Support vector machines (SVMs)

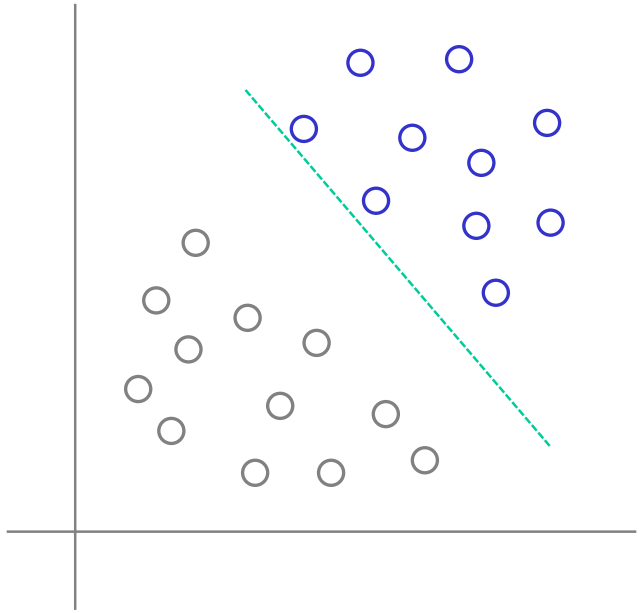
Announcements

- A2 grades will be uploaded later today or tomorrow
- A3 due on December 2nd
- Optional A4 will be released this week (it will be due during finals week)
- Final exam is on December 16

Reminder about perceptrons

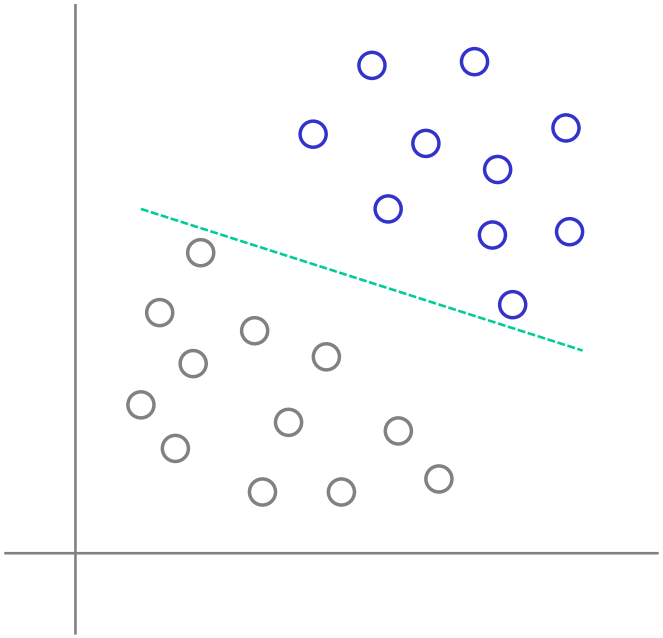
- **Perceptron Convergence Theorem**: If a classification problem is linearly separable, a perceptron will reach a solution in a finite number of iterations

- The solution weight vector is **not** unique. There are infinite possible solutions and decision boundaries.
  - Perceptrons find any separating hyperplane
  - The hyperplane depends on initialization and ordering of training points



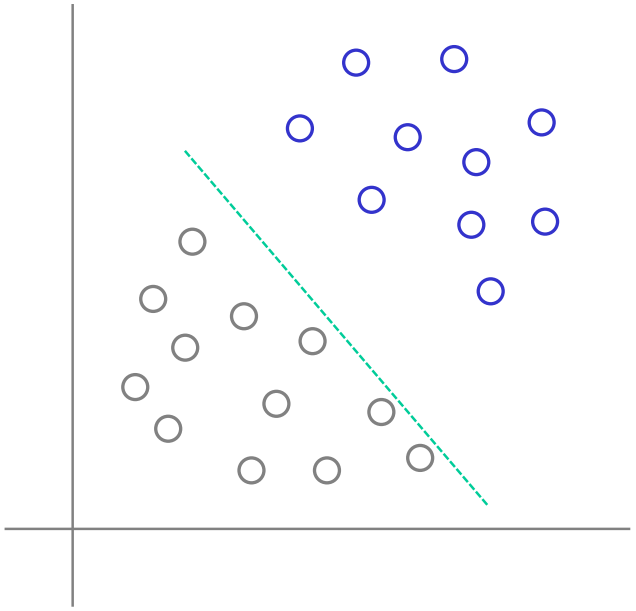
Reminder about perceptrons

- **Perceptron Convergence Theorem**: If a classification problem is linearly separable, a perceptron will reach a solution in a finite number of iterations
- The solution weight vector is **not** unique. There are infinite possible solutions and decision boundaries.
  - Perceptrons find any separating hyperplane
  - The hyperplane depends on initialization and ordering of training points
- ***If done differently***



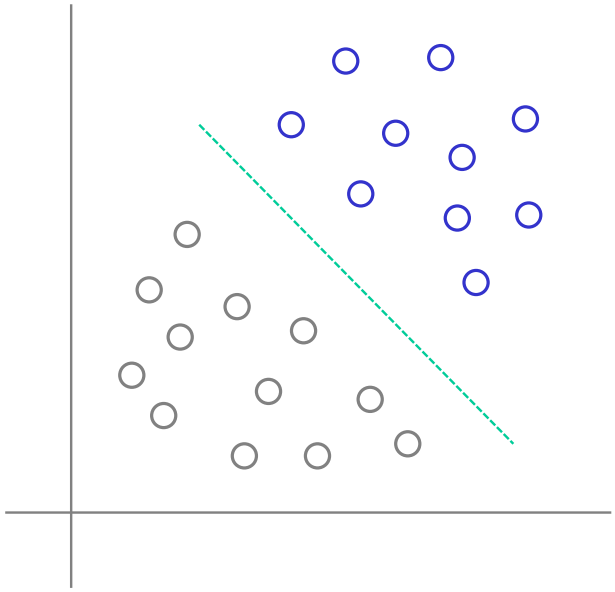
Reminder about perceptrons

- **Perceptron Convergence Theorem**: If a classification problem is linearly separable, a perceptron will reach a solution in a finite number of iterations
- The solution weight vector is **not** unique. There are infinite possible solutions and decision boundaries.
  - Perceptrons find any separating hyperplane
  - The hyperplane depends on initialization and ordering of training points
- ***If done differently….Again***



Reminder about perceptrons

- **Perceptron Convergence Theorem**: If a classification problem is linearly separable, a perceptron will reach a solution in a finite number of iterations
- The solution weight vector is **not** unique. There are infinite possible solutions and decision boundaries.
  - Perceptrons find any separating hyperplane
  - The hyperplane depends on initialization and ordering of training points
- ***If done differently….Again...And Again***

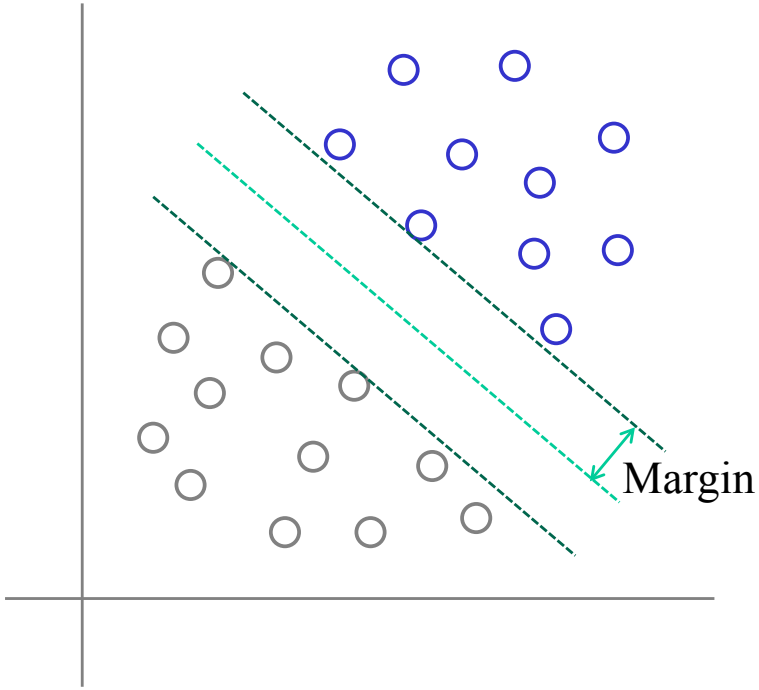


Motivation

- For a linearly separable classification task, there are generally infinitely many separating hyperplanes
  - Perceptron learning, however, stops as soon as one of them is reached
  - Some hyperplanes may be better than others

- To improve generalization, we want to place a decision boundary as far away from training classes as possible.
  - In other words, place the boundary at equal distances from class boundaries

Optimal Hyperplane



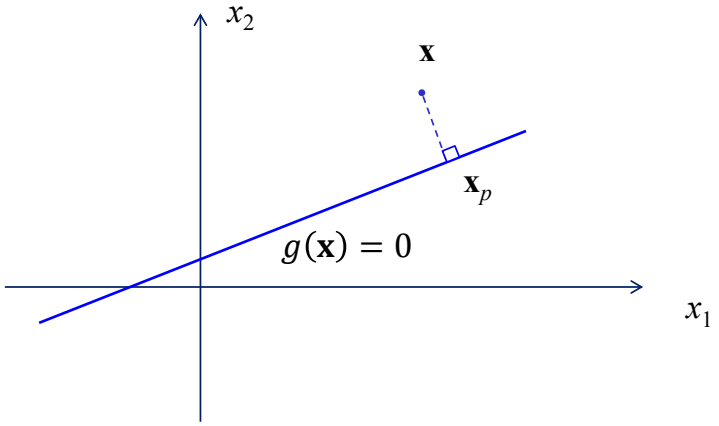
- Given a training sample, the support vector machine constructs a hyperplane as the decision surface in such a way that the margin of separation between positive and negative examples is maximized.

Decision Boundary

- Given a linear discriminant function

$$g(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b = 0$$

- To find its distance to a given pattern **x**, project **x** onto the decision boundary



Decision Boundary (cont.)

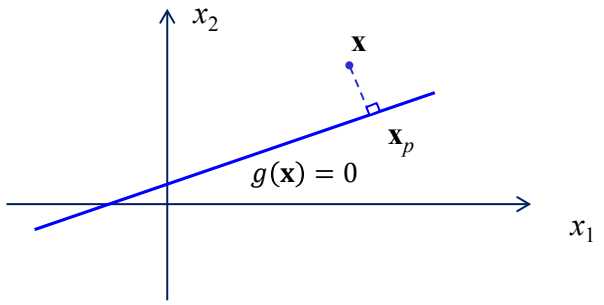
- **x** can be re-written as a function of the projection and the weights

$$\mathbf{x} = \mathbf{x}_p + r\frac{\mathbf{w}}{||\mathbf{w}||}$$

- $\mathbf{x}_p$ is **x**'s projection
- The second term arises from the fact that the weight vector is perpendicular to the decision boundary

- The algebraic distance *r* is positive if **x** is on the positive side of the boundary and negative if **x** is on the negative side

$$g(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b = 0$$

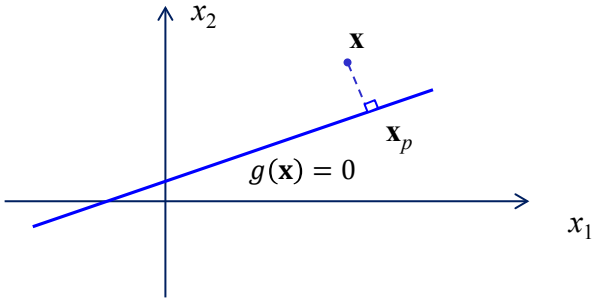


Decision Boundary (cont.)

- Since **x** can be written in terms of its projection and *r*, then the decision boundary can as well:

$$
\begin{aligned}
g(\mathbf{x}) &= g\left(\mathbf{x}_p + r\frac{\mathbf{w}}{||\mathbf{w}||}\right) \\
&= \mathbf{w}^T\left(\mathbf{x}_p + r\frac{\mathbf{w}}{||\mathbf{w}||}\right) + b \\
&= \mathbf{w}^T\mathbf{x}_p + b + r||\mathbf{w}|| \\
&= r||\mathbf{w}||
\end{aligned}
$$

$$g(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b = 0$$

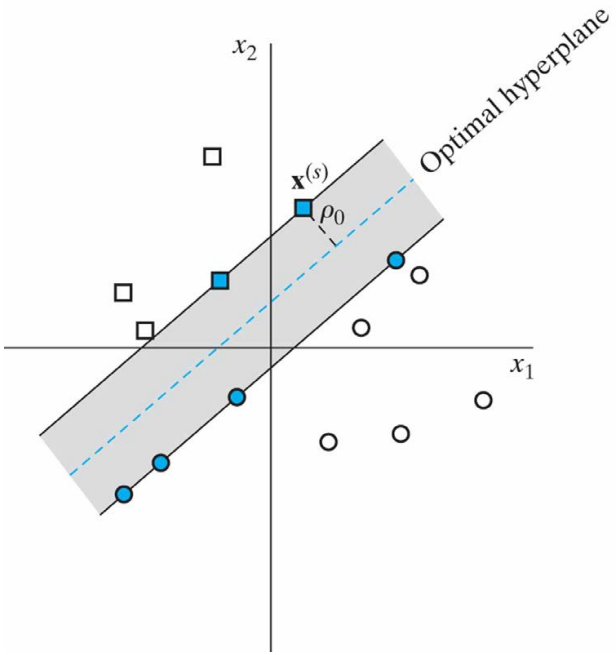


- Thus, $r = \frac{g(\mathbf{x})}{||\mathbf{w}||}$

Margin of Separation

- The ***margin of separation*** is the smallest distance of the hyperplane to a data set

- The training patterns (i.e. inputs) closest to the optimal hyperplane are called ***support vectors***
  - We'll show that non-support vectors have no role



Finding the Optimal Hyperplane for Linearly Separable Problems

- **Question**: Given N pairs of inputs and desired outputs $<\mathbf{x}_i, d_i>$, How to find $\mathbf{w}_o$ and b_o for the optimal hyperplane?

- Without loss of generality, $\mathbf{w}_o$ and b_o must satisfy

$$\mathbf{w}_o{}^T \mathbf{x}_i + b_o \geq 1 \quad \text{for } d_i = 1$$
$$\mathbf{w}_o{}^T \mathbf{x}_i + b_o \leq -1 \quad \text{for } d_i = -1$$

or

$$d_i(\mathbf{w}_o{}^T \mathbf{x}_i + b_o) \geq 1$$

where the equality holds for support vectors only

Optimal Hyperplane

- For a support vector $\mathbf{x}^{(s)}$, its algebraic distance to the optimal hyperplane is:

$$r = \frac{g(\mathbf{x}^{(s)})}{||\mathbf{w}_o||} = \begin{cases} \frac{1}{||\mathbf{w}_o||} & \text{if } d^{(s)} = 1 \\ \frac{-1}{||\mathbf{w}_o||} & \text{if } d^{(s)} = -1 \end{cases}$$

$$\mathbf{w}_o{}^T\mathbf{x}_i + b_o \geq 1 \quad \text{for } d_i = 1$$
$$\mathbf{w}_o{}^T\mathbf{x}_i + b_o \leq -1 \quad \text{for } d_i = -1$$

- Thus, the margin of separation (from optimal hyperplane to support vector) is:

$$|r| = \frac{1}{||\mathbf{w}_o||}$$

In other words, maximizing the margin of separation is equivalent to minimizing $||\mathbf{w}_o||$

- The optimal margin of separation between support vectors from the two classes is:

$$\rho = 2|r| = \frac{2}{||\mathbf{w}_o||}$$

Primal Problem

- Therefore $\mathbf{w}_o$ and b_o satisfy

$$d_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 \quad \text{for } i = 1, \ldots, N$$

and $\Phi(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T\mathbf{w}$ is minimized

- The above constrained minimization problem is called the ***primal*** problem

Lagrangian Formulation

- Using Lagrangian formulation, we construct the Lagrangian function:

$$J(\mathbf{w}, b, \alpha) = \frac{1}{2}\mathbf{w}^T\mathbf{w} - \sum_{i=1}^{N} \alpha_i [d_i(\mathbf{w}^T\mathbf{x}_i + b) - 1]$$

where $\alpha_i \geq 0$

  - where nonnegative variables, $\alpha_i's$ are called Lagrange multipliers

Lagrangian Formulation

- The solution of the optimization problem is a saddle point, minimized with respect to **w** and b, but maximized with respect to α

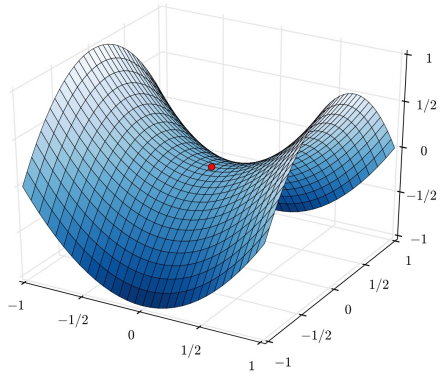
$$\arg\max_{\boldsymbol{\alpha}} \left(\arg\min_{\boldsymbol{w},b} J(\boldsymbol{w}, b, \boldsymbol{\alpha}) \right)$$

- Lets start by solving: $\arg\min_{\boldsymbol{w},b} J(\boldsymbol{w}, b, \boldsymbol{\alpha})$
  - Condition 1:

$$\frac{\partial J(\mathbf{w}, b, \alpha)}{\partial \mathbf{w}} = 0$$

  - Condition 2:

$$\frac{\partial J(\mathbf{w}, b, \alpha)}{\partial b} = 0$$



Saddle point (source: Wikipedia)

Lagrangian Formulation

$$J(\mathbf{w}, b, \alpha) = \frac{1}{2}\mathbf{w}^T\mathbf{w} - \sum_{i=1}^{N} \alpha_i [d_i(\mathbf{w}^T\mathbf{x}_i + b) - 1]$$

where $\alpha_i \geq 0$

- From condition 1:

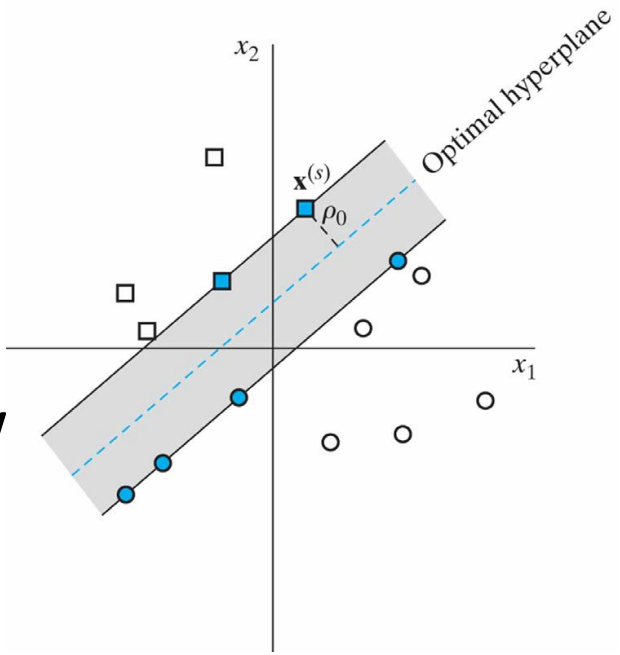
$$\frac{\partial J(\mathbf{w}, b, \alpha)}{\partial \mathbf{w}} = 0 \quad \Rightarrow \quad \mathbf{w} - \sum_i \alpha_i d_i \mathbf{x}_i = 0 \quad \Rightarrow \quad \mathbf{w} = \sum_i \alpha_i d_i \mathbf{x}_i$$

- From condition 2:

$$\frac{\partial J(\mathbf{w}, b, \alpha)}{\partial b} = 0 \quad \Rightarrow \quad \sum_i \alpha_i d_i = 0$$

Finding Optimal Weights

- To solve: $\arg\min_{\boldsymbol{w},b} J(\boldsymbol{w}, b, \boldsymbol{\alpha})$
  - We found that: $\mathbf{w} = \sum_i \alpha_i d_i \mathbf{x}_i$
  - Under constraints $\alpha_i \geq 0$ and $\sum_i \alpha_i d_i$
- Still need to solve for α_i's (will show (later)



How to Find α? Dual Problem

$$J(\boldsymbol{w}, b, \alpha) = \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} - \sum_{i=1}^{N} \alpha_i [d_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) - 1]$$

- The cost function can be first be expanded

$$= \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} - \sum_{i=1}^{N} \alpha_i d_i \boldsymbol{w}^T\boldsymbol{x}_i - b\sum_{i=1}^{N} \alpha_i d_i + \sum_{i=1}^{N} \alpha_i$$

- It can then be simplified using the solution for ***w***

$$\mathbf{w} = \sum_i \alpha_i d_i \mathbf{x}_i \quad \Rightarrow \quad \mathbf{w}^T\mathbf{w} = \sum_i \alpha_i d_i \mathbf{w}^T\mathbf{x}_i$$

$$\Rightarrow \quad J(\boldsymbol{w}, b, \alpha) = -\frac{1}{2}\sum_{i=1}^{N} \alpha_i d_i \boldsymbol{w}^T\boldsymbol{x}_i - b\sum_{i=1}^{N} \alpha_i d_i + \sum_{i=1}^{N} \alpha_i$$

How to Find α? Dual Problem (Cont.)

$$J(\boldsymbol{w}, b, \alpha) = -\frac{1}{2}\sum_{i=1}^{N}\alpha_i d_i \boldsymbol{w}^T \boldsymbol{x}_i - b\sum_{i=1}^{N}\alpha_i d_i + \sum_{i=1}^{N}\alpha_i$$

- After, plugging in for ***w*** and removing the 2nd term from our constraint

$$\sum_i \alpha_i d_i = 0$$

$$\mathbf{w} = \sum_i \alpha_i d_i \mathbf{x}_i$$

- Hence, *J(**w**,b,α)* becomes

$$Q(\alpha) = \sum_i \alpha_i - \frac{1}{2}\sum_i\sum_j \alpha_i \alpha_j d_i d_j \mathbf{x}_i{}^T \mathbf{x}_j$$

Dual Problem

- The solution to the Lagrangian function, is a saddle point, minimized w.r.t. **w** and b, but maximized w.r.t. α

$$\arg\max_{\boldsymbol{\alpha}} \left(\arg\min_{\boldsymbol{w},b} J(\boldsymbol{w}, b, \boldsymbol{\alpha}) \right)$$

- The ***dual*** problem is stated as follows:
  - The Lagrange multipliers maximize

$$Q(\alpha) = \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j d_i d_j \mathbf{x}_i{}^T \mathbf{x}_j$$

  - subject to (the prior constraints):

$$(1) \sum_i \alpha_i d_i = 0$$
$$(2) \alpha_i \geq 0$$

The dual problem can be solved as a quadratic optimization problem (Q is quadratic in terms of α)

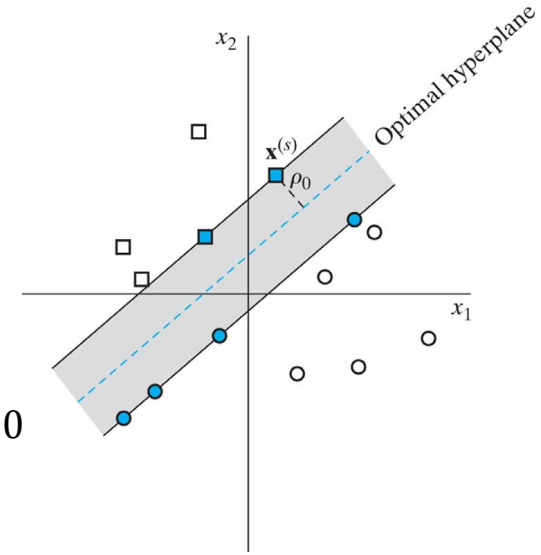
Karush-Kuhn-Tucker Conditions

- ***Remark***: The above constrained optimization problem satisfies:

$$\alpha_i[d_i(\mathbf{w}^T\mathbf{x}_i + b) - 1] = 0$$

called the Karush-Kuhn-Tucker conditions

- In other words, $\alpha_i = 0$ when $d_i(\mathbf{w}^T\mathbf{x}_i + b) - 1 > 0$
- α_i can be greater than 0 only when $d_i(\mathbf{w}^T\mathbf{x}_i + b) - 1 = 0$, that is, for support vectors



$$\mathbf{w}_o{}^T\mathbf{x}_i + b_o \geq 1 \quad \text{for } d_i = 1$$
$$\mathbf{w}_o{}^T\mathbf{x}_i + b_o \leq -1 \quad \text{for } d_i = -1$$

$$J(\mathbf{w}, b, \alpha) = \frac{1}{2}\mathbf{w}^T\mathbf{w} - \sum_{i=1}^{N} \alpha_i[d_i(\mathbf{w}^T\mathbf{x}_i + b) - 1]$$

Solution

$\mathbf{w} = \sum_i \alpha_i d_i \mathbf{x}_i$

- Having found optimal multipliers, $\alpha_{o,i}$

$$\mathbf{w}_o = \sum_{i=1}^{N_S} \alpha_{o,i} d_i \mathbf{x}_i$$

- where N_s is the number of support vectors

Finding b?

- For any support vector $\mathbf{x}^{(s)}$, we have

$$d^{(s)}\left(\mathbf{w}_o{}^T\mathbf{x}^{(s)} + b_o\right) = 1$$

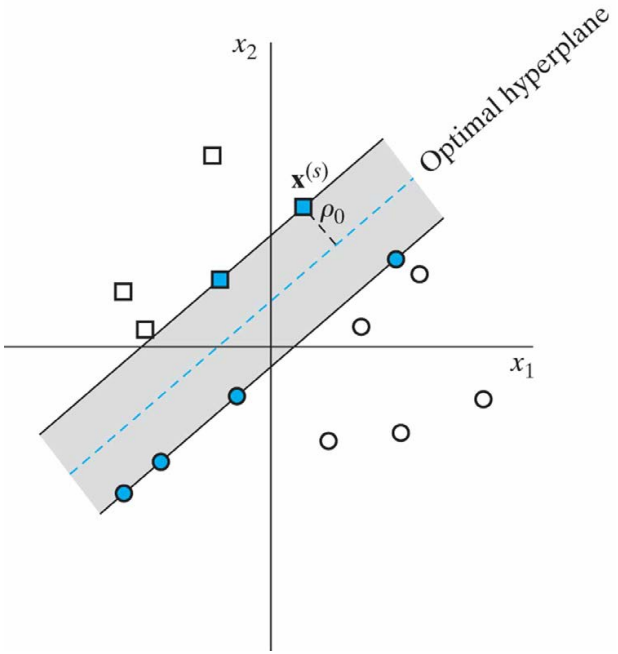
- Hence,

$$b_o = \frac{1}{d^{(s)}} - \mathbf{w}_o{}^T\mathbf{x}^{(s)} = \frac{1}{d^{(s)}} - \sum_{i=1}^{N_S} \alpha_{o,i} d_i \mathbf{x}_i{}^T\mathbf{x}^{(s)}$$

- Note that for robustness, one should average over all support vectors to compute b_o

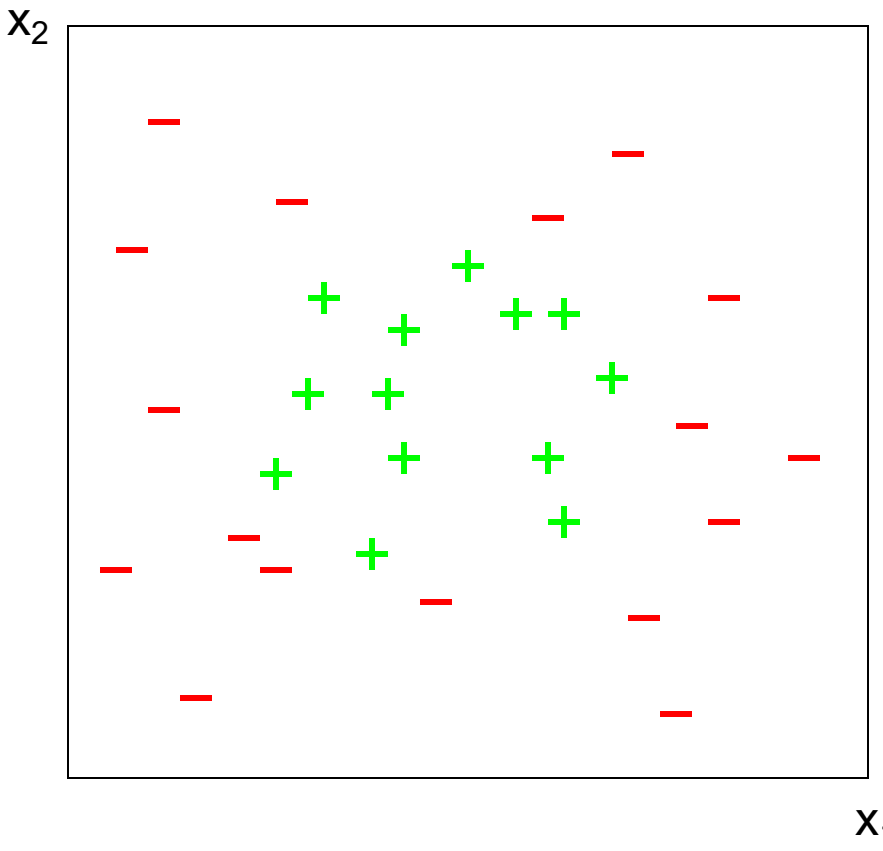
Interim summary

- Support vectors are the training patterns (i.e. inputs) closest to the optimal hyperplane
- To improve generalization, we want to place a decision boundary as far away from training classes as possible.
- Finding the maximum margin hyperplane has been formulated as a constrained quadratic problem
  - Convex problem, well studied, conceptually easy to solve
  - It can be solved in the primal or dual formulation
  - Only some data points contribute to the solution (i.e. support vectors)
  - So far, only applies to linearly separable data



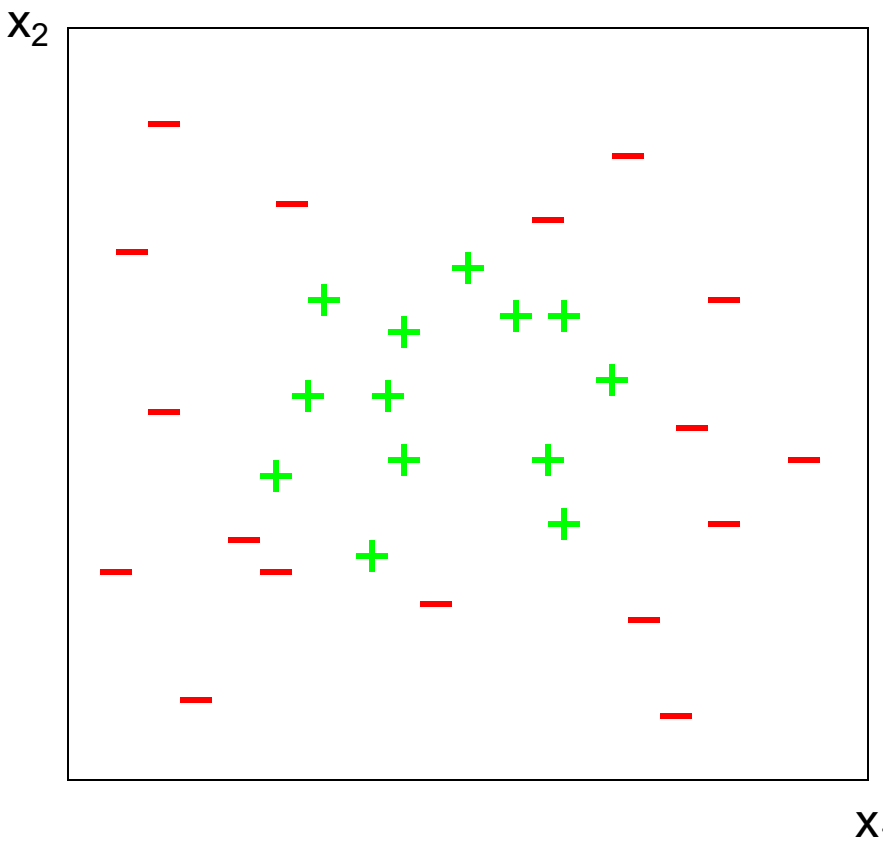
What if space is not linearly separable?

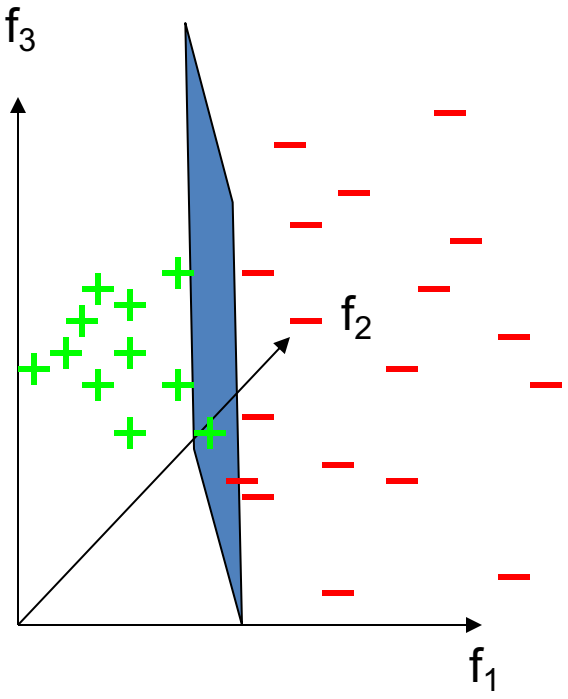
- Higher dimensions! And transformations! And robustness!



What if space is not linearly separable?

- Higher dimensions! And transformations! And robustness!
- Choose $f_1=x_1^2$, $f_2=x_2^2$, $f_3=\sqrt{2}\,x_1x_2$





The kernel trick

- **Cover's Theorem**: A complex classification problem, cast in a high-dimensional space nonlinearly, is more likely to be linear separable than in the low-dimensional input space
- SVM for non-linear classification
  - Nonlinear mapping of the input space into a high-dimensional feature space
  - Constructing the optimal hyperplane for the feature space

The kernel trick

- Replace $Q(\alpha) = \sum_i \alpha_i - \frac{1}{2}\sum_i \sum_j \alpha_i \alpha_j d_i d_j \mathbf{x}_i{}^T \mathbf{x}_j$ with

$$Q(\alpha) = \sum_i \alpha_i - \frac{1}{2}\sum_i \sum_j \alpha_i \alpha_j d_i d_j \underline{K(\mathbf{x_i}, \mathbf{x_j})}$$

Kernel function

  - Notice we can't necessarily precompute the decision boundary anymore; but since alphas are sparse, classification can still be efficient

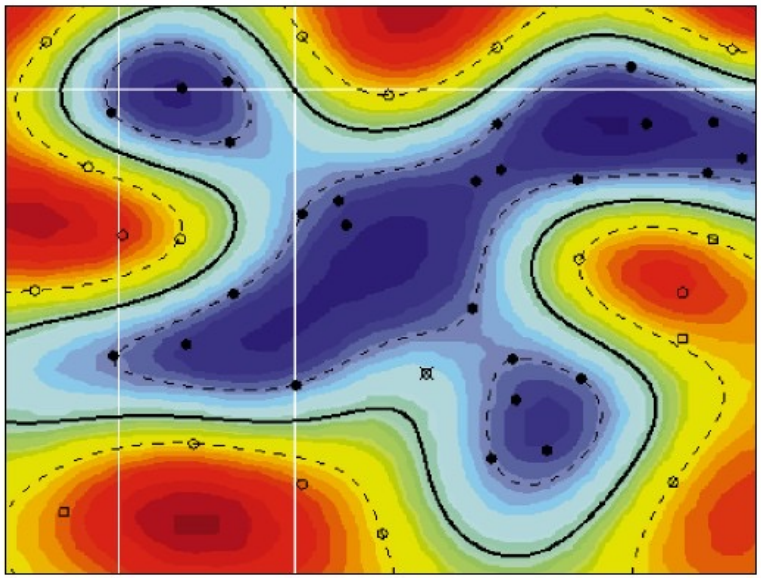
  - Kernels of the form $K(\mathbf{x_i}, \mathbf{x_j}) = f(\mathbf{x_i}) \cdot f(\mathbf{x_j})$, where f() is a feature embedding function, work out particularly nicely

Kernel Functions

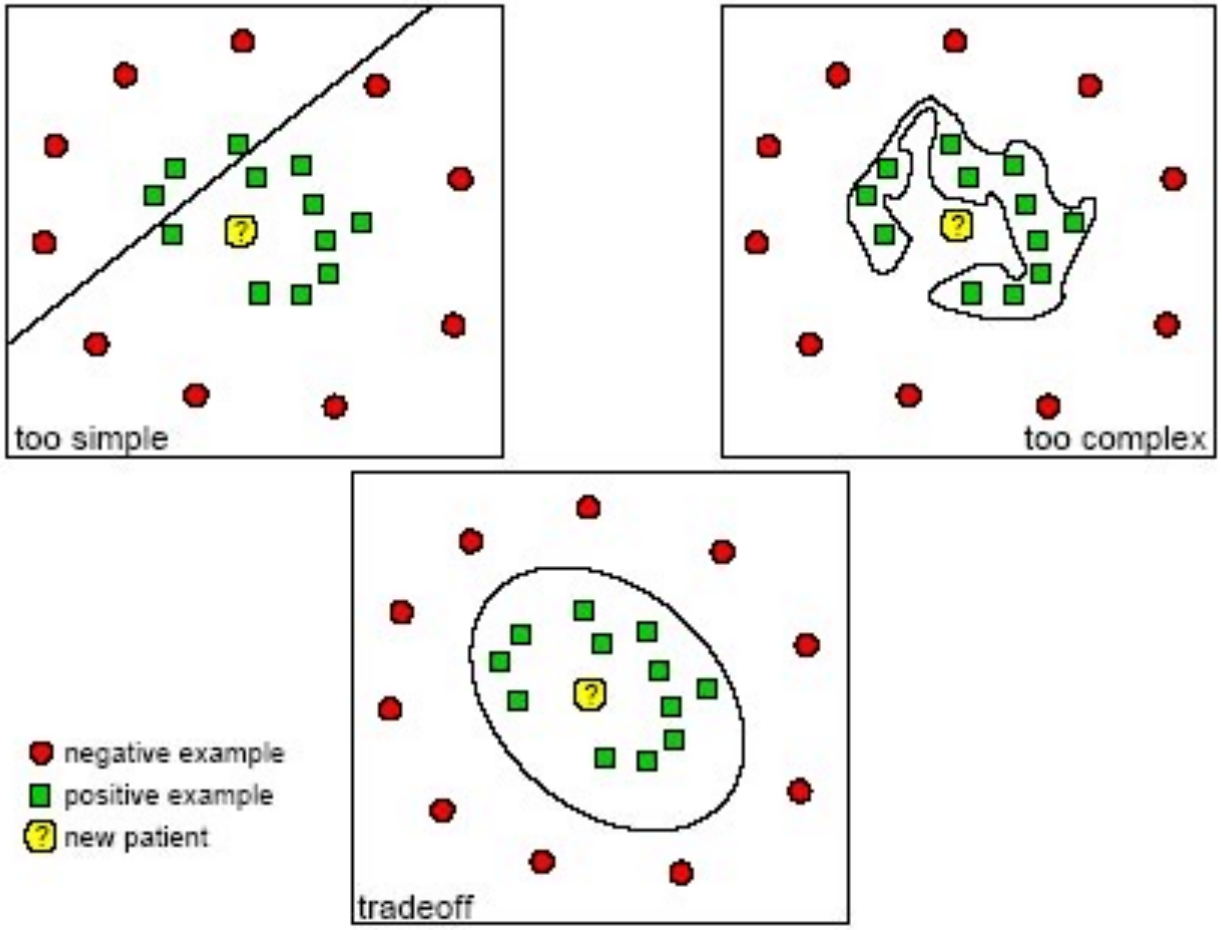
- Can implicitly compute a feature mapping to a high dimensional space, without having to construct the features! (all you need is the dot product)
- Example: $K(\mathbf{a},\mathbf{b}) = (\mathbf{a}\bullet\mathbf{b})^2$
  - $(a_1b_1 + a_2b_2)^2$
    $= a_1^2b_1^2 + 2a_1b_1a_2b_2 + a_2^2b_2^2$
    $= [a_1^2, a_2^2, \sqrt{2}a_1a_2]\bullet[b_1^2, b_2^2, \sqrt{2}b_1b_2]$

Types of Kernel

- Polynomial $K(a,b) = (a^T b+1)^d$
- Gaussian $K(a,b) = \exp(-||a-b||^2/s^2)$
- Sigmoid, etc…
- Decision boundaries in feature space may be highly curved in original space!

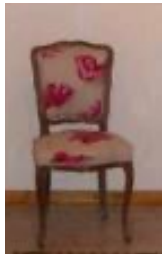


Overfitting / underfitting



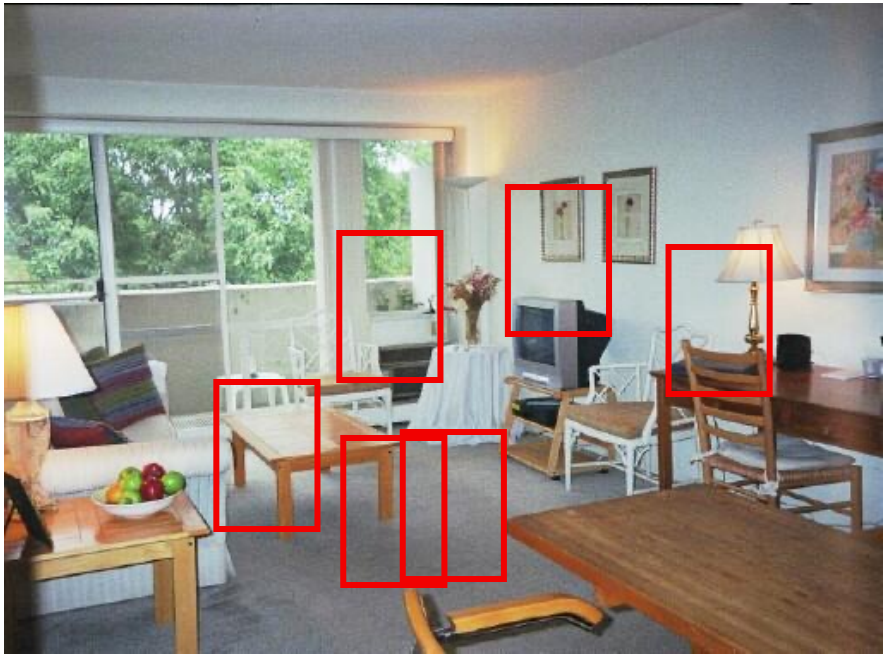
SVM practicalities

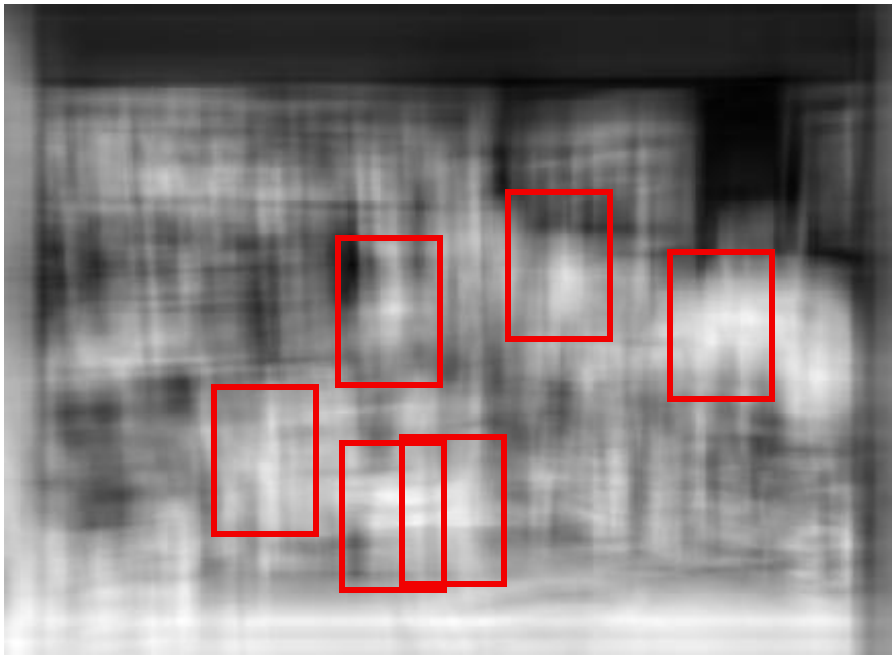
- SVMs often have very good performance
  - E.g., digit classification, face recognition, etc
- Still need parameter tweaking
  - Kernel type
  - Kernel parameters
  - Regularization weight
- Fast optimization for medium datasets (~100k)
- Off-the-shelf libraries
  - E.g. SVM[light]



Example: Object recognition

Find the chair in this image



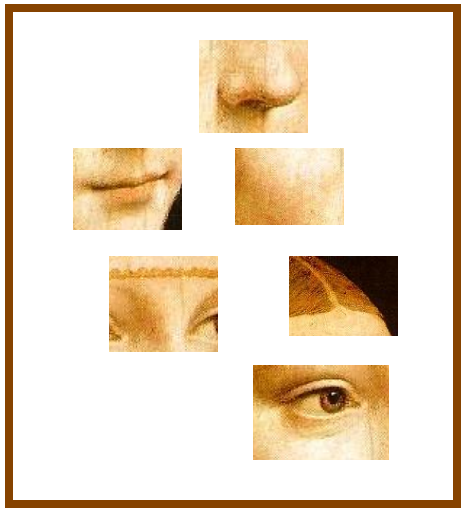


Pretty much garbage

Simple template matching is not going to suffice

Bag of features: outline

1. Extract features



Bag of features: outline

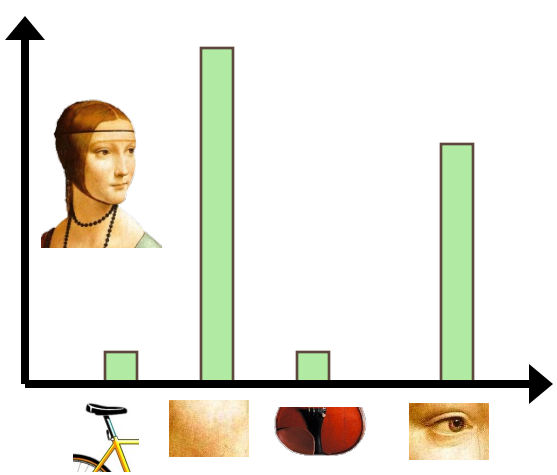
1. Extract features
2. Learn “visual vocabulary”

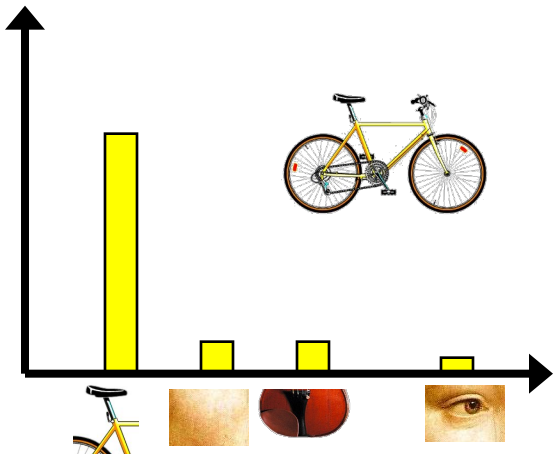
Bag of features: outline

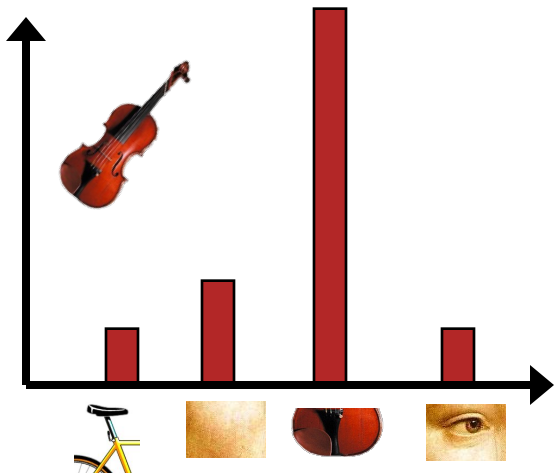
1. Extract features
2. Learn “visual vocabulary”
3. Quantize features using visual vocabulary

Bag of features: outline

1. Extract features
2. Learn “visual vocabulary”
3. Quantize features using visual vocabulary
4. Represent images by frequencies of “visual words”

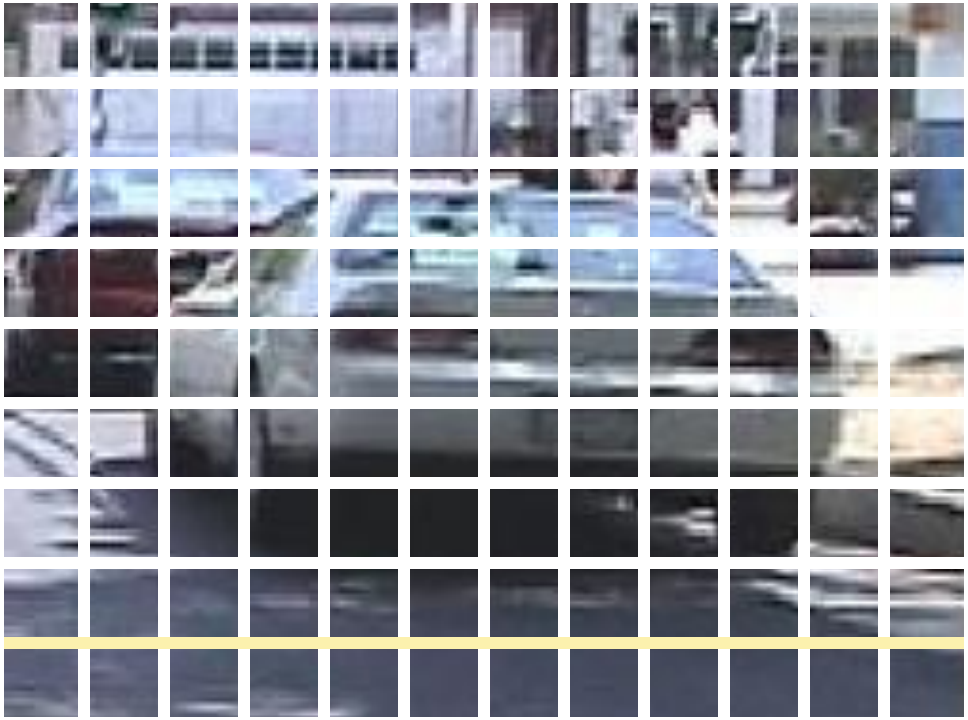






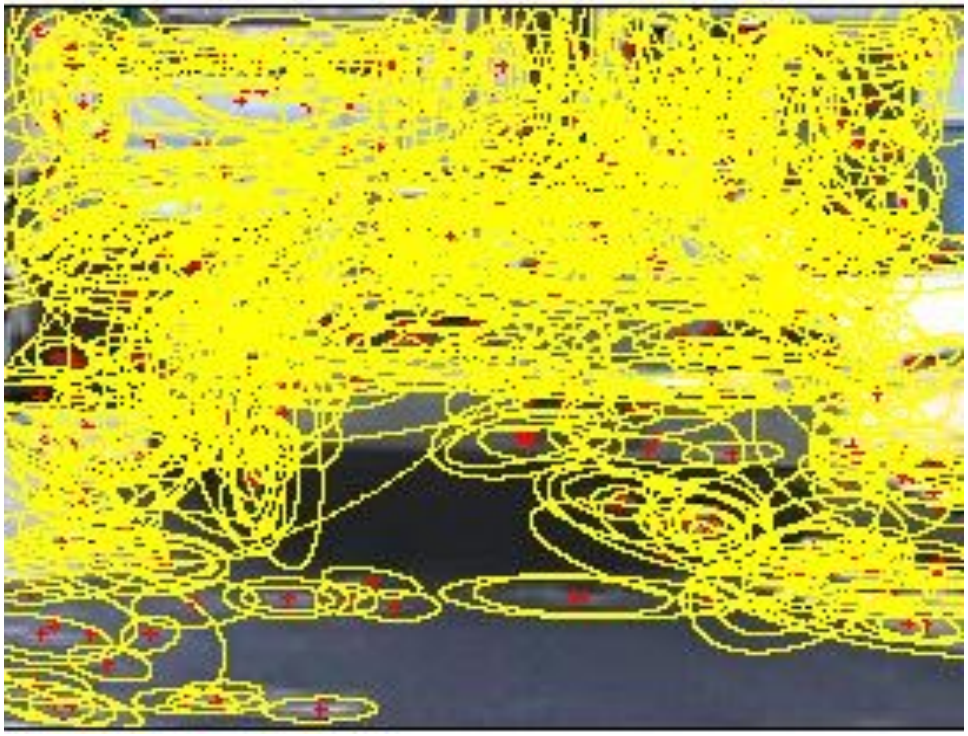
1. Feature extraction

- Regular grid
  - Vogel & Schiele, 2003
  - Fei-Fei & Perona, 2005



1. Feature extraction

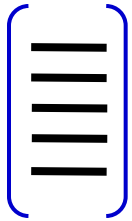
- Regular grid
  - Vogel & Schiele, 2003
  - Fei-Fei & Perona, 2005
- Interest point detector
  - Csurka et al. 2004
  - Fei-Fei & Perona, 2005
  - Sivic et al. 2005



1. Feature extraction

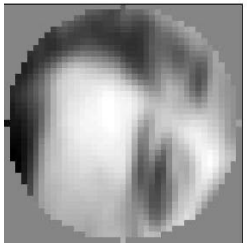
- Regular grid
  - Vogel & Schiele, 2003
  - Fei-Fei & Perona, 2005
- Interest point detector
  - Csurka et al. 2004
  - Fei-Fei & Perona, 2005
  - Sivic et al. 2005
- Other methods
  - Random sampling (Vidal-Naquet, 2002, Crandall 2006)
  - Segmentation-based patches (Barnard et al. 2003)

1. Feature extraction

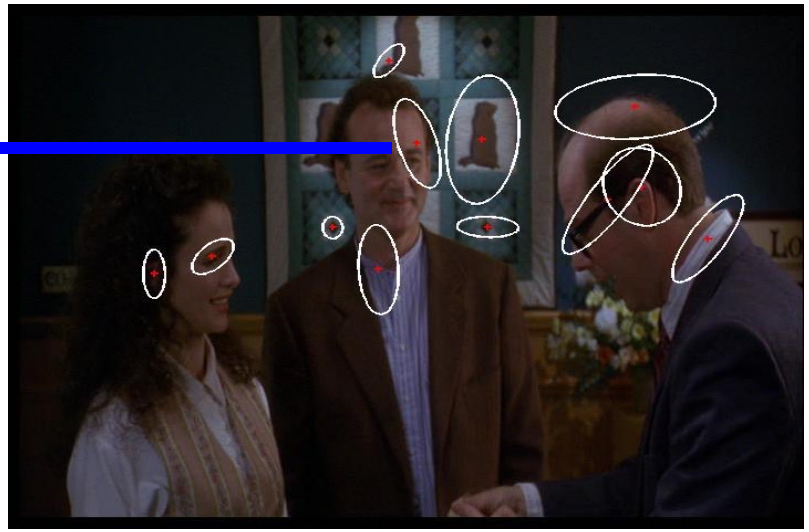


Compute SIFT descriptor

[Lowe'99]



Normalize patch



Detect patches

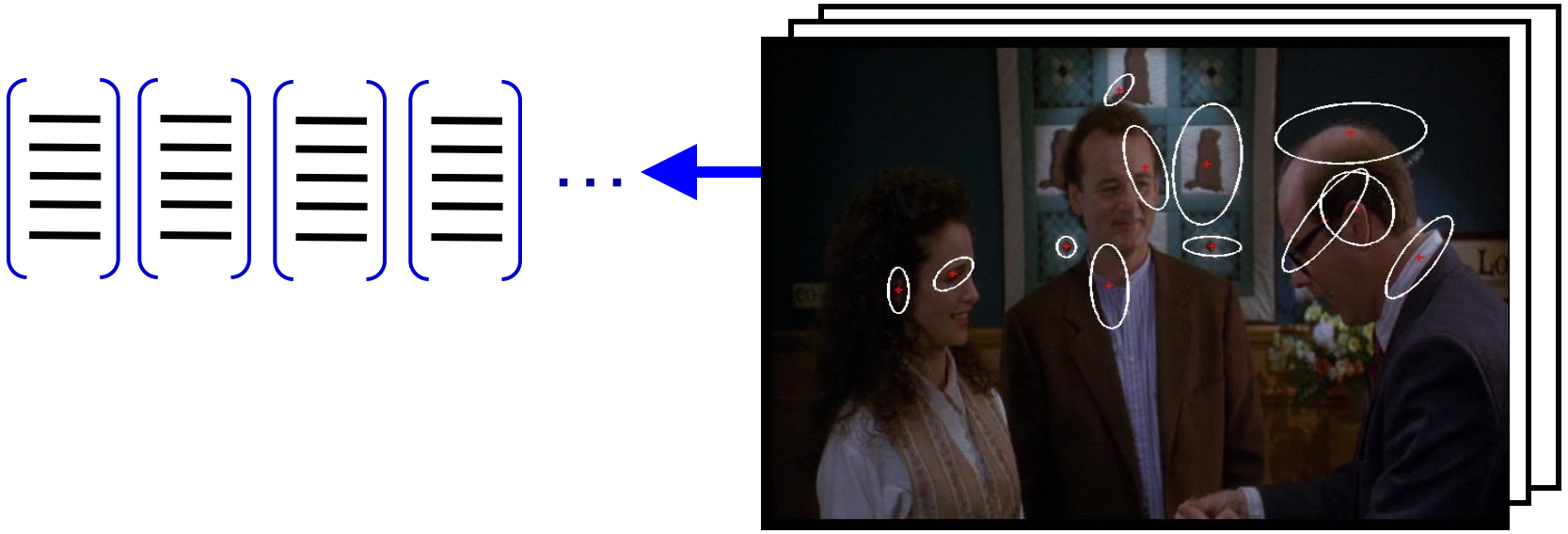
[Mikojaczyk and Schmid '02]

[Mata, Chum, Urban & Pajdla, '02]

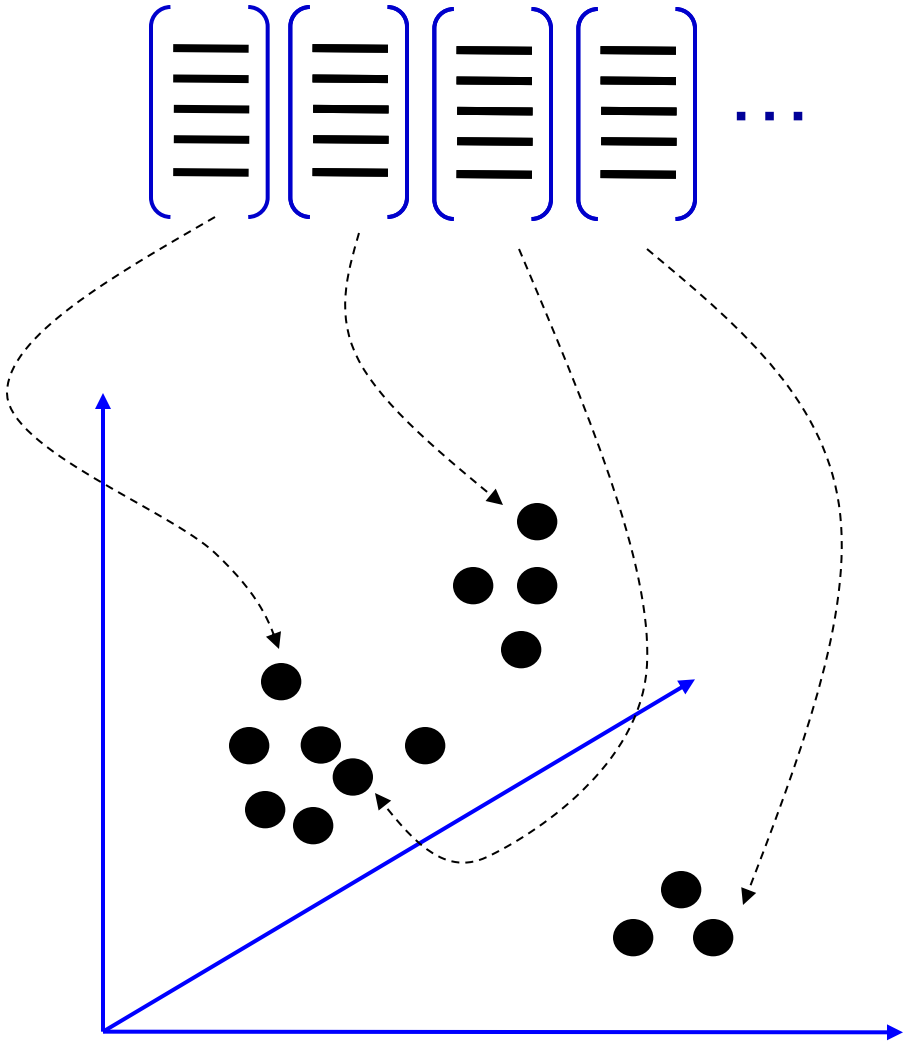
[Sivic & Zisserman, '03]

Slide credit: Josef Sivic

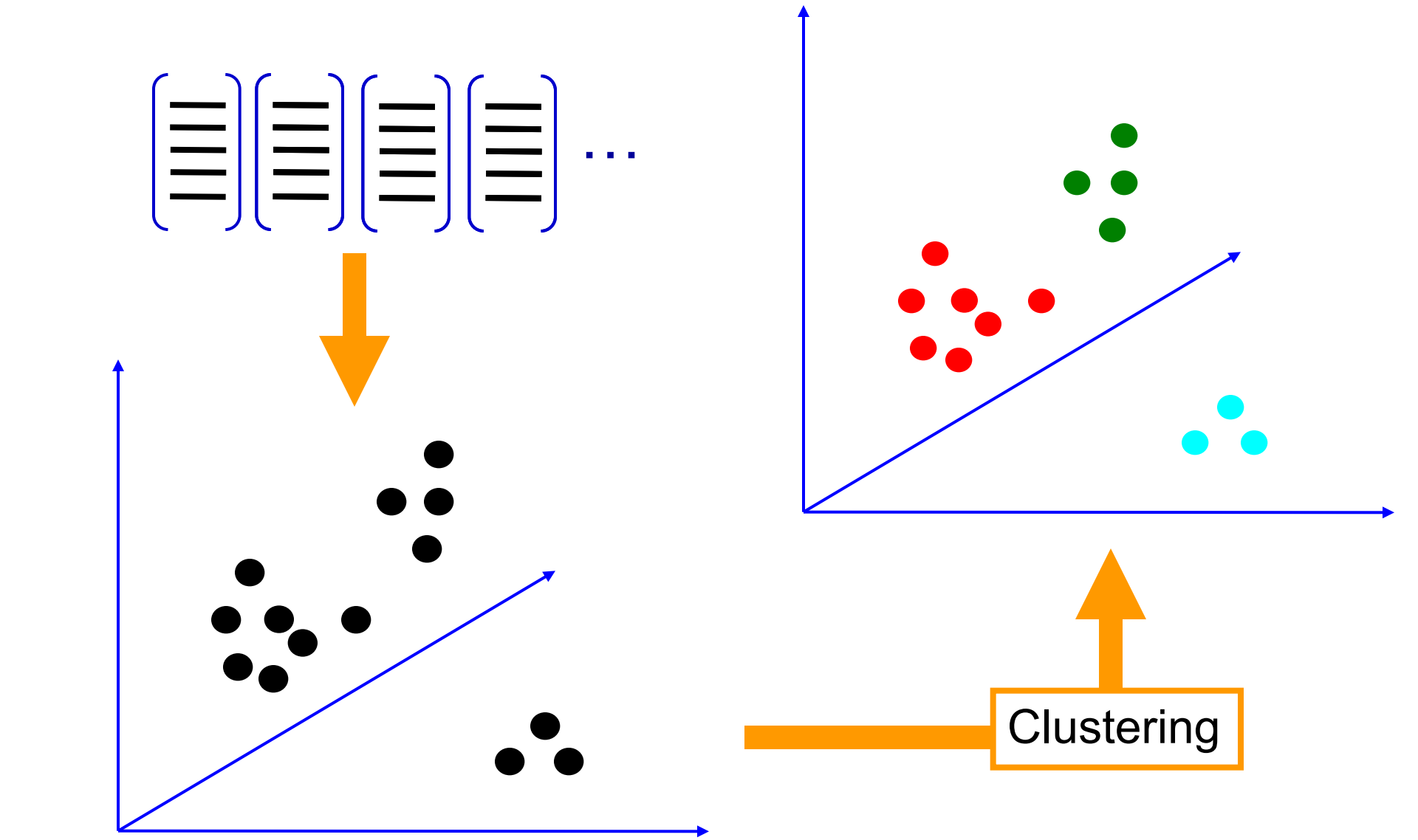
1. Feature extraction



2. Learning the visual vocabulary

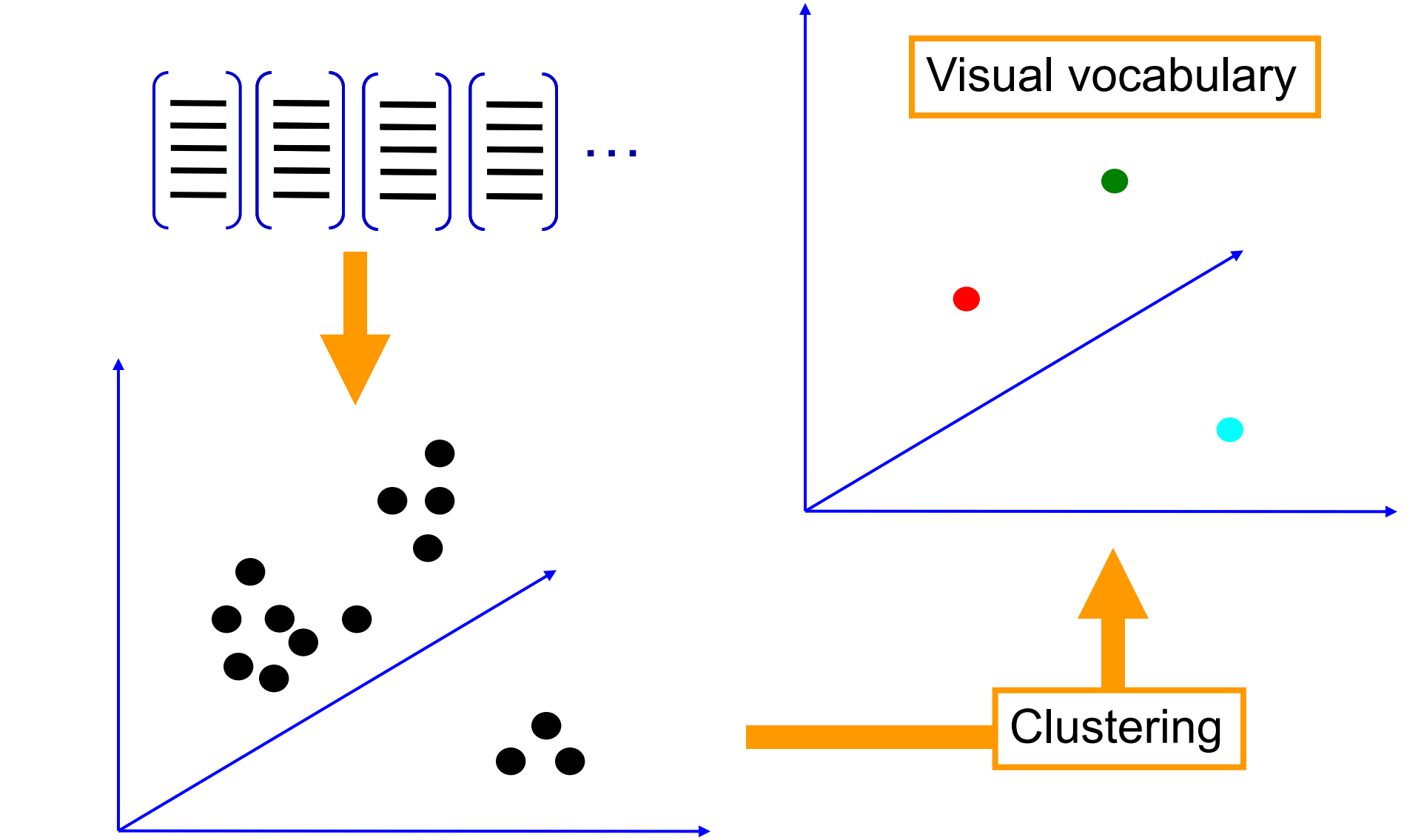


2. Learning the visual vocabulary



Slide credit: Josef Sivic

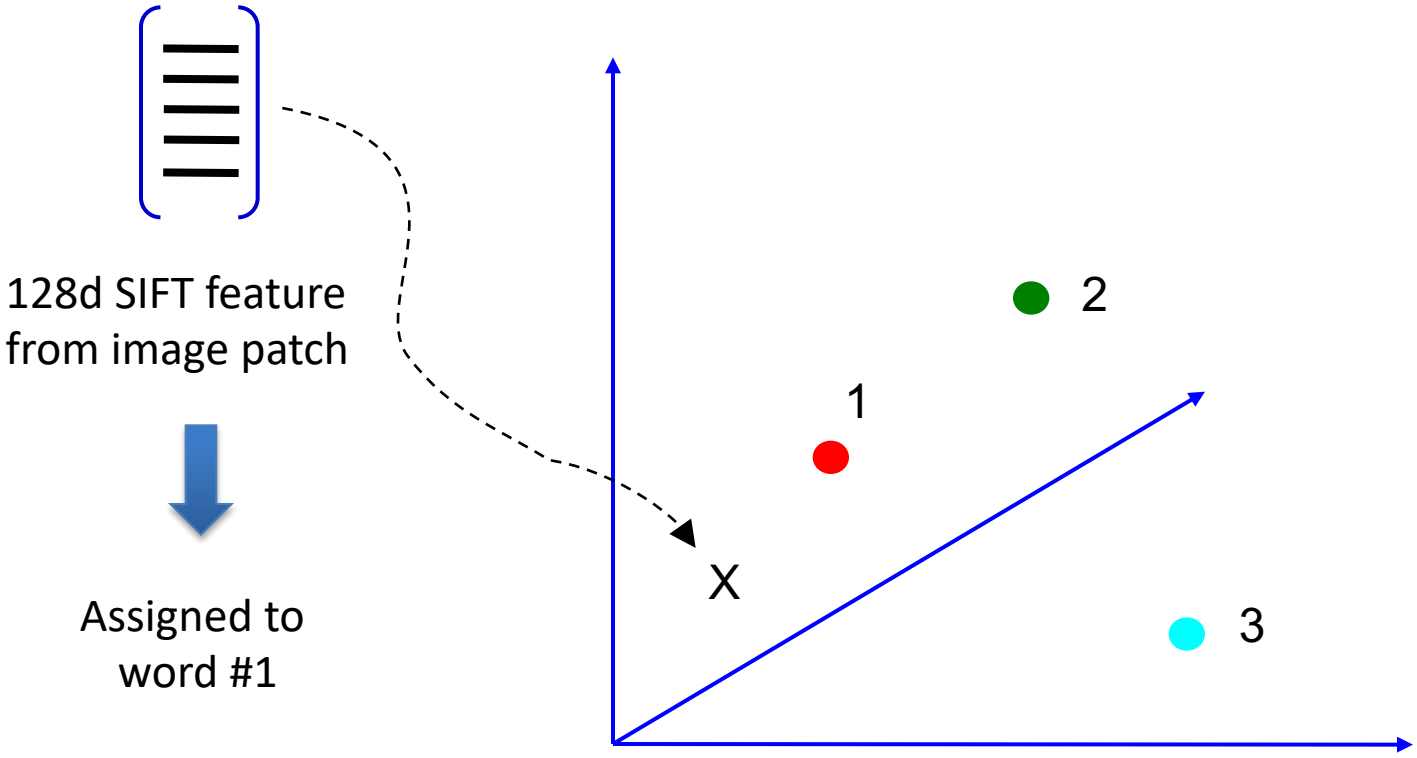
2. Learning the visual vocabulary



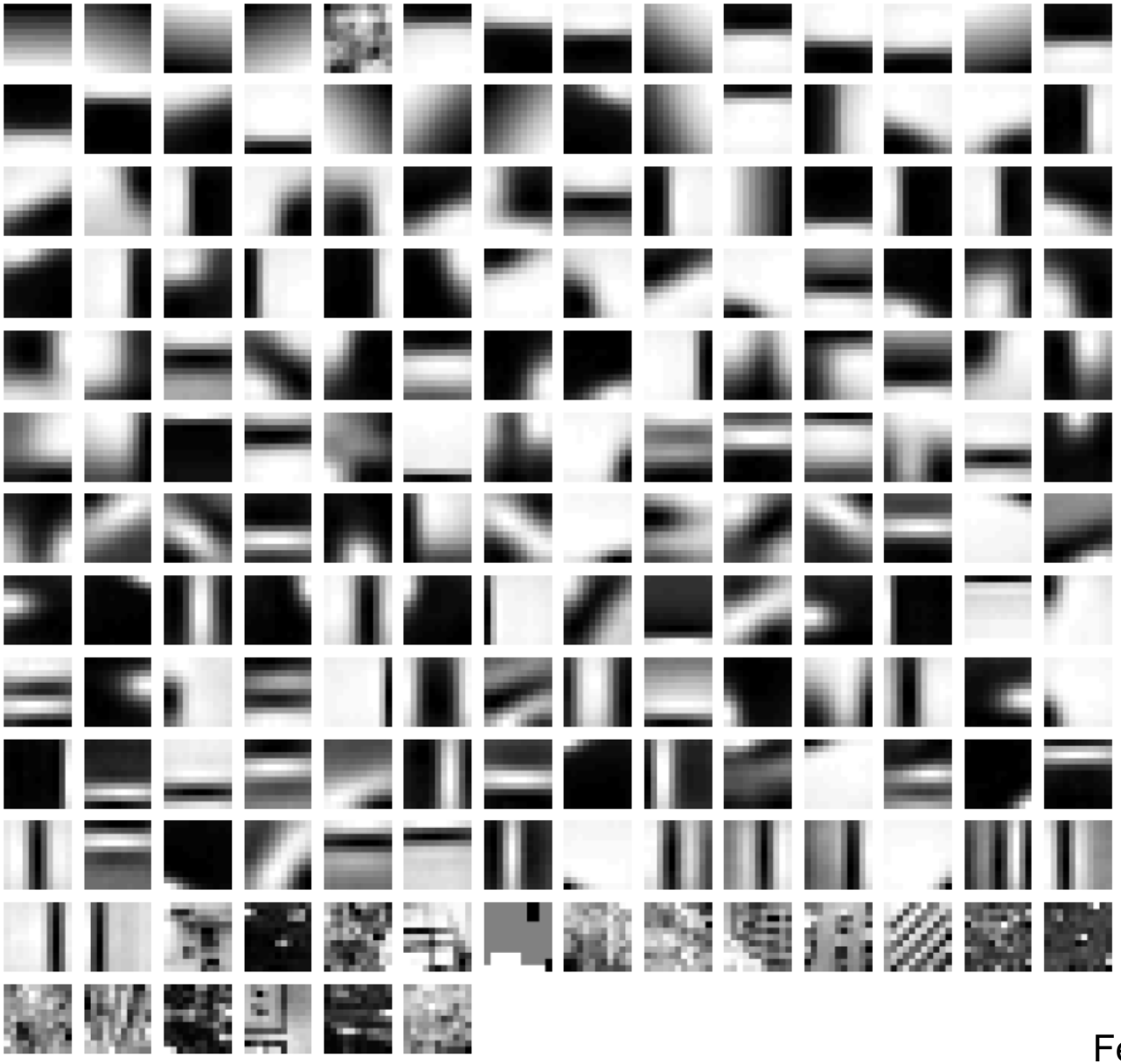
Slide credit: Josef Sivic

Vector quantization

- Given a new feature vector, find the closest codevector, and assign it the nearest cluster centroid

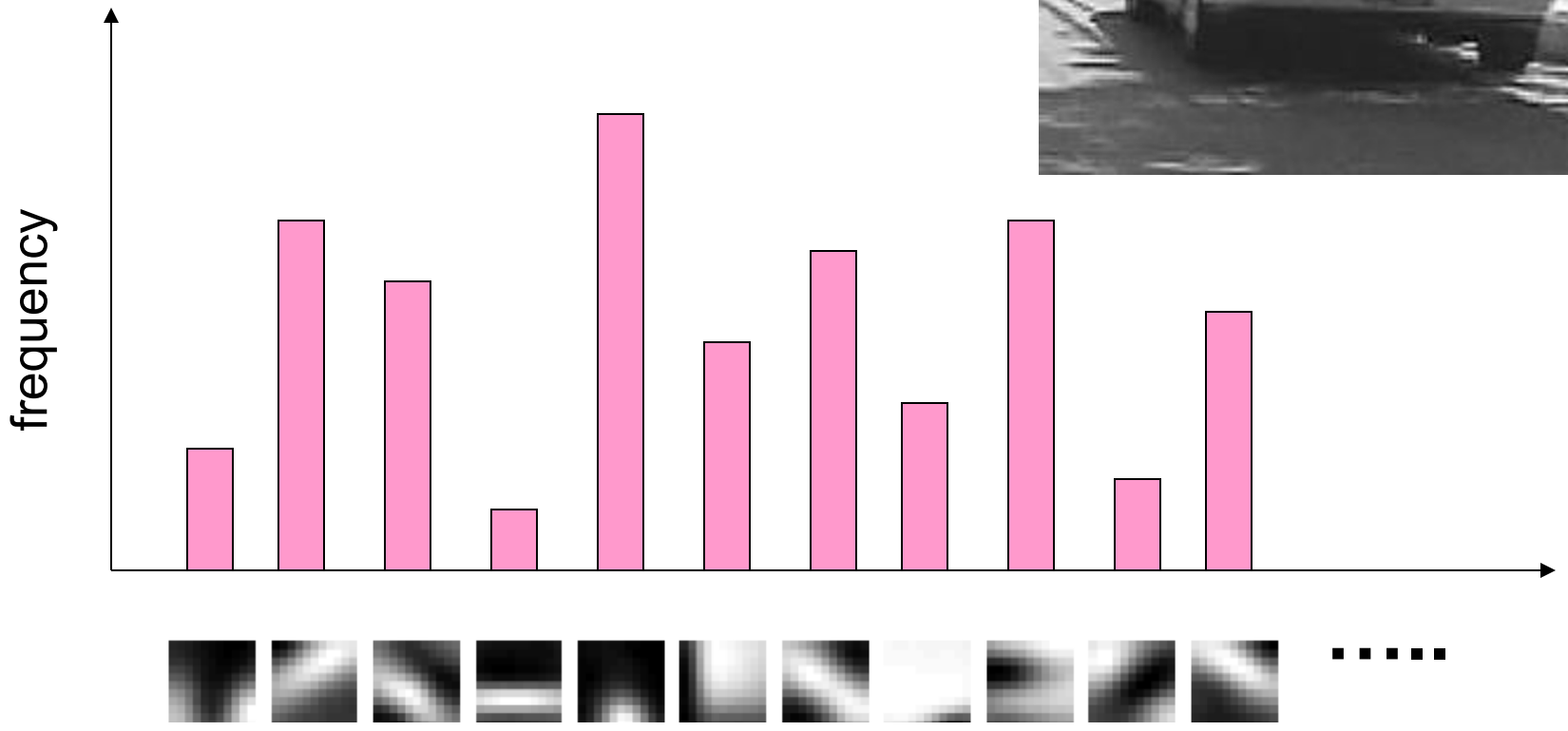


Example visual vocabulary



Fei-Fei et al. 2005

3. Image representation



4. Learning a classifier

- SVMs are de facto standard technique
  - RBF kernel usually works slightly (2-3%) better, but at significantly higher computational cost
  - Why such a modest improvement?

Secret of SVM popularity

- At their core, SVMs are simple linear classifiers, but:
  - Soft-margin learning allows for outliers – not “brittle” like perceptron learning
  - Well-principled, easy to understand learning – less likely to fall into local minima
  - Many applications have high dimensional feature spaces with redundancy – linear classification works okay
  - Fast – classification is just a dot product, basic learning also fast (but gets complicated with kernels, slack variables, huge datasets, high dimensionality)
  - Kernel trick – allows trade-off between strength of model and danger of overfitting

Next class

- Convolutional neural networks