Machine Learning

– winter term 2016/17 –

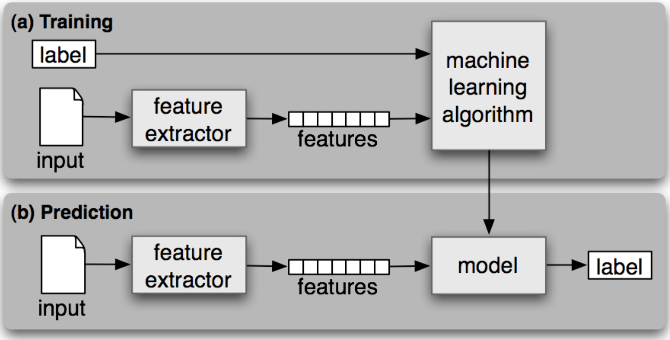
Chapter 03: Features

Prof. Adrian Ulges
Masters "Computer Science"
DCSM Department
University of Applied Sciences RheinMain

Outline

1. Feature Properties
2. Three Basic Techniques
3. Features for Images: Filters
4. Features for Images: Local Features
5. Features for Text

Introduction image from [?]



- Often, feature extraction is **more important** to the success of an ML system than the model/classifier!

We will

- ... discuss some generic properties of *good features*.
- ... present *three basic techniques* in feature engineering.
- ... look at some features for *images* and *text*.

Features

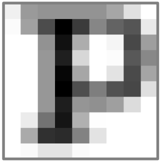
- Features are formal representations of **real-world objects**
- We can think of them as attribute-value pairs

```
color = silver,       // categorial feature
rating = ****,        // ordinal feature
mileage = 20.8,       // numerical feature
price development = (34.457, 32.189, 29.745)
                      // vector feature
```

- The term **"feature"** can refer to a **single value** (such as *color* or *mileage*), or to the object's whole **feature vector**.
- Feature vectors are often **high-dimensional** (like the pixels of an image or the terms of a text)
- In the following, we will discuss mostly **numerical** features *(we can turn all features into numerical ones by histogramming + one-hot encoding)*

Features: Example

- Use the **raw data** as features? $\rightarrow$ Example: OCR



(240, 154, 147, …,
251, 161, 76, …,
…
… , 254)

10x9 pixels = 900-dimensional feature vector

- In many cases we can do better: **Cherrypicking the 'right' features** makes the classifier's job easier.

Remarks

- Modern **deep learners** *(later)* tend to operate on the raw data and **learn** their features themselves.

Key Question

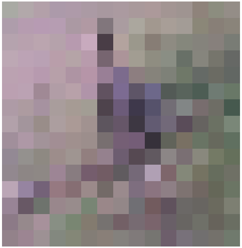
What are properties of good features?

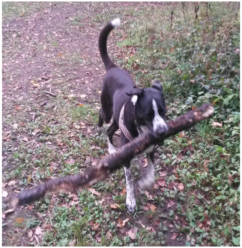
Objective 1: Compactness

"Feature extraction is a special form of dimensionality reduction"

(en.wikipedia.org)

- We require features to be **compact**
  - ... for efficiency reasons
  - ... for accuracy reasons *(curse of dimensionality)*
- Example: Using raw pixel values is **inefficient**
  (3 megapixels = 3,000,000 features → subsample the image)
- We will address other forms of **dimensionality reduction** later





Objective 2: Invariance

Invariance in Computer Science

- An invariant is a property that always evaluates to the **same value**, *before and after* applying a sequence of operations

```
int x := 10;
{x==10}
foo(x);
{x==10}
```

- Invariants are used to prove the absence of side effects and the correctness of algorithms

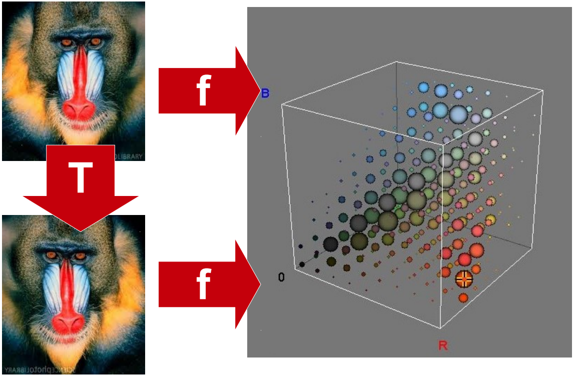
Invariance of Features in ML

- ... is basically the same: We call a feature f **invariant** (or *robust*) with respect to a transformation T if the feature **does not change** (or does not change *significantly*) **when transforming** the input object:

$$f(T(\mathbf{x})) = f(\mathbf{x}) \qquad \left(\text{or } f(T(\mathbf{x})) \approx f(\mathbf{x})\right)$$

Invariance: Example image from [?]

The feature “color histogram” is invariant with respect to flipping the input image



Invariance: Sample Transformations T images from [?] [?]

In machine learning, we want to be invariant to lots of transformations

Machine Learning on Images

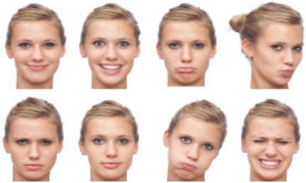
- illumination
- perspective, pose
- geometric transformations
- noise, compression artifacts

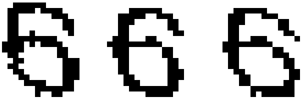
Machine Learning on Text

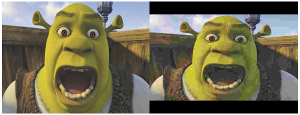
- language
- wording (synonyms)

Other Machine Learning

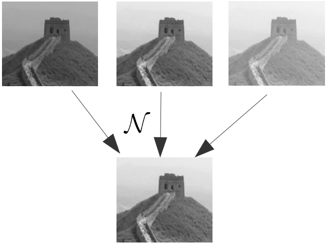
- inflation (in credit scoring)
- user rating level
  (in recommenders)
- ...







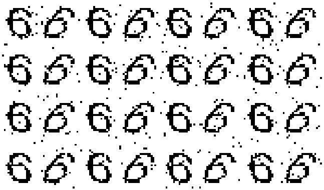
Strategies to achieve Invariance



Approach 1: Normalization

- Correct for the effect of T by normalizing
- Example: normalize for inflation
- Example: brightness normalization

Approach 2: Virtual Samples



- Generate extra training samples by applying T to the existing ones
- Example: OCR training samples
- Example: Kinekt body pose recognition

Approach 3: Integration

- Apply all possible variations of T to the input object and average the resulting features

$$f^{inv}(\mathbf{x}) = \frac{1}{|\mathcal{T}|} \int_{T \in \mathcal{T}} f(T(\mathbf{x}))\, dT$$

Objective 3: Discriminativity

- Features should be **discriminative**: They should allow us to distinguish objects from different classes
- Discriminativity and invariance are often **hard to combine**
- **Example** *(maximal invariance, minimal discriminativity)*

$$f(\mathbf{x}) = 42 \quad \forall \mathbf{x}$$

- **Example** *(should we go for invariance or discriminativity?)*

$f($  $) = f($  $)?$

Key Questions

- How do we find features that are **both robust/invariant and discriminative**?
- With respect to **which transformations T** should we be robust?
- Are all transformations **equally likely**?

Outline

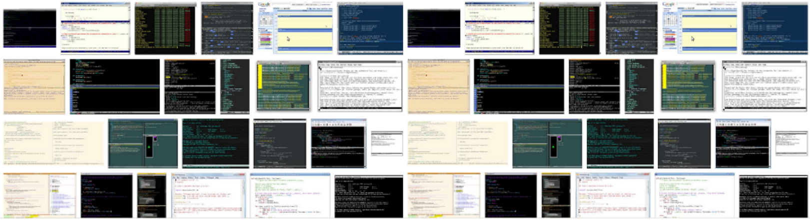
Features: Three Basic Techniques

1. Feature Selection
   - **Uninformative** features make ML problems **harder**.
2. Feature Normalization
   - Features should not **dominate**.
3. Feature Transformation
   - There is an interdependency between **features** and **ML model**.

1. Uninformative Features are Harmful

- ML problems are often **high-dimensional**
  (with hundreds or thousands of features)
- Which features are **informative** for our problem?
  (we will try to learn them $\rightarrow$ later)

Example: Nearest Neighbors $\rightarrow$ the "curse of dimensionality"



1. Uninformative Features are Harmful

Conclusions

- (NN-)classification in high dimensions becomes difficult ...
- ... if most of the dimensions contain just **noise**
  (leading to noise in the computed distances)

Remark

- The same holds for **all** classifiers: Uninformative features cause **Overfitting**!

Example: Titanic Dataset

- Decision tree accuracy (5-fold-crossvalidation on training set)
- We add uninformative features $\sim U[0, 1]$ to the data
- We set `max_depth=10` (the effect grows with `max_depth`)

# noise features	0	10	100
accuracy (%)	77.6	73.7	72.5

2. Features should not Dominate

Nearest Neighbors (NN)

- Will NN-classification work in this example?
- **Problem**: The feature "price" dominates the similarity measure!

	price	mile-age	eco-friendly
Prof. Ulges' car	14.000	5,6	1
Prof. Ulges' wives' car	70.000	11,2	0
Prof. Ulges' wives' 2. car	80.000	6,9	?

Approach: Feature Normalization

- Let $x_1, ..., x_n$ be a features' (sorted) values in the training data
- **Approach 1: Min-Max-Normalization**

$$x_i' = (x_i - x_1)/(x_n - x_1) \quad \in \quad [0,1]$$

- **Approach 2: Standardization**

$$x_i' = (x_i - \bar{x})/s \qquad // \text{ with mean } \bar{x} \text{ and standard deviation } s$$

Approach 3: Whitening

Let $X_1, ..., X_d$ be normally distributed random variables.
We subsume them to a **random vector** $\mathbf{X} := (X_1, ..., X_d)$.

Definition (Multivariate Normal Distribution)

Let $\mu \in \mathbb{R}^d$ be a vector and $\Sigma \in \mathbb{R}^{d \times d}$ a quadratic matrix. The distribution of a random vector $\mathbf{X} = (X_1, ..., X_d)$ with density

$$p(\mathbf{x}; \mu, \Sigma) = \frac{1}{(2\pi)^{\frac{d}{2}} |\Sigma|^{\frac{1}{2}}} \cdot e^{-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)}$$

is called the **multivariate normal distribution** $\mathcal{N}(\mathbf{x}; \mu, \Sigma)$.

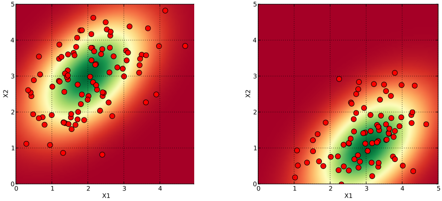
Example

Example Visualization in 2D

As an **example**, we visualize the *bivariate* (2D) normal distribution

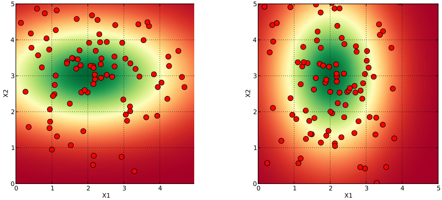
- μ is the density's maximum. Changing μ leads to a **shift** of the density.

$$\mu = (2,3) \rightarrow$$
$$\mu = (3,1)$$



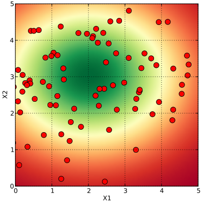
- Changing values on Σ's diagonal leads to a **re-scaling**

$$\Sigma = \begin{pmatrix} 3 & 0 \\ 0 & 1 \end{pmatrix} \rightarrow$$
$$\Sigma' = \begin{pmatrix} 1 & 0 \\ 0 & 2 \end{pmatrix}$$

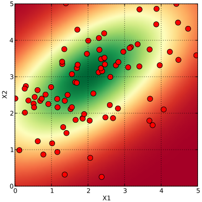


Visualization in 2D

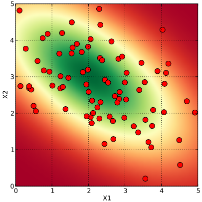
- Values Σ_{ij} off Σ's diagonal (i.e., $i \neq j$) are the **covariances** between variables X_i and X_j.
- We distinguish **three cases**:
  - $\Sigma_{ij} = 0$ (X_i and X_j are uncorrelated)
  - $\Sigma_{ij} > 0$ (positive correlation between X_i and X_j)
  - $\Sigma_{ij} < 0$ (negative correlation between X_i and X_j)



$\Sigma_{12} = 0$



$\Sigma_{12} > 0$



$\Sigma_{12} < 0$

The Multivariate Normal Distribution: Parameters

- We call μ the **center** of the distribution, and Σ its **covariance matrix**. Σ determines the distribution's **shape**.
- Σ is positive semi-definite and symmetric.
- The diagonal contains the **variances** of **X**'s dimensions:

$$\Sigma_{ii} = Var(X_i)\Big(=: \sigma_i^2\Big)$$

- In case the random variables $X_1, ..., X_d$ are **independent**, Σ is a diagonal matrix:

$$\Sigma = \begin{pmatrix} \sigma_1^2 & 0 & ... & 0 \\ 0 & \sigma_2^2 & .... & 0 \\ 0 & ... & ... & 0 \\ 0 & ... & 0 & \sigma_d^2 \end{pmatrix}$$

Feature Normalization: Whitening

Definition (Whitening Transform)

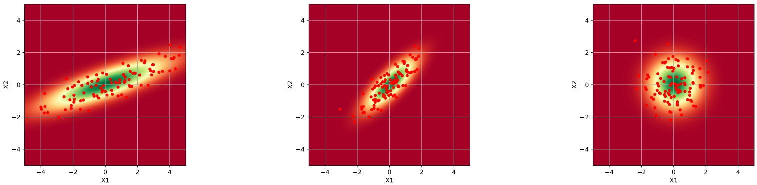
Let $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n \in \mathbb{R}^d$ *be a training set with covariance matrix* Σ *with eigenvalues* $\lambda_1, ..., \lambda_d$ *and eigenvectors* $\mathbf{p}_1, ..., \mathbf{p}_d$. *We define the* $d \times d$ *matrices*

$$D^{-\frac{1}{2}} := Diag\left(\frac{1}{\sqrt{\lambda_1}}, \frac{1}{\sqrt{\lambda_2}}, ..., \frac{1}{\sqrt{\lambda_d}}\right) \quad and \quad P = \begin{pmatrix} \mathbf{p}_1 \ \mathbf{p}_2 \ ... \ \mathbf{p}_d \end{pmatrix}$$

Then, we call the following transformation a **whitening***:*

$$\mathbf{x}' := D^{-\frac{1}{2}} \cdot P \cdot \mathbf{x}$$

Illustration



original standardized (see above) whitened

Feature Normalization: Whitening

Remarks

- The whitening transform produces data with covariance matrix I ($=$ the identity matrix):
  - the variance along each axis is 1
  - all correlations are 0 (the axes are **decorrelated**)
- A proof will follow later (see PCA)

3. Interdependency Features ⇔ Model

Food for Thought

- **Correct?** "We need to standardize when using a nearest neighbors model but not when using a decision tree."

3. Interdependency Features ⇔ Model image from [?]

Example: Online-Shop

- Goal: Predict whether a **product in your shop** will be bought
- Features

$$x_1 := \text{the product's } \textbf{price}$$
$$x_2 := \text{the product's } \textbf{average price} \text{ over other many other shops}$$

Do it Yourself

- Sketch the data in feature space.
- What works better: **decision trees** or a **linear classifier**?
- How can we resolve the problem?

3. Interdependency Features $\Leftrightarrow$ Model image from [?]

Outline

Image Features: Overview

We can view images as **signals**:

Definition (Signal (1D and 2D))

Given $M \in \mathbb{N}^+$*, we call*

$$s : \mathbb{Z} \to \{1, ..., M\}$$

a (discrete) 1D signal, and

$$s : \mathbb{Z} \times \mathbb{Z} \to \{1, ..., M\}$$

a (discrete) 2D signal.

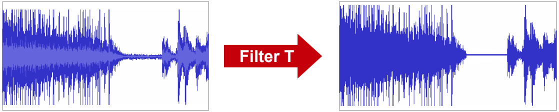
- **Examples**: digital audio signals (1D) and images (2D)
- Feature extraction for images borrows methods from **signal processing**

Signals and Filters

A **filter** is a mapping T that **transforms** a (1D or 2D) signal s into another signal s':

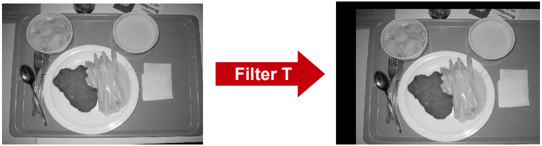
Example: Silence Detection

$$s(t) \mapsto \begin{cases} s(t) & \text{if } |s(t)| \geq c \\ 0 & \text{else} \end{cases}$$



Example: 2D Translation

$$s(x, y) \mapsto s(x + c_x, y + c_y)$$



FIR Filters

We can define many filters by **linear**, mask-based operations. These are called **finite impulse response (FIR) filters**:

Definition (FIR Filter (1D))

Let s be a (1D) signal, $M \in \mathbb{N}$ and

$$(w_{-M}, w_{-M+1}, \ldots, w_{-1}, w_0, w_1, \ldots, w_M)$$

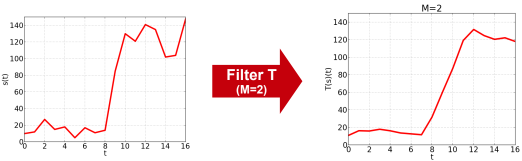
be a **filter mask**. *Then, the following filter is a* **finite impulse response filter**:

$$s(t) \mapsto \sum_{\tau=-M}^{M} s(t-\tau) \cdot w_\tau$$

Remarks

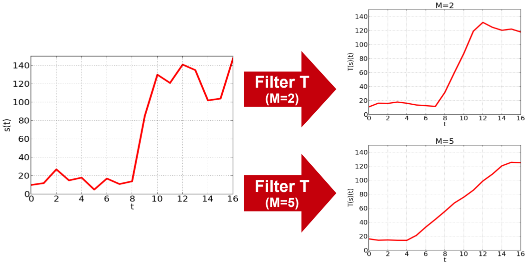
- The transformation with a FIR filter is called a **convolution**!
- The **filter mask** is also called a **kernel**.

Example 1: $(w_{-2}, w_{-1}, w_0, w_1, w_2) = (\frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5})$



Remarks

By varying the size of the mask, we obtain a stronger/weaker smoothing

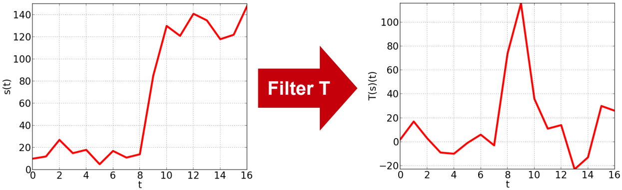


Example 2

- We define $(w_{-1}, w_0, w_1) = (1, 0, -1)$

$$s(t) \mapsto \sum_{\tau=-1}^{1} s(t-\tau) \cdot w(\tau) = s(t+1) - s(t-1)$$

- This filter approximates the signal's **derivative**



FIR Filters (2D)

We define FIR filters for 2D:

Definition (FIR Filter (2D))

Let s be a (2D) signal, $M \in \mathbb{N}$, and

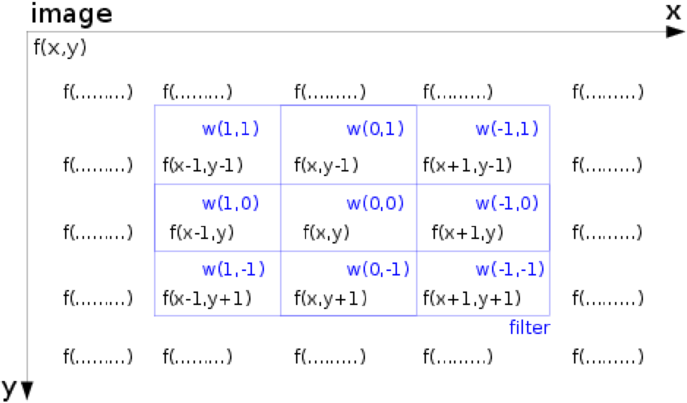
$$\begin{pmatrix} w_{-M,-M} & \dots & w_{-M,0} & \dots & w_{-M,M} \\ \dots & & & & \\ w_{0,-M} & \dots & w_{0,0} & \dots & w_{0,M} \\ \dots & & & & \\ w_{M,-M} & \dots & w_{M,0} & \dots & w_{M,M} \end{pmatrix}$$

be a **filter mask**. *Then, the following filter is a* **finite impulse response filter**:

$$s(x, y) \mapsto \sum_{\xi=-M}^{M} \sum_{\lambda=-M}^{M} s(x - \xi, y - \lambda) \cdot w_{\xi,\lambda}$$

FIR Filters (2D)

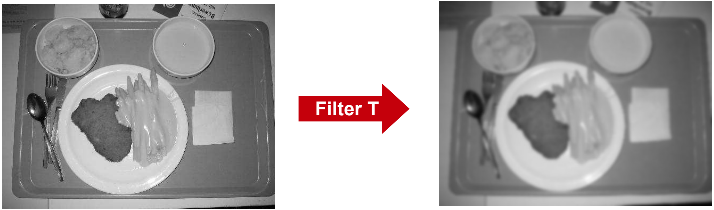
- We place the mask at every position of the image
- We compute the **weighted sum** of the pixel intensities, weighted by the mask's values



2D FIR Filters: Example 1

The **mean filter blurs** the input image

$$\begin{pmatrix} w_{-2,-2} & w_{-2,-1} & w_{-2,0} & w_{-2,1} & w_{-2,2} \\ w_{-1,-2} & w_{-1,-1} & w_{-1,0} & w_{-1,1} & w_{-1,2} \\ w_{0,-2} & w_{0,-1} & w_{0,0} & w_{0,1} & w_{0,2} \\ w_{1,-2} & w_{1,-1} & w_{1,0} & w_{1,1} & w_{1,2} \\ w_{2,-2} & w_{2,-1} & w_{2,0} & w_{2,1} & w_{2,2} \end{pmatrix} = \frac{1}{25} \cdot \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

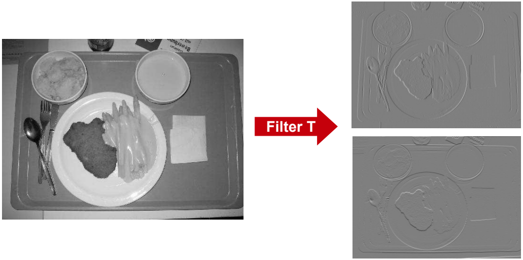


2D FIR Filters: Example 2

What do these Filters do?

$$\begin{pmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{pmatrix} \qquad \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}$$

These are the **Sobel filters**: They are commonly used to compute the partial derivatives $\frac{\partial s(x,y)}{\partial x}$, $\frac{\partial s(x,y)}{\partial y}$ of an image *(which indicate the* ***edges*** *of an image)*



Edges and the Gradient

- Edges are characterized by **abrupt changes** in intensity
- Edges have a **local orientation** in each pixel
- We can measure edges by properties of the **gradient** *(based on the images' partial derivatives)*

The **gradient**

$$\nabla s(x,y) = \left(\frac{\partial s(x,y)}{\partial x}, \frac{\partial s(x,y)}{\partial y}\right)$$

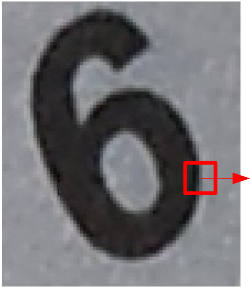
The **gradient's norm**

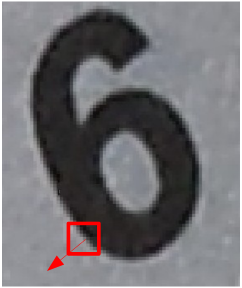
$$G = ||\nabla s(x,y)|| = \sqrt{\left(\frac{\partial s(x,y)}{\partial x}\right)^2 + \left(\frac{\partial s(x,y)}{\partial y}\right)^2}$$

The **gradient's angle**

$$\alpha = atan\left(\frac{\partial s(x,y)}{\partial x}, \frac{\partial s(x,y)}{\partial y}\right)$$

The Gradient: Example





$$\frac{\partial s(x,y)}{\partial x} \approx 50, \frac{\partial s(x,y)}{\partial y} \approx 0$$

$$||\nabla s(x,y)|| \approx \sqrt{50^2 + 0^2} = 50$$

$$\alpha = atan(0, 50) = 0^\circ$$

$$\frac{\partial s(x,y)}{\partial x} \approx -40, \frac{\partial s(x,y)}{\partial y} \approx 40$$

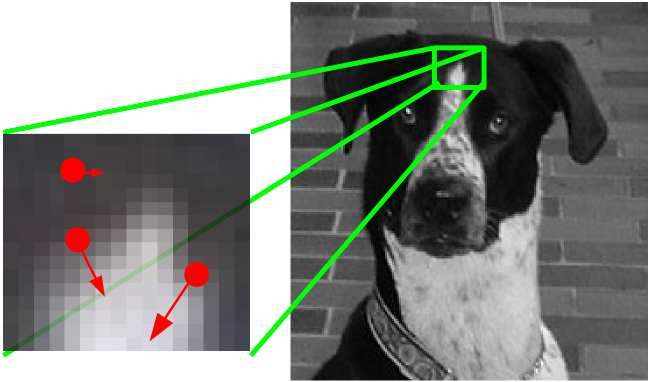
$$||\nabla s(x,y)|| \approx \sqrt{(-40)^2 + 40^2} \approx 56$$

$$\alpha = atan(-40, 40) = -135^\circ$$

The Gradient: Properties

Remarks

- The gradient always points into the direction of the **strongest increase in intensity**
- The gradient's norm $||s(x, y)||$ corresponds to the **strength of the edge**



Outline

Local Features: Motivation

- Often, we are interested only in a certain **part** of the image
- **Example**: Object recognition

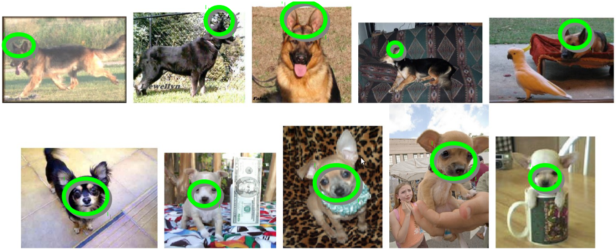
Challenges

The object's representation changes due to...

- illumination *(brightness, position of light source, shadows, ...)*
- camera position and object pose
- occlusion and background *(also called "clutter")*
- Variations of objects within the class *("intra-class variaton")*

Local Features: Motivation[?]

Key Idea: Even when images from the same class are not **globally** similar, they share certain **local characteristics**



Approach 1: Hand-engineer Local Features (here)

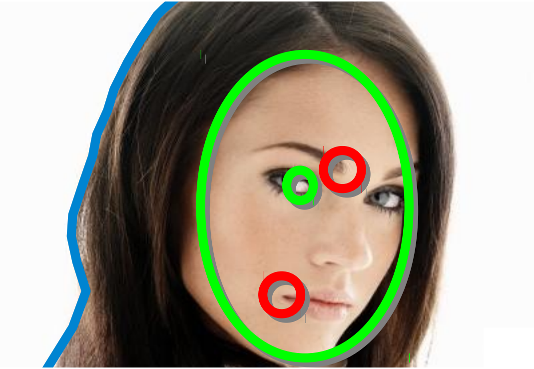
- state-of-the-art until 2011 *(and still used frequently today)*
- **SIFT**, SURF, HoG, Canny, ORB, ...

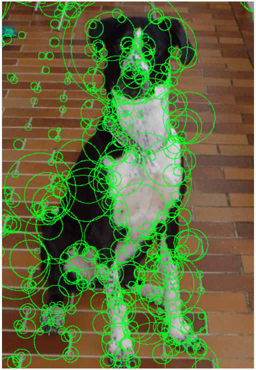
Approach 2: Learn Local Features (later)

- state-of-the-art since 2011
- Convolutional Neural Networks (CNNs) → later

Some (Hand-engineered) Local Features image from [?]

- edges
- corners
- blobs (here)





Local Features: Matching image from [?]

After extracting local features, we *match* them to recognize objects

Local Features: Matching (Formalization)

Simple Matching Algorithm

1. For each training image, **detect local features** and describe them by **local feature vectors**, $f_1, .., f_n$
2. Do the same for the **test image**, obtaining local feature vectors $f'_1, .., f'_m$
3. **Matching**: Perform a **nearest neighbor search**, i.e. for each test feature, f'_j, find the closest training feature vector $f_{nn(j)}$
4. **Reasoning**: Based on the resulting matches, make a decision (for example, which object is visible)

Remarks

- There are lots of improvements *(speed-up of neighbor search, plausibility checks by feature positions, ...)*
- **Key Questions**
  - How do we **detect** local regions of interest?
  - How do we **describe** their appearance?

Local Feature Detection: The DoG-Filter

Definition (The DoG Filter)

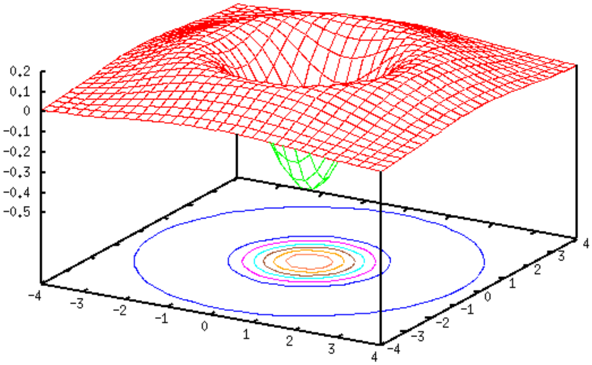
Let σ_1, σ_2 be standard deviations with $\sigma_1 > \sigma_2$. Then, we call the 2D filter with

$$w_{\xi,\lambda} = \underbrace{\frac{1}{\sigma_1 \cdot \sqrt{2\pi}} exp\Big(- \frac{\xi^2 + \lambda^2}{2\sigma_1^2}\Big)}_{\mathcal{N}(\xi,\lambda;\sigma_1)} - \underbrace{\frac{1}{\sigma_2 \cdot \sqrt{2\pi}} exp\Big(- \frac{\xi^2 + \lambda^2}{2\sigma_2^2}\Big)}_{\mathcal{N}(\xi,\lambda;\sigma_2)}$$

a **Difference-of-Gaussians (DoG)** *filter.*

What does this filter do?

The DoG-Filter: Illustration image from [?]



- The DoG filter approximates the so-called **Mexican Hat** (aka "Laplacian-of-Gaussians") operator
- The DoG filter detects **blobs** *(dark regions surrounded by a bright background)*

DoG Detection Algorithm

- We choose parameters σ_2, σ_1 (often, $\sigma_1 \approx 1.6 \cdot \sigma_2$)
- We filter the image with the resulting DoG filter
- We obtain a **response image** $r(x, y)$
- We choose local **extrema** (maxima and minima) of the response (where $|r(x, y)| > t$)
- These are the **centers** of our **blobs**!

Question

- How do we choose the **size** of the blobs to detect?

Feature Detection: Scale Invariance

- Modern feature detectors come with a free **scale parameter**
- For DoG: the scale σ_2 *(from which we compute* σ_1*)*
- This parameter determines if our detector localizes **fine, small** structures or **coarse, wide-spread** structures

$\sigma_2 = 0.1$

$\sigma_2 = 1.1$

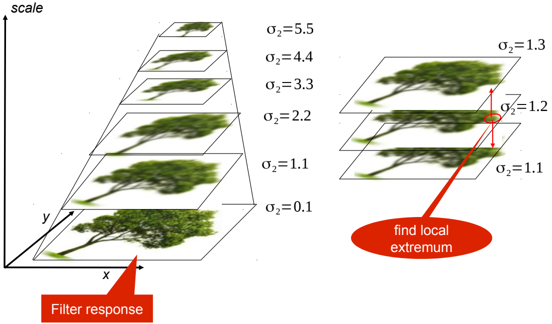
$\sigma_2 = 2.2$

$\sigma_2 = 3.3$

$\sigma_2 = 4.4$

$\sigma_2 = 5.5$

Scale-invariant Feature Detection: Approach



- We **repeat** detection on **multiple scales** (by varying σ_2)
- Algorithmically, instead of increasing σ_2, we can just **scale down** the input image and keep σ_2 fixed
- Instead of a single two-dimensional response image, we obtain a **pyramid** of response images, the **scale space**
- We now search the pyramid for **local extrema** of the filter response

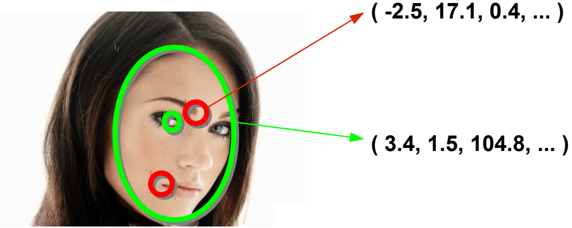
Scale-invariant Features: Example

Local Features: Feature Description

Question 1: Feature Detection ✓

Question 2: Feature Description

- What **feature extraction** do we apply to describe the **appearance of local regions**?



- In the following: One very popular approach called **SIFT** *(Scale-invariant Feature Transform)* [**?**] (> 37K citations)

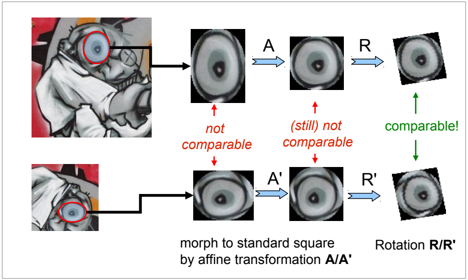
SIFT Features

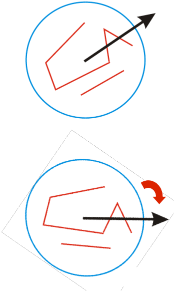
SIFT uses **two steps** to describe a local region of interest *(ROI)*

1. Region **normalization**
2. Description by **gradient histograms**

Step 1: Region Normalization

- Scale all ROIs to a standard-size square
- Determine the *dominant edge direction* α in the square
- **Rotate** the region such that $\alpha = 0^\circ$

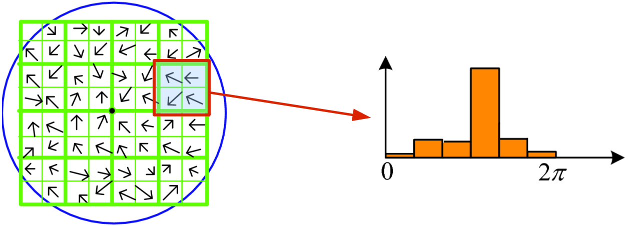




SIFT Features image from [?]

Step 2: Description by Gradient Histograms

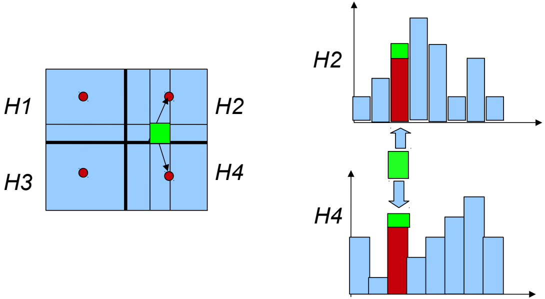
- Subdivide the (normalized) ROI into 4×4 windows
- For each window, store a **normalized histogram** of the 8 (discretized) gradient orientations
- Each pixel (x, y)'s gradient vector $\nabla s(x, y)$ adds a bit of **weight** to its direction in the histogram. The weight is determined by the **edge strength** $||\nabla s(x, y)||$!
- **Concatenate** the 4×4 histograms (each 8-dimensional) into a 128-dimensional local feature vector



SIFT Features

Step 2: One more Improvement

- So far, each pixel contributes to the histogram of **'its' subwindow**
- This is not robust to **small shifts**: Some pixels end up in a different subwindow, and the feature may change strongly
- Improvement: Each pixel contributes **a bit** to each of its 4 **neighbor subwindows**
- Weights are determined by **bilinear interpolation**



Local Features: Do-it-Yourself

We have learned of local features by DoG blob detection and gradient-based SIFT description. Those are usually simply called **SIFT Features**.
To which of these transformations are SIFT features **invariant/robust**?

- rotation
- illumination changes
- (small) translation
- non-affine distorsion

Outline

Some Remarks regarding Text Features

ML Applications involving Text

- Information extraction / part-of-speech tagging
- Sentiment analysis
- Spam filtering
- Information retrieval
- Recommendation (of news, videos, movies, jobs, ...)
- ...

Remarks

- In **this chapter**, we will have a look at some simple text features, including the common **bag-of-words** features
- The focus will still be on **simple text statistics**
- A very useful reference: Python's `nltk` module!

Text Features: Segmentation

- First Question: What is a "**term**"?
- **Text segmentation** into terms is not a trivial problem

Example	**Approach**
Germany's chancellor	*rule-based recognition*
3/20/91 vs. Mar 12, 1991	*rule-based recognition*
(0049) 611/9495-1215	*rule-based recognition*
San Francisco	*statistical methods*
Lebensversicherungsgesellschaft vs. Malerei	*compound splitter (dictionary-based vs. statistical methods)*

- *Simply Splitting at spaces is not 100% accurate but common.*

Text Segmentation image from [?]

Code Example: Python

- This code uses **regular expressions**, which allow us to search a wide range of text patterns in strings

Operator	Behavior
.	Wildcard, matches any character
^abc	Matches some pattern *abc* at the start of a string
abc$	Matches some pattern *abc* at the end of a string
[abc]	Matches one of a set of characters
[A-Z0-9]	Matches one of a range of characters
ed|ing|s	Matches one of the specified strings (disjunction)
*	Zero or more of previous item, e.g., a*, [a-z]* (also known as *Kleene Closure*)
+	One or more of previous item, e.g., a+, [a-z]+
?	Zero or one of the previous item (i.e., optional), e.g., a?, [a-z]?
{n}	Exactly *n* repeats where *n* is a non-negative integer
{n,}	At least *n* repeats
{,n}	No more than *n* repeats
{m,n}	At least *m* and no more than *n* repeats
a(b|c)+	Parentheses that indicate the scope of the operators

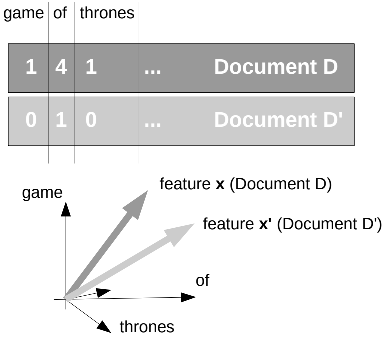
```
>>> text = 'That U.S.A. poster-print costs $12.40...'
>>> pattern = r'''(?x)    # set flag to allow verbose regexps
...     ([A-Z]\.)+        # abbreviations, e.g. U.S.A.
...   | \w+(-\w+)*        # words with optional internal hyphens
...   | \$?\d+(\.\d+)?%?  # currency and percentages, e.g. $12.40, 82%
...   | \.\.\.            # ellipsis
...   | [][.,;"'?():-_`]  # these are separate tokens
... '''
>>> nltk.regexp_tokenize(text, pattern)
['That', 'U.S.A.', 'poster-print', 'costs', '$12.40', '...']
```

Text Features: Bag-of-Words

- We define a **vocabulary** of terms $t_1, ..., t_m$
- Each document D is described by a vector $\mathbf{x} = (x_1, ..., x_m)$
- The entries x_i indicate the importance of term t_i for D
- $\mathbf{x}$ is very **sparse** (only terms appearing in D get a weight $\neq 0$)

Popular Term Weightings (more in [?], Chapter 6)

- $x_i :=$ # of occurrences of term t_i in D (*“term frequency” tf_i*)
- $x_i := log(tf_i)$
- $x_i :=$ 1 (0) if term t_i appears (not) in the document
- $x_i := tf_i$, weighted such that frequent terms get less weight *(tf-idf)*
- $x_i :=$ Okapi BM25 weights
- ...



Text Features: Normalization

- We also **normalize** text to increase robustness to flexion and sentence structure
- **Step 1**: Lower-casing *(Sometimes → sometimes)*
- **Step 2**: Stemming = reducing words to their stem

Stemming: Methods

- Rule-based Methods
  - Example rule: `*t → *` (geht → geh)
  - Example rule: `*en → *` (gehen → geh)
- Dictionary-based Methods
  - Example: `stem['ging'] = 'geh'`
  - popular for languages with strong flexion *(like German)*

Stemming: Code Example

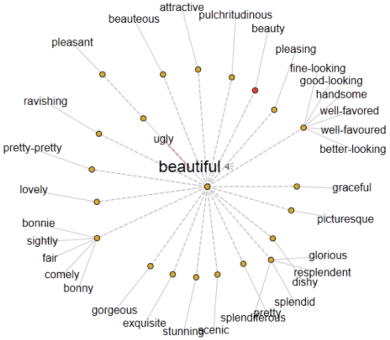
```
def naive_stem(word):
    regexp = r'^(.*?)(ing|ly|ed|ious|ies|ive|es|s|ment)?'
    stem, suffix = re.findall(regexp, word)[0]
    return stem

>>> tokens = ['women', 'swords', 'is', 'lying']

>>> [naive_stem(t) for t in tokens]

    ['women', 'sword', 'i', 'ly']          // naive

>>> [nltk.WordNetLemmatizer().lemmatize(t)
     for t in tokens]

    ['woman', 'sword', 'is', 'lying']    // dict-based

>>> [nltk.PorterStemmer().stem(t)
     for t in tokens]

    ['women', 'sword', 'is', 'lie']       // rule-based

```

Text Features: Synsets image from [?]

- Can we achieve invariance to **synonyms**?

  "What a beautiful day!" vs.
  "What a lovely day!"

- A frequent approach are **thesauri**: A thesaurus is a collection of terms, connected by (pre-defined) relations
- Typical relations
  - synonyms *(beautiful vs. lovely)*
  - antonym *(beautiful vs. ugly)*
  - generalization/specialization *(a boat* **is a** *vehicle)*
- Synonyms form so-called **synsets**



Synsets: Python Example

```
>>> from nltk.corpus import wordnet as wn
>>> wn.synsets("dog")

  [Synset('dog.n.01'),
   Synset('frump.n.01'),
   Synset('dog.n.03'),
   Synset('cad.n.01'),
   Synset('frank.n.02'),
   Synset('pawl.n.01'),
   Synset('andiron.n.01'),
   Synset('chase.v.01')]

>>> for synset in wn.synsets("dog"):
        print "dog =", synset.definition

   dog = a member of the genus Canis ...
   dog = a dull unattractive unpleasant woman
   dog = informal term for a man
   dog = a smooth-textured sausage ...
   dog = metal supports for logs in a fireplace
   dog = go after with the intent to catch
   ...
```

From Thesauri to Ontologies image from [?]

- We can extend the concept of a thesaurus to *ontologies*
- An ontology can be thought of as a generalized **knowledge base** containing objects and relations between them
- Ontologies can be **combined** by linking their objects

Example: The DBPedia Project

- 20.8 mio. "things", crawled from Wikipedia infoboxes
- $>$ 500 mio. "facts"
- representation by RDF *(Resource Description Framework)*
- allows **smarter search** ("give me all cities in New Jersey with more than 10,000 inhabitants")

```
{{Infobox Town AT |
  name = Innsbruck |
  image_coa =  InnsbruckWappen.png |
  image_map = Karte-tirol-I.png |
  state = [[Tyrol]] |
  regbzk = [[Statutory city]] |
  population = 117,342 |
  population_as_of = 2006 |
  pop_dens = 1,119 |
  area = 104.91 |
  elevation = 574 |
  lat_deg = 47 |
  lat_min = 16 |
  lat_hem = N |
  lon_deg = 11 |
  lon_min = 23 |
  lon_hem = E |
  postal_code = 6010-6080 |
  area_code = 0512 |
  licence = I |
  mayor = Hilde Zach |
  website = [http://innsbruck.at] |
}}
```

Text Features: N-Grams

- So far, we have neglected the **order** of words in the document

  "I can ***not*** *believe it –* ***What*** *a* ***cool*** *video!" vs.*
  "This video is ***not cool*** *–* ***What*** *a..."*

- A simple statistical approach are **n-grams**: Instead of segmenting text into single tokens, we segment it into subsequences of n tokens each!

In the Example

- bag-of-words feature

  $\left\{ \textit{(This: 1), (video: 1), (is: 1), (cool: 1), ...} \right\}$

- n-gram feature

  $\left\{ \textit{(This video: 1), (video is: 1), (is not: 1), (not cool: 1), ...} \right\}$

- Problem: Features get (even more) **high-dimensional**!

References I

[1] Affine Covariant Features Dataset.
http://www.robots.ox.ac.uk/~vgg/research/affine/ (retrieved: Oct 2016).

[2] Body Language: What we're really saying.
https://capitaleap.org/blog/2013/06/12/body-language-what-were-really-saying/ (retrieved: Oct 2016).

[3] Did you blink? The structured Web just arrived.
http://www.mkbergman.com/354/did-you-blink-the-structured-web-just-arrived/ (retrieved: Oct 2016).

[4] picture shared by Christoph Lampert.
contact: http://pub.ist.ac.at/~chl/.

[5] Studie: "Kreditschwemme" kommt beim Mittelstand nicht an .
http://www.wirtschaft.com/studie-kreditschwemme-kommt-beim-mittelstand-nicht/ (retrieved: Oct 2016).

[6] The USC-SIPI Image Database.
http://sipi.usc.edu/database/ (retrieved: Oct 2016).

[7] Thesaurus Quotes.
http://quotesgram.com/img/thesaurus-quotes/3138560/ (retrieved: Oct 2016).

[8] Wang, R.: Computer Image Processing and Analysis (E161) Course (Harvey Mudd College).
http://fourier.eng.hmc.edu/e161/lectures/gradient/node8.html (retrieved: Oct 2016).

[9] Yes, this is Megan Fox.
like, everywhere on the internet... (retrieved: Oct 2016).

[10] Shrek, 2001.

References II

[11] S. Bird, E. Klein, and E. Loper.
Natural Language Processing with Python.
O'Reilly Media, Inc., 2009.

[12] D. G. Lowe.
Distinctive image features from scale-invariant keypoints.
Int. J. Comput. Vision, 60(2):91–110, 2004.

[13] C. Manning, P. Raghavan, and H. Schütze.
Introduction to Information Retrieval.
Cambridge University Press, 2008.