Machine Learning

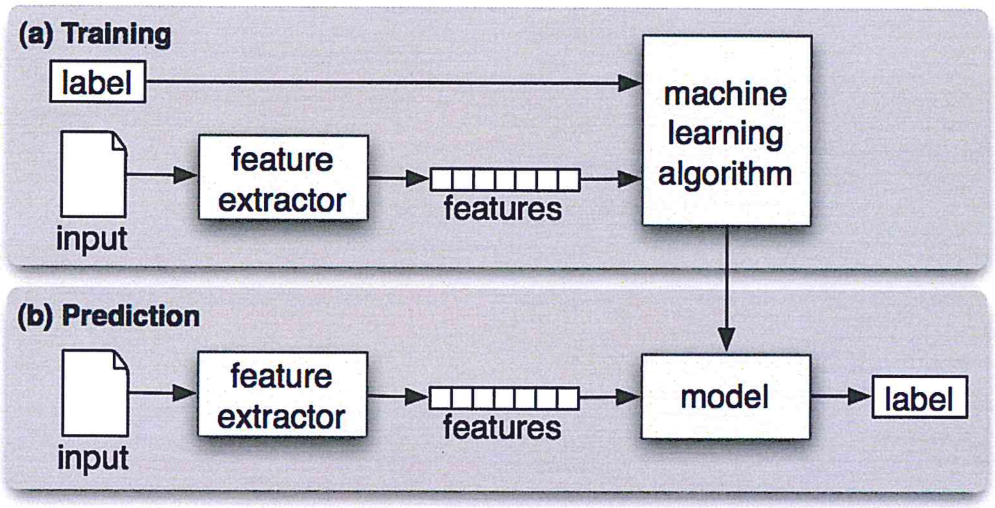
– winter term 2016/17 –

Chapter 03: Features

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

Outline

Introduction image from [11]



- 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

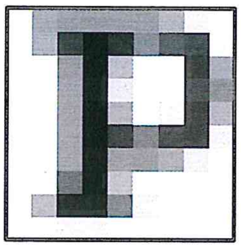
- Features are formal representations of **real-world objects**
- We can think of them as attribute-value pairs

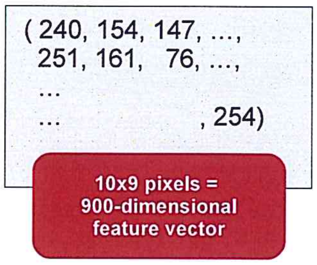
```
color = silver,         // categorial feature
rating = ****,          // ordinal feature
mileage = 20.8,         // numerical feature
price development = (34.457, 32.189, 29.745)
                        // vector feature
```

- 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? → Example: OCR





- 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

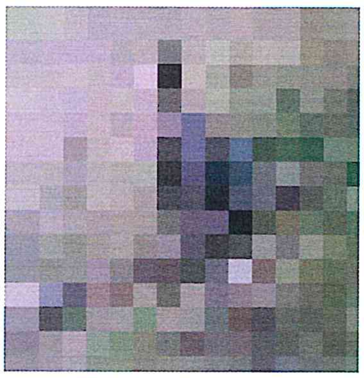
What are properties of good features?

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

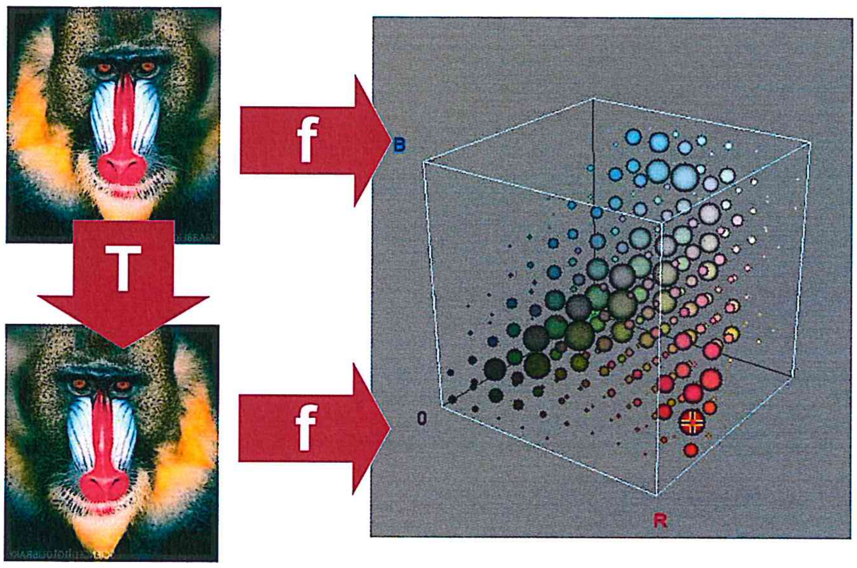
Invariance of Features in ML

- ... 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 [6]

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



Invariance: Sample Transformations T images from [2] [10]

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

Machine Learning on Images

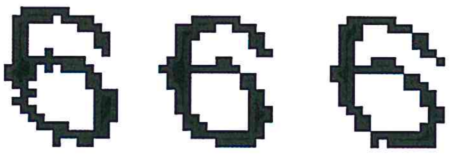
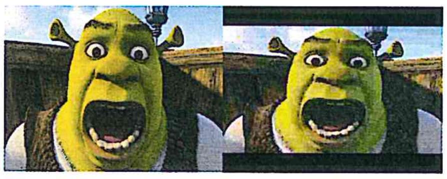
- illumination
- perspective, pose
- geometric transformations
- noise, compression artifacts

Machine Learning on Text

- language
- wording (synonyms)

Other Machine Learning

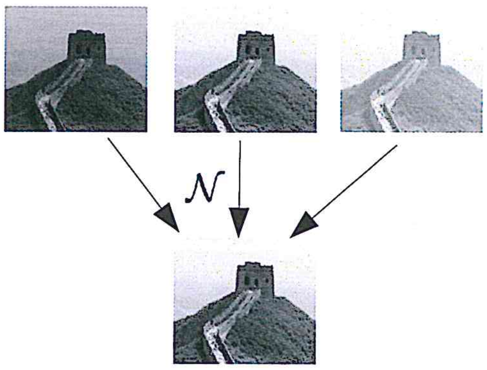
- inflation (in credit scoring)
- user rating level (in recommenders)
- ...

Strategies to achieve Invariance

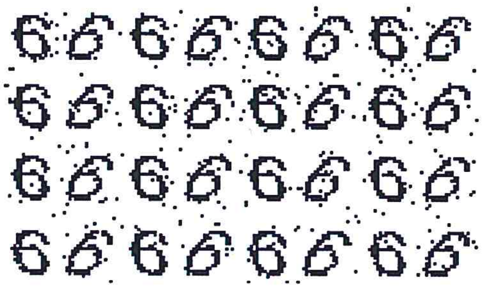
Approach 1: Normalization

- 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?)*

$$f(\mathsf{M}) = f(\mathsf{W})?$$

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: Neares 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

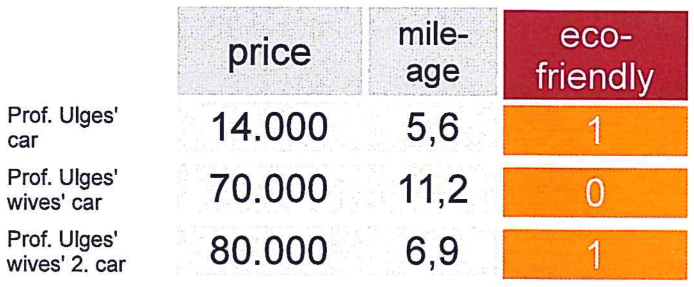
- Decision tree accuracy (5-fold-crossvalidation on training set)
- We add uninformative features $\sim U[0,1]$ to the data
- We set `max_depth=10` (the effect grows with `max_depth`)

# 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 "PS" 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	1

Approach: Feature Normalization

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

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

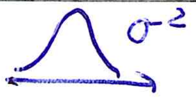
- **Approach 2: Standardization**

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

Approach 3: Whitening

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

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 $\mathbf{X} = (X_1, \dots, X_d)$ 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 $d=2,\ \mu = \begin{pmatrix} 0 \\ 0 \end{pmatrix},\ \Sigma = \begin{pmatrix} 9 & 0 \\ 0 & 1 \end{pmatrix},\ \Sigma^{-1} = \begin{pmatrix} 1/9 & 0 \\ 0 & 1 \end{pmatrix}$

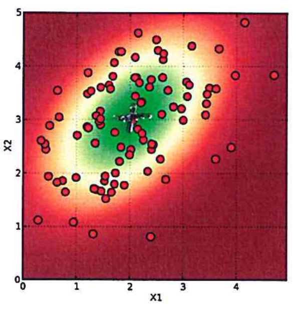
$$p(x, \dots) = \frac{1}{2\pi \cdot 3} \cdot e^{-\frac{1}{2} \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}^T \begin{pmatrix} 1/9 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}} = \frac{1}{2\pi \cdot 3} \cdot e^{-\frac{1}{2} \cdot \left(\frac{x_1^2}{9} + x_2^2 \right)}$$

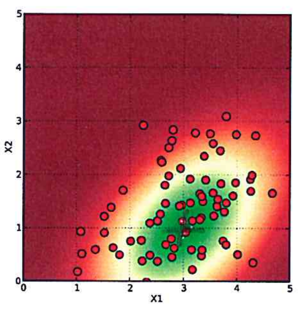
Example Visualization in 2D

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.

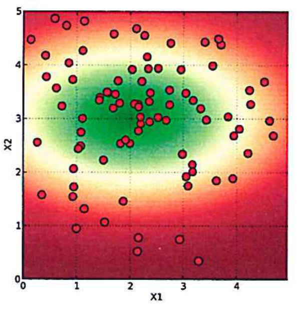
  $\mu = (2,3) \rightarrow$
  $\mu = (3,1)$

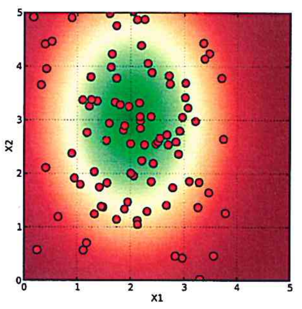




- Changing values on Σ's diagonal leads to a **re-scaling**

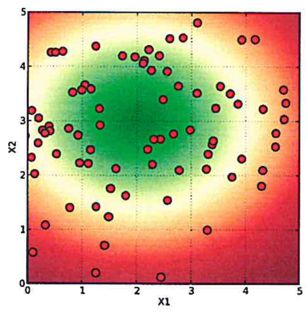
  $\Sigma = \begin{pmatrix} 3 & 0 \\ 0 & 1 \end{pmatrix} \rightarrow$
  $\Sigma' = \begin{pmatrix} 1 & 0 \\ 0 & 2 \end{pmatrix}$



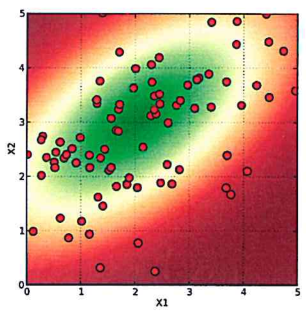


Visualization in 2D

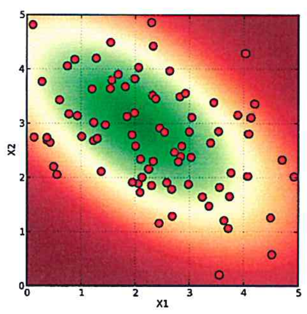
- Values Σ_{ij} off Σ's diagonal (i.e., $i \neq j$) indicate the **covariance / correlation** between different variables
- We distinguish **three cases**:
  - $\Sigma_{ij} = 0$ (X_i and X_j are uncorrelated)
  - $\Sigma_{ij} > 0$ (positive correlation between X_i and X_j)
  - $\Sigma_{ij} < 0$ (negative 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 entries in Σ correspond to the **covariances** between the single variables of the random vector:

$$\Sigma_{ij} = Cov(X_i, X_j) = E((X_i - \mu_i) \cdot (X_j - \mu_j))$$

- The diagonal contains the **variances** of $\mathbf{X}$'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)

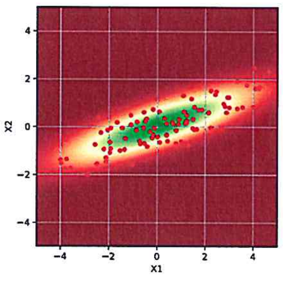
Let $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n \in \mathbb{R}^d$ be a training set with covariance matrix Σ with eigenvalues $\lambda_1, ..., \lambda_d$ and eigenvectors $\mathbf{p}_1, ..., \mathbf{p}_d$. We define the $d \times d$ matrices

$$D^{-\frac{1}{2}} := Diag(\frac{1}{\sqrt{\lambda_1}}, \frac{1}{\sqrt{\lambda_2}}, ..., \frac{1}{\sqrt{\lambda_d}}) \quad \text{and} \quad P = \begin{pmatrix} p_1 \; p_2 \; ... \; p_d \end{pmatrix}$$

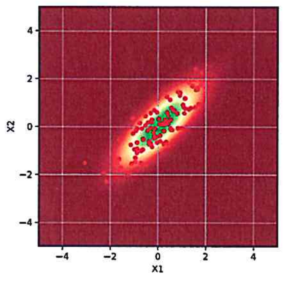
Then, we call the following transformation a **whitening***:*

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

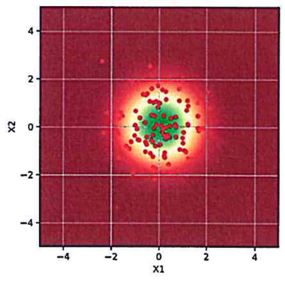
Illustration



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 $\Leftrightarrow$ Model

Food for Thought

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

3. Interdependency Features $\Leftrightarrow$ Model image from [5]

Example: Online-Shop

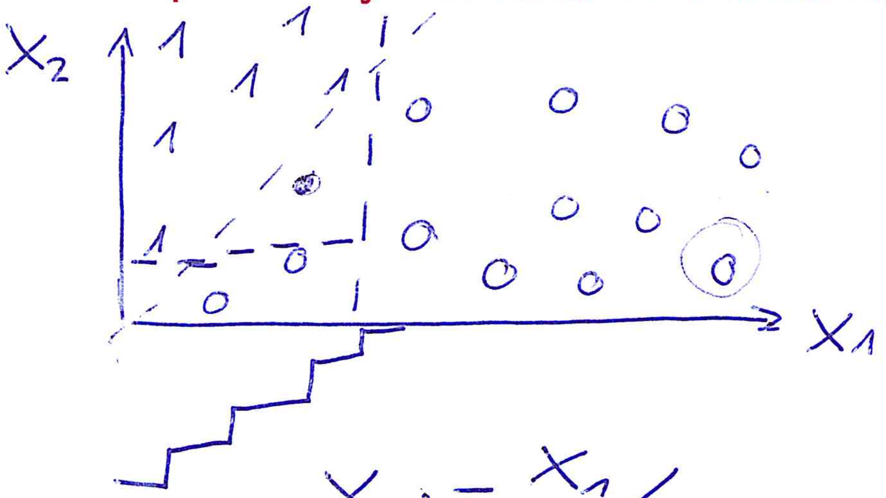
- Goal: Predict whether a **product in your shop** will be bought
- Features

$$\begin{aligned} x_1 &:= \text{the product's } \textbf{price} \\ x_2 &:= \text{the product's } \textbf{average price} \\ &\quad\;\; \text{over other many other shops} \end{aligned}$$

Do it Yourself

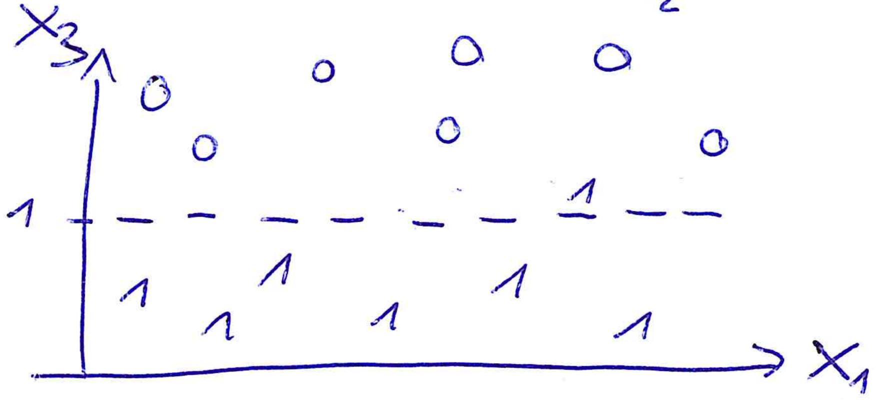
- Sketch the data in feature space.
- What works better: **decision trees** or a **linear classifier**?
- How can we resolve the problem?

3. Interdependency Features ⇔ Model image from [5]



0 = item not bought
1 = item bought

$$X_3 := X_1 / X_2$$



Outline

emarks regarding Text Features

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!

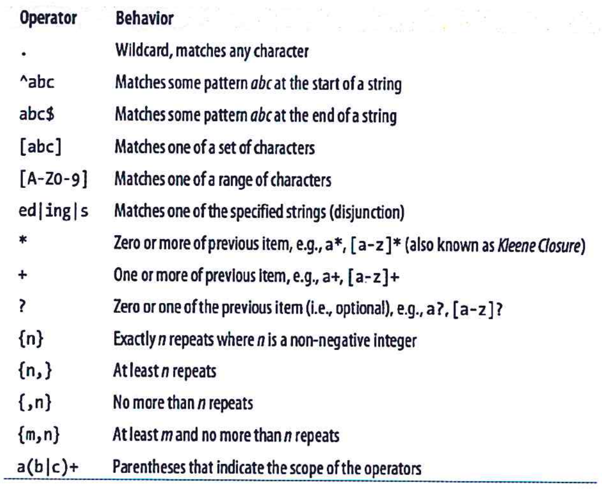
Text Features: Segmentation

- 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 [11]



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

Code Example: Python

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

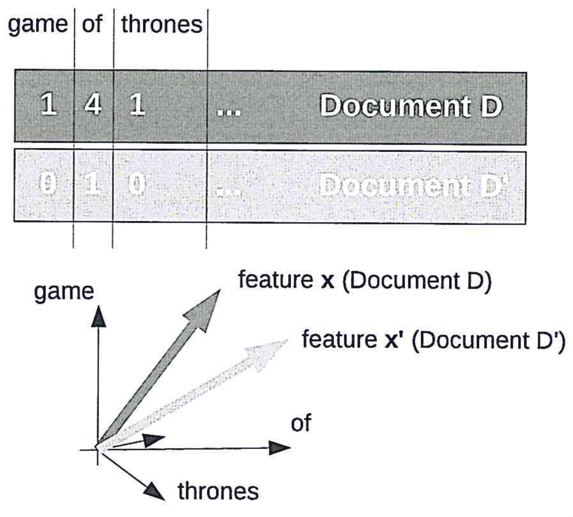
```
>>> text = 'That U.S.A. poster-print costs $12.40...'
>>> pattern = r'''(?x)    # set flag to allow verbose regexps
...     ([A-Z]\.)+        # abbreviations, e.g. U.S.A.
...   | \w+(-\w+)*        # words with optional internal hyphens
...   | \$?\d+(\.\d+)?%?  # currency and percentages, e.g. $12.40, 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
- **x** is very **sparse** (only terms appearing in D get a weight $\neq 0$)

Popular Term Weightings (more in [13], 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

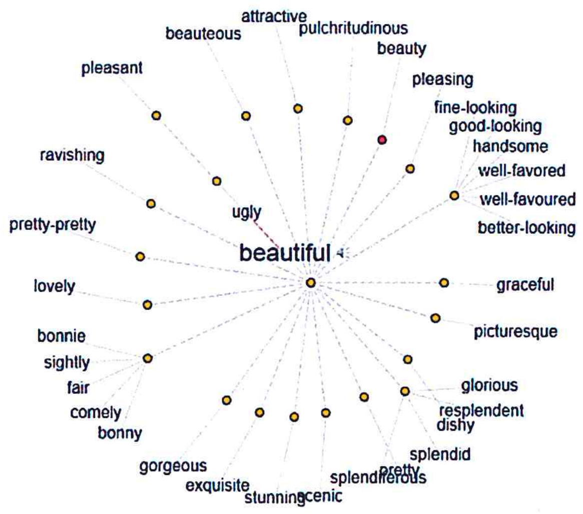
```
def naive_stem(word):
    regexp = r'^(.*?)(ing|ly|ed|ious|ies|ive|es|s|ment)?'
    stem, suffix = re.findall(regexp, word)[0]
    return stem

>>> 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 [7]

- 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

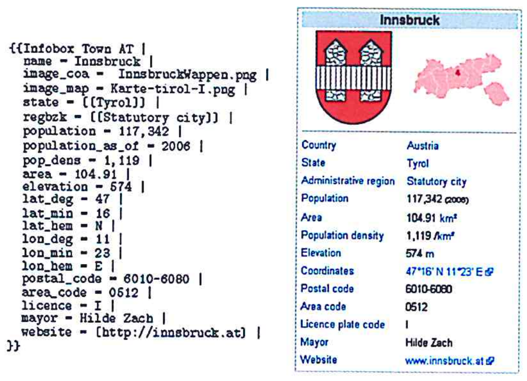
```
>>> from nltk.corpus import wordnet as wn
>>> wn.synsets("dog")

  [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 [3]

- 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] |
}}
```

Innsbruck	
Country	Austria
State	Tyrol
Administrative region	Statutory city
Population	117,342 (2006)
Area	104.91 km²
Population density	1,119 /km²
Elevation	574 m
Coordinates	47°16′ N 11°23′ E
Postal code	6010-6080
Area code	0512
Licence plate code	I
Mayor	Hilde Zach
Website	www.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

  $\Big\{$ *(This: 1), (video: 1), (is: 1), (cool: 1), ...* $\Big\}$

- n-gram feature

  $\Big\{$ *(This video: 1), (video is: 1), (is not: 1), (not cool: 1), ...* $\Big\}$

- 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.