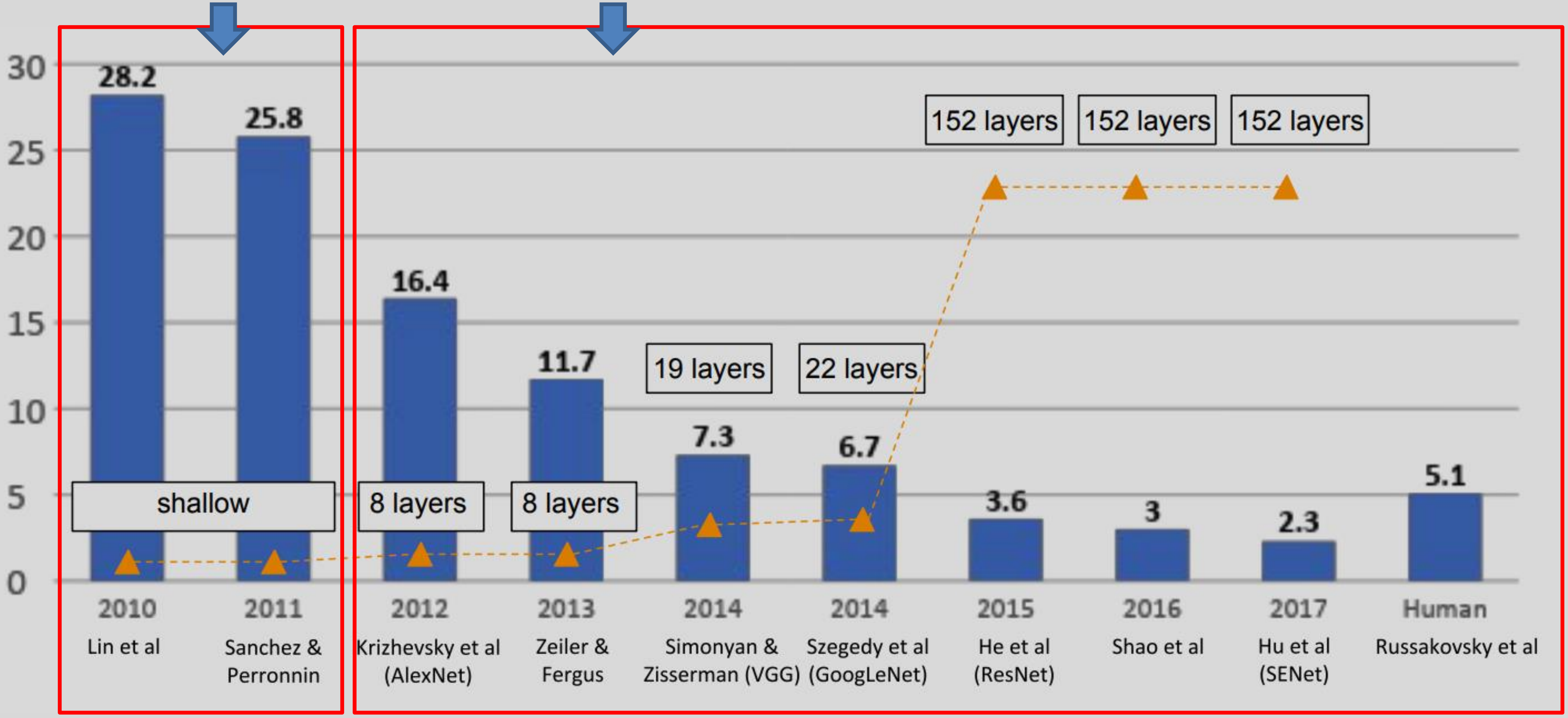
Multi-class SVM

- The simple implementation is the one-vs-all scheme.
- Train binary SVM for each class.

ImageNet challenge result

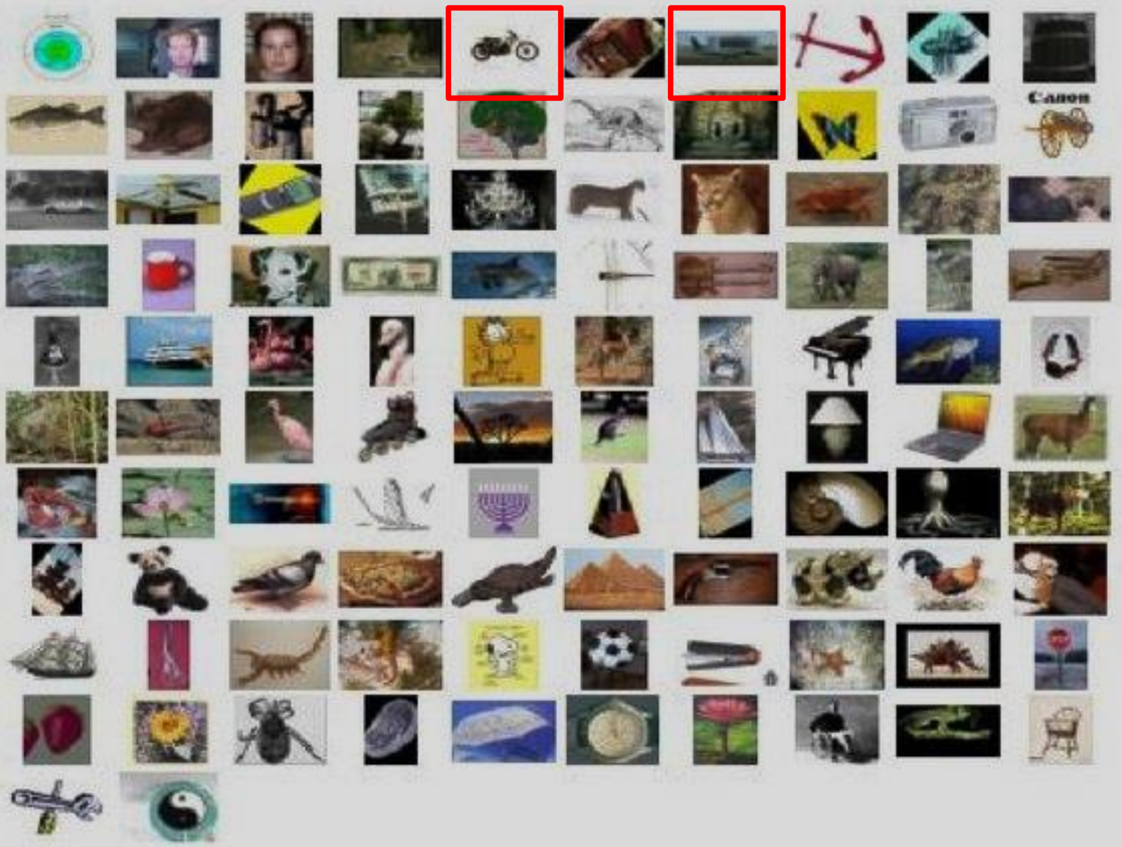
These are based on BOW representation.

Deep Learning era.



BOW Implementation – Feature extraction

Caltech 101 data : airplane vs. motorbike



Database link: http://www.vision.caltech.edu/Image_Datasets/Caltech101/101_ObjectCategories.tar.gz

BOW Implementation – Feature extraction

```
from google.colab import drive
drive.mount('/content/gdrive')

from google.colab.patches import cv2_imshow
import cv2
import numpy as np

categories =  ['airplanes', 'Motorbikes']
base_path = '/content/gdrive/My Drive/CSE472/BOW/Train_images'
detector = cv2.ORB_create()

train_paths = []
train_labels = []
train_features = np.array([])
img_len = 200
count = 0

for idx, category in enumerate(categories):
    dir_path = base_path + '/' + category

    for i in range(img_len):
        img_path = dir_path + '/' + 'image_%04d.jpg' % (i+1)
        train_paths.append(img_path)
        train_labels.append(idx)
        img = cv2.imread(img_path)
        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        kpt, desc= detector.detectAndCompute(gray, None)
        if train_features.size == 0:
          train_features = np.float32(desc)
        else:
          train_features = np.append(train_features, np.float32(desc), axis = 0)

        count+=1
        print('%d/%d - %s - %d feature points are detected\n' % (count,img_len*2, img_path, desc.shape[0]))
```

Upload data and will generate visual dictionary here.

BOW Implementation – Feature extraction

```
from google.colab import drive
drive.mount('/content/gdrive')

from google.colab.patches import cv2_imshow
import cv2
import numpy as np

categories =  ['airplanes', 'Motorbikes']
base_path = '/content/gdrive/My Drive/CSE472/BOW/Train_images'
detector = cv2.ORB_create()

train_paths = []
train_labels = []
train_features = np.array([])
img_len = 200
count = 0

for idx, category in enumerate(categories):
    dir_path = base_path + '/' + category

    for i in range(img_len):
        img_path = dir_path + '/' + 'image_%04d.jpg' % (i+1)
        train_paths.append(img_path)
        train_labels.append(idx)
        img = cv2.imread(img_path)
        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        kpt, desc= detector.detectAndCompute(gray, None)
        if train_features.size == 0:
          train_features = np.float32(desc)
        else:
          train_features = np.append(train_features, np.float32(desc), axis = 0)

        count+=1
        print('%d/%d - %s - %d feature points are detected\n' % (count,img_len*2, img_path, desc.shape[0]))
```

Upload data and will generate visual dictionary here.

Initialize feature extractor.

BOW Implementation – Feature extraction

```
from google.colab import drive
drive.mount('/content/gdrive')

from google.colab.patches import cv2_imshow
import cv2
import numpy as np

categories =  ['airplanes', 'Motorbikes']
base_path = '/content/gdrive/My Drive/CSE472/BOW/Train_images'
detector = cv2.ORB_create()

train_paths = []
train_labels = []
train_features = np.array([])
img_len = 200
count = 0

for idx, category in enumerate(categories):
    dir_path = base_path + '/' + category

    for i in range(img_len):
        img_path = dir_path + '/' + 'image_%04d.jpg' % (i+1)
        train_paths.append(img_path)
        train_labels.append(idx)
        img = cv2.imread(img_path)
        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        kpt, desc= detector.detectAndCompute(gray, None)
        if train_features.size == 0:
          train_features = np.float32(desc)
        else:
          train_features = np.append(train_features, np.float32(desc), axis = 0)

        count+=1
        print('%d/%d - %s - %d feature points are detected\n' % (count,img_len*2, img_path, desc.shape[0]))
```

Upload data and will generate visual dictionary here.

Initialize feature extractor.

Extract features for each train images,
and Concatenate them in one variable.

BOW Implementation – Generate visual codebook

```
dictionary_size = 50
dict_file='/content/gdrive/My Drive/CSE472/BOW/dictionary.npy'

criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 100, 0.1)
ret,label, dictionary=cv2.kmeans(train_features,dictionary_size,None,criteria,10,cv2.KMEANS_RANDOM_CENTERS)

np.save(dict_file, dictionary)
```

This will cluster extracted features into K visual words.
(This step will take few minutes)

BOW Implementation – Generate visual codebook

```
dictionary_size = 50
dict_file='/content/gdrive/My Drive/CSE472/BOW/dictionary.npy'

criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 100, 0.1)
ret,label, dictionary=cv2.kmeans(train_features,dictionary_size,None,criteria,10,cv2.KMEANS_RANDOM_CENTERS)

np.save(dict_file, dictionary)
```

This will cluster extracted features into K visual words.
(This step will take few minutes)

```
dictionary.shape
```

```
(50, 32)
```

We set K=50, feature dimension=32

BOW Implementation – Make image histograms

```
knn = cv2.ml.KNearest_create()
knn.train(dictionary, cv2.ml.ROW_SAMPLE, np.float32(range(dictionary_size)))
train_desc = np.float32(np.zeros((len(train_paths), dictionary_size)))

for i, path in enumerate(train_paths):
    img = cv2.imread(path)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    kpt, desc = detector.detectAndCompute(gray, None)

    ret, result, neighbours, dist = knn.findNearest(np.float32(desc), k=1)
    hist, bins = np.histogram(np.int32(result), bins=range(dictionary_size + 1))
    train_desc[i, :] = np.float32(hist) / np.float32(np.sum(hist))
    print('%d/%d - Representing %s \n' % (i, len(train_paths), img_path))
```

Use 1 nearest neighbor classifier.

BOW Implementation – Make image histograms

```
knn = cv2.ml.KNearest_create()
knn.train(dictionary, cv2.ml.ROW_SAMPLE, np.float32(range(dictionary_size)))
train_desc = np.float32(np.zeros((len(train_paths), dictionary_size)))

for i, path in enumerate(train_paths):
    img = cv2.imread(path)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    kpt, desc = detector.detectAndCompute(gray, None)

    ret, result, neighbours, dist = knn.findNearest(np.float32(desc), k=1)
    hist, bins = np.histogram(np.int32(result), bins=range(dictionary_size + 1))
    train_desc[i, :] = np.float32(hist) / np.float32(np.sum(hist))
    print('%d/%d - Representing %s \n' % (i, len(train_paths), img_path))
```

Use 1 nearest neighbor classifier.

Extract feature descriptor.

BOW Implementation – Make image histograms

```
knn = cv2.ml.KNearest_create()
knn.train(dictionary, cv2.ml.ROW_SAMPLE, np.float32(range(dictionary_size)))
train_desc = np.float32(np.zeros((len(train_paths), dictionary_size)))

for i, path in enumerate(train_paths):
    img = cv2.imread(path)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    kpt, desc = detector.detectAndCompute(gray, None)

    ret, result, neighbours, dist = knn.findNearest(np.float32(desc), k=1)
    hist, bins = np.histogram(np.int32(result), bins=range(dictionary_size + 1))
    train_desc[i, :] = np.float32(hist) / np.float32(np.sum(hist))
    print('%d/%d - Representing %s \n' % (i, len(train_paths), img_path))
```

Use 1 nearest neighbor classifier.

Extract feature descriptor.

Find nearest codeword
and map into histogram.

BOW Implementation – Make image histograms

```
knn = cv2.ml.KNearest_create()
knn.train(dictionary, cv2.ml.ROW_SAMPLE, np.float32(range(dictionary_size)))
train_desc = np.float32(np.zeros((len(train_paths), dictionary_size)))

for i, path in enumerate(train_paths):
    img = cv2.imread(path)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    kpt, desc = detector.detectAndCompute(gray, None)

    ret, result, neighbours, dist = knn.findNearest(np.float32(desc), k=1)
    hist, bins = np.histogram(np.int32(result), bins=range(dictionary_size + 1))
    train_desc[i, :] = np.float32(hist) / np.float32(np.sum(hist))
    print('%d/%d: Representing %s \n' % (i, len(train_paths), img_path))
```

Use 1 nearest neighbor classifier.

Extract feature descriptor.

Find nearest codeword
and map into histogram.

Accumulate all histograms in 'train_desc'.

BOW Implementation – Train SVM

```
svm_model_file = '/content/gdrive/My Drive/CSE472/BOW/svmmodel.xml'
svm = cv2.ml.SVM_create()
svm.trainAuto(train_desc, cv2.ml.ROW_SAMPLE, np.array(train_labels))
svm.save(svm_model_file)
```

Train the SVM classifier.

BOW Implementation – Testing

```
test_desc = np.float32(np.zeros((2, dictionary_size)))

img_path1 = base_path + '/' + categories[0] +'/image_0600.jpg'
img_path2 = base_path + '/' + categories[1] +'/image_0600.jpg'

img1 = cv2.imread(img_path1)
gray1 = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
kpt1, desc1= detector.detectAndCompute(gray1, None)
ret, result1, neighbours, dist = knn.findNearest(np.float32(desc1), k=1)
hist1,bins = np.histogram(np.int32(result1),bins=range(dictionary_size + 1))

img2 = cv2.imread(img_path2)
gray2 = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)
kpt2, desc2= detector.detectAndCompute(gray2, None)
ret, result2, neighbours, dist = knn.findNearest(np.float32(desc2), k=1)
hist2,bins = np.histogram(np.int32(result2),bins=range(dictionary_size + 1))

test_desc[0, :] = np.float32(hist1) / np.float32(np.sum(hist1))
test_desc[1, :] = np.float32(hist2) / np.float32(np.sum(hist2))

ret, result = svm.predict(test_desc)
```

Load some testing images.

BOW Implementation – Testing

```
test_desc = np.float32(np.zeros((2, dictionary_size)))

img_path1 = base_path + '/' + categories[0] +'/image_0600.jpg'
img_path2 = base_path + '/' + categories[1] +'/image_0600.jpg'

img1 = cv2.imread(img_path1)
gray1 = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
kpt1, desc1= detector.detectAndCompute(gray1, None)
ret, result1, neighbours, dist = knn.findNearest(np.float32(desc1), k=1)
hist1,bins = np.histogram(np.int32(result1),bins=range(dictionary_size + 1))

img2 = cv2.imread(img_path2)
gray2 = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)
kpt2, desc2= detector.detectAndCompute(gray2, None)
ret, result2, neighbours, dist = knn.findNearest(np.float32(desc2), k=1)
hist2,bins = np.histogram(np.int32(result2),bins=range(dictionary_size + 1))

test_desc[0, :] = np.float32(hist1) / np.float32(np.sum(hist1))
test_desc[1, :] = np.float32(hist2) / np.float32(np.sum(hist2))

ret, result = svm.predict(test_desc)
```

Load some testing images.

Make BOW representation for each.

BOW Implementation – Testing

```
test_desc = np.float32(np.zeros((2, dictionary_size)))

img_path1 = base_path + '/' + categories[0] +'/image_0600.jpg'
img_path2 = base_path + '/' + categories[1] +'/image_0600.jpg'

img1 = cv2.imread(img_path1)
gray1 = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
kpt1, desc1= detector.detectAndCompute(gray1, None)
ret, result1, neighbours, dist = knn.findNearest(np.float32(desc1), k=1)
hist1,bins = np.histogram(np.int32(result1),bins=range(dictionary_size + 1))

img2 = cv2.imread(img_path2)
gray2 = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)
kpt2, desc2= detector.detectAndCompute(gray2, None)
ret, result2, neighbours, dist = knn.findNearest(np.float32(desc2), k=1)
hist2,bins = np.histogram(np.int32(result2),bins=range(dictionary_size + 1))

test_desc[0, :] = np.float32(hist1) / np.float32(np.sum(hist1))
test_desc[1, :] = np.float32(hist2) / np.float32(np.sum(hist2))

ret, result = svm.predict(test_desc)
```

Load some testing images.

Make BOW representation for each.

Do SVM classification.

BOW Implementation – Testing

```
result
```

```
array([[0.],
       [1.]], dtype=float32)
```

SVM classifies the first image as ‘airplane’ and the second image as 'mortorbike'.

```
from google.colab.patches import cv2_imshow

cv2_imshow(img1)
cv2_imshow(img2)
```

Visualize the first and second images.

Exchangeability

- Orderless document representation: frequencies of words from a dictionary Salton & McGill (1983)

Exchangeability

documents

words

$$p(x_1, x_2, \ldots, x_N) = \int p(\theta)\left(\prod_{i=1}^{N} p(\boxed{x_i} \mid \theta)\right) d\theta$$

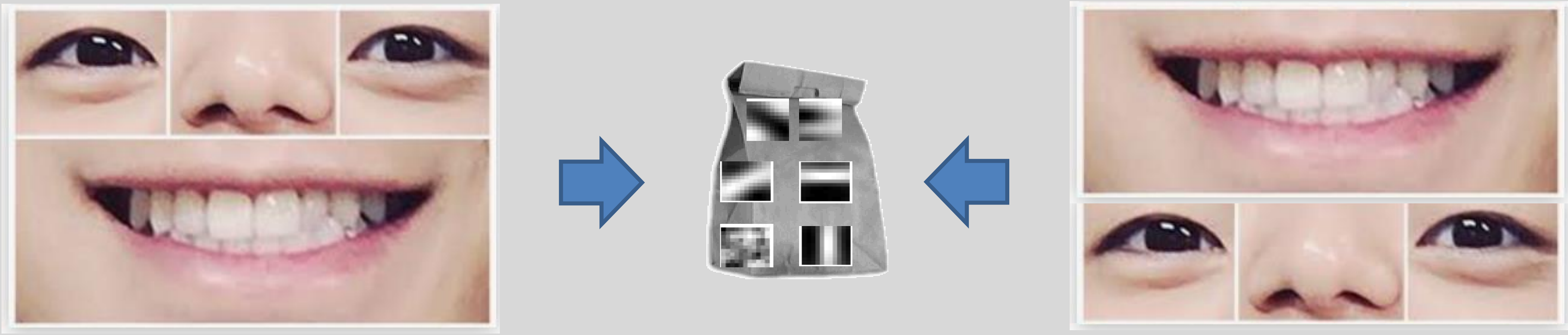
De Finetti Theorem of exchangeability: the joint probability distribution of words is invariant to permutation.

Exchangeability

Document 1 = “the quick brown fox jumped”
Document 2 = “brown quick jumped fox the”

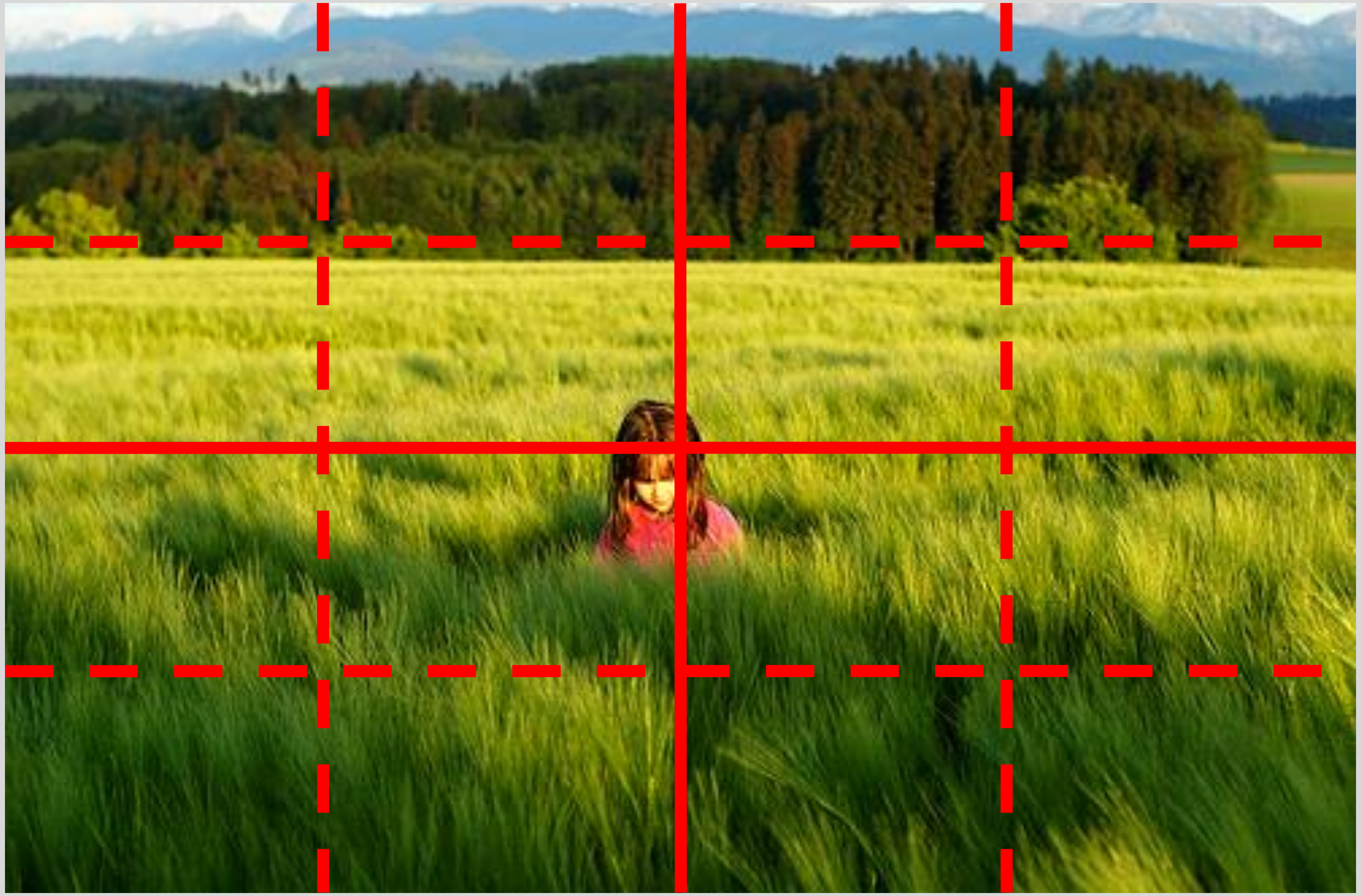
Would a bag of words model represent these two documents differently?

Spatial configuration

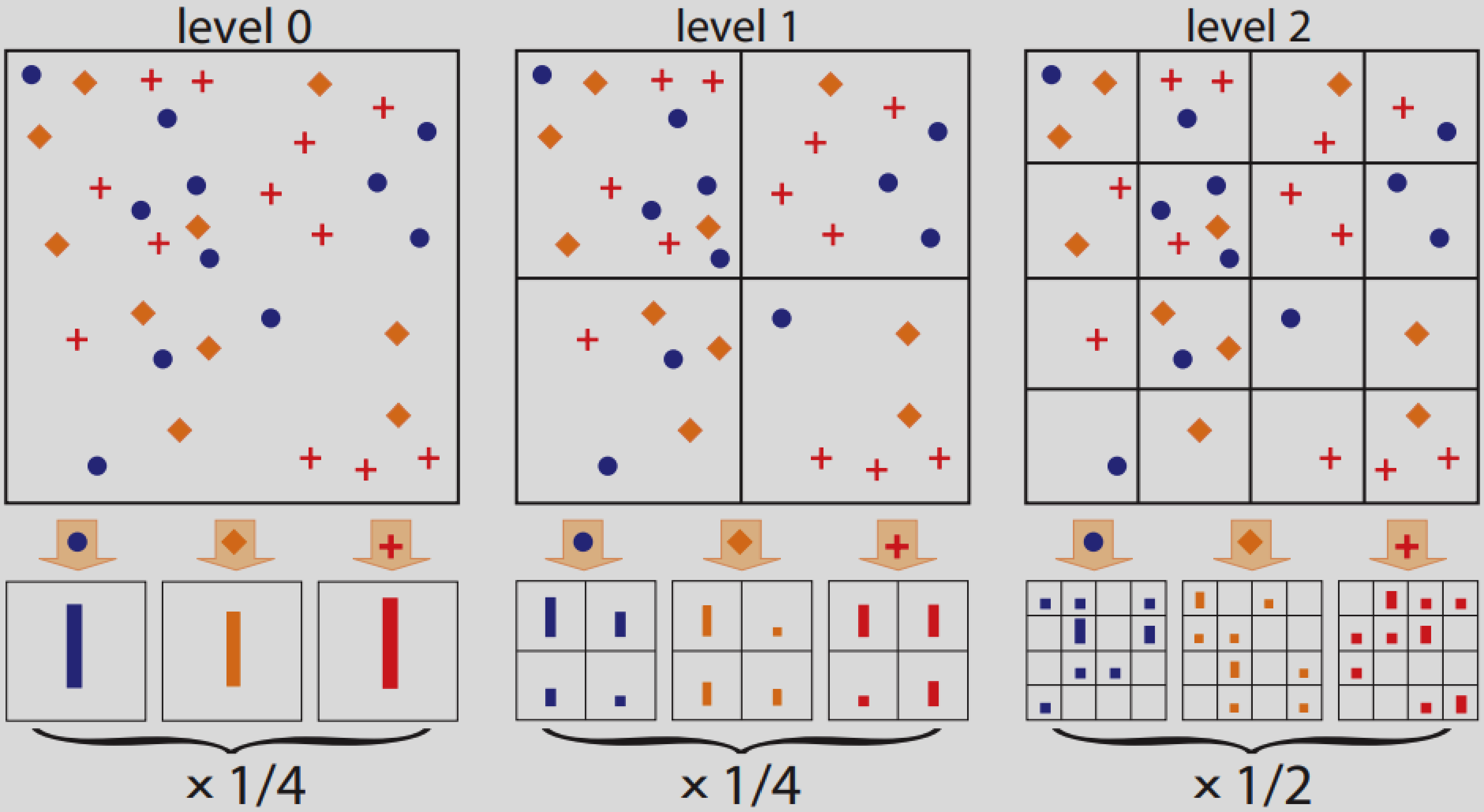


Two different images will have similar BOW representation.

Spatial configuration



Spatial configuration



Thank you!

UNIST

ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

2007