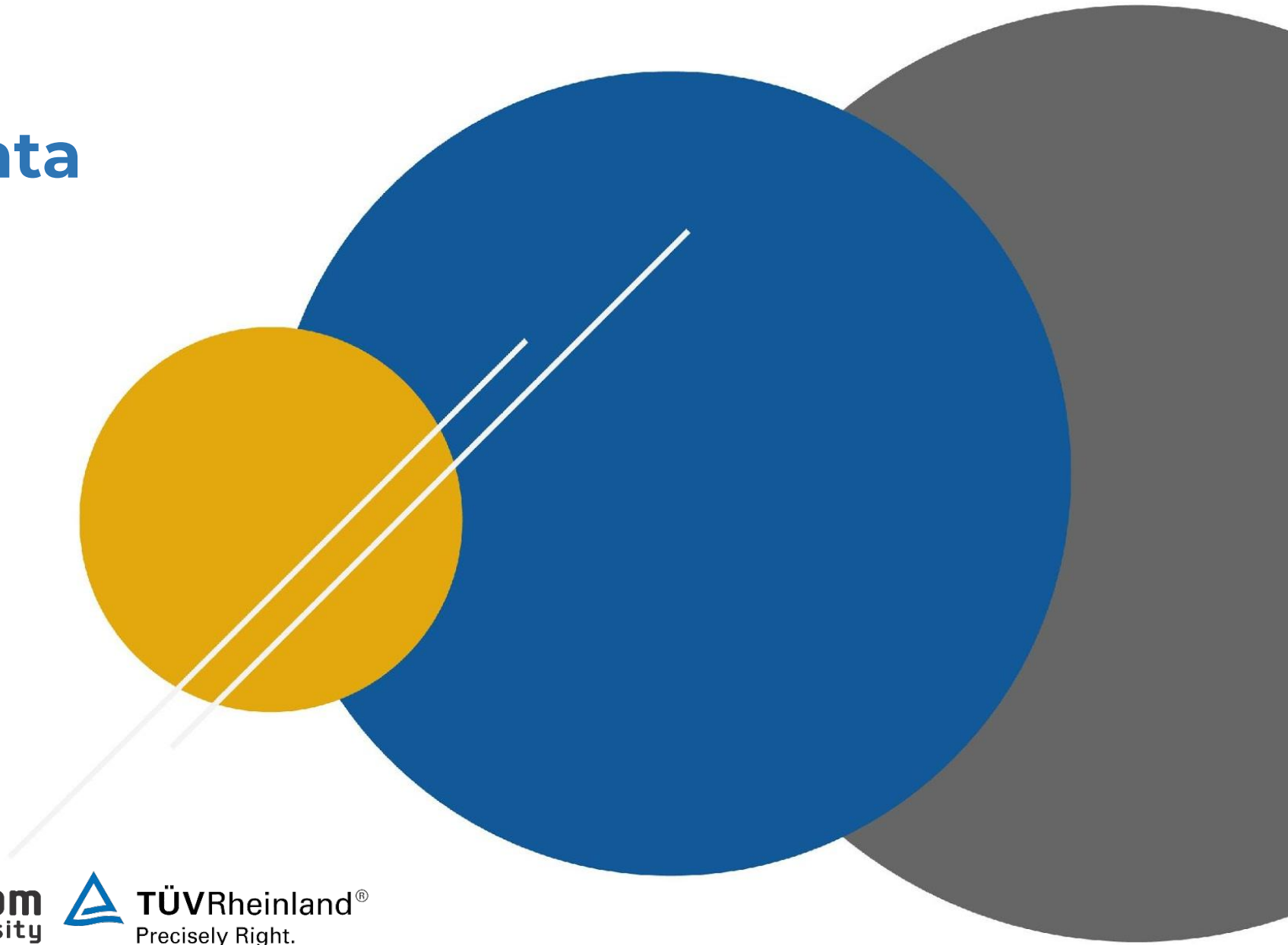


Unstructured Data

Text Mining



How This Course is Delivered

- Text mining is basically data mining on unstructured text. It's an implementation of the many techniques explained in previous sessions.
- Combining the basic theory by focusing on implementation using Python.
- The basic explanation of the theory and algorithm is not explained using in depth mathematical and statistical approach, to make it easier for participants who do not have statistics or mathematics background



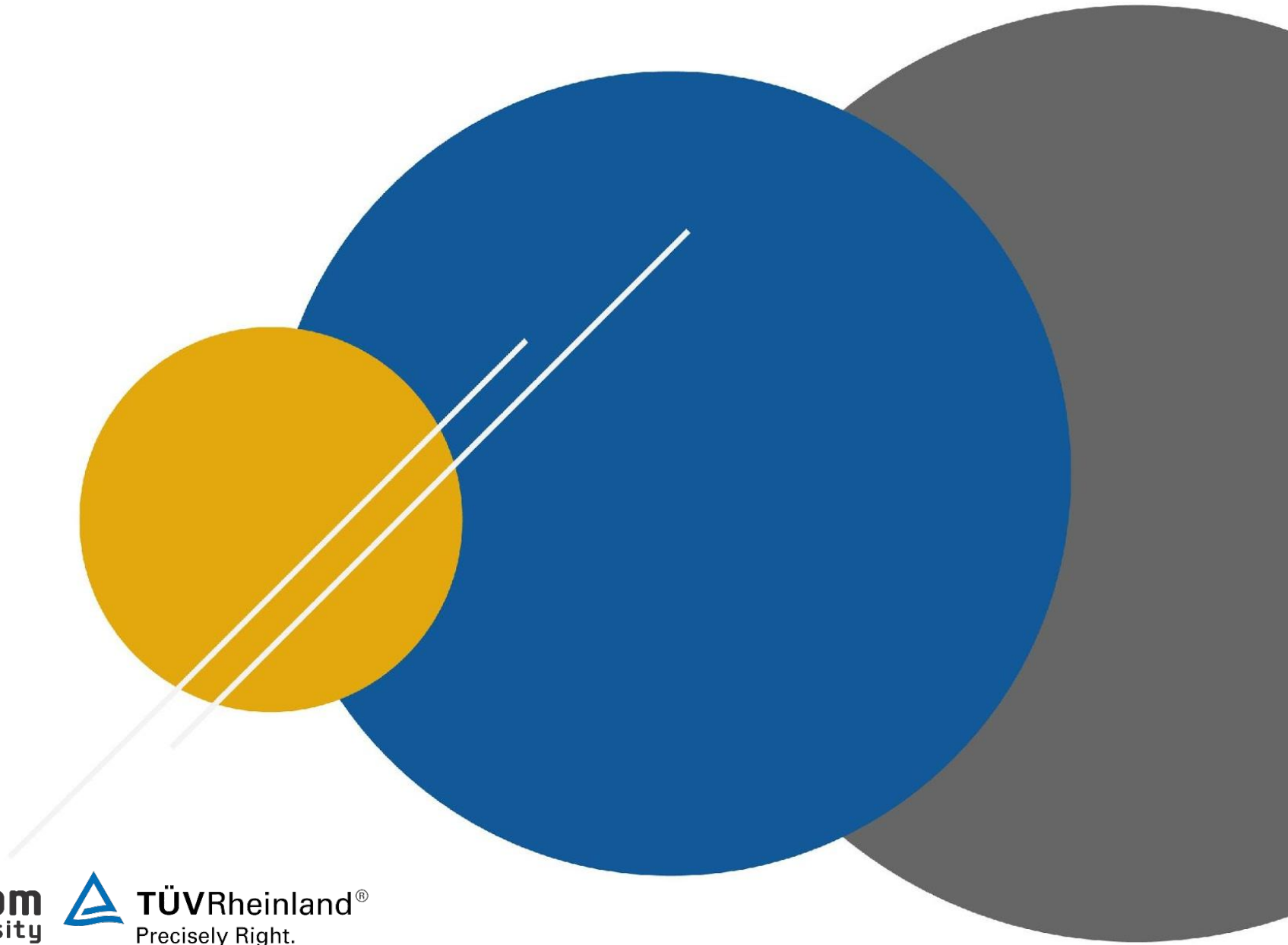
Agenda

- ➡ Definition of text mining and common application
- ➡ Text mining and NLP preprocessing technique
- ➡ How to extract important information from text
- ➡ Understand how text is handled in Python



Chapter 1

Introduction



Text Mining

Definition

- Broad umbrella terms describing a range of technologies for analyzing and processing semi structured and unstructured text data.
- The unifying theme behind each of these technologies is the need to “turn unstructured text into structured data” so powerful algorithms can be applied to large document databases.
- Converting text into a structured, numerical format and applying analytical algorithms require knowing how to both use and combine techniques for handling text, ranging from individual words to documents to entire document databases
- In building a statistical language system, it is best to devise a model that can make good use of available data, even if the model seems overly simplistic.



Text Mining

Purpose

- Turn text data into high-quality information and/or actionable knowledge
 - Minimizes human effort on consuming text data
 - Supplies knowledge for optimal decision making → actionable knowledge



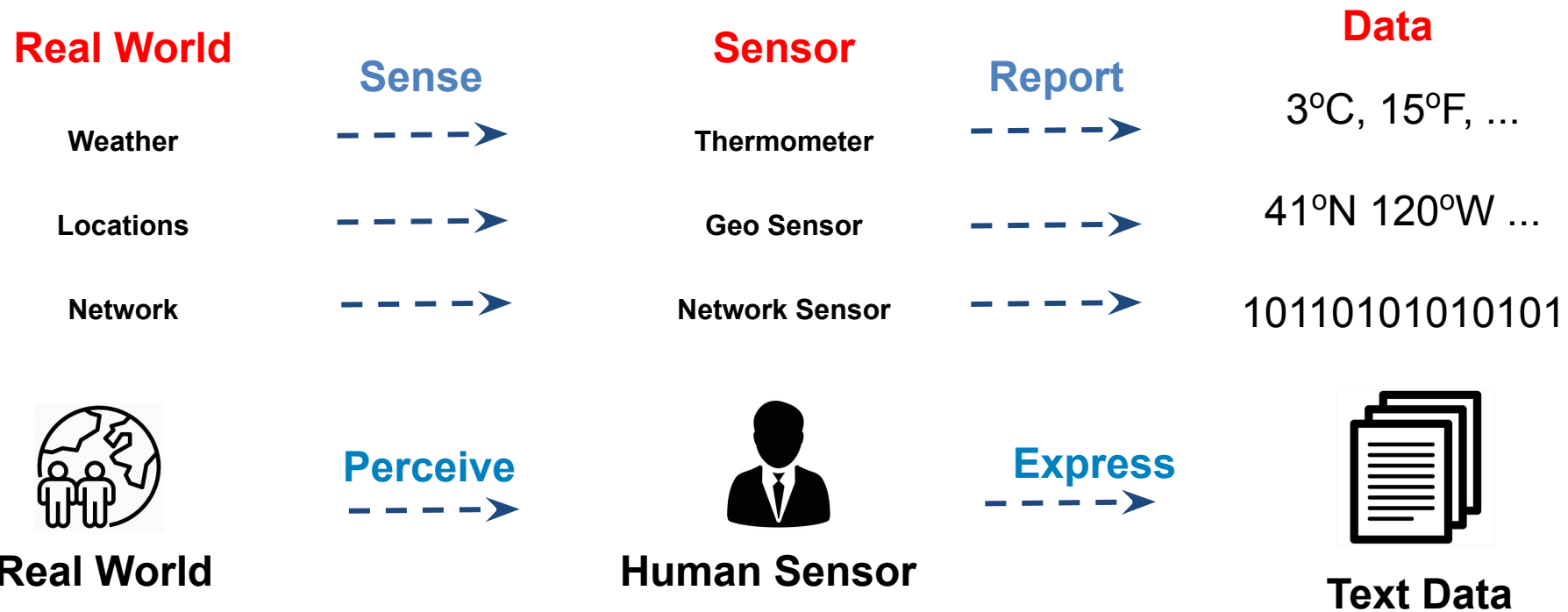
Text Mining

Applications

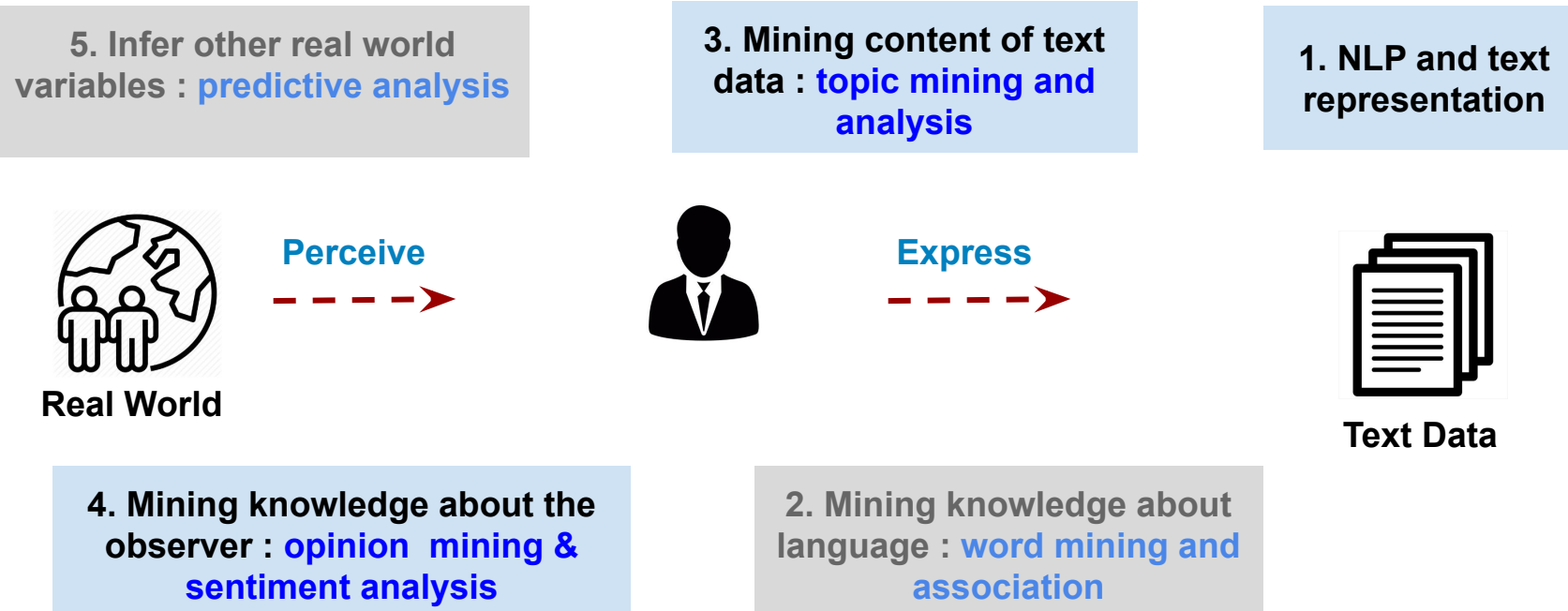
- **Summary:** search for the most important information from a text.
- **ChatBot:** an automatic question and answer application between machines and humans.
- **Text categorization:** determine the topic of a particular document
- **Keyword Tag:** keyword tags are selected keywords that represent text.
- **Sentiment Analysis:** determine the sentiment or value of opinion in a text, these sentiments can be negative, neutral, or positive sentiments.
- **Speech-to-text and text-to-speech conversions:** Turns sound into text and vice versa
- **Translator Machine:** Translation of text from one language to another
- **Spelling Checker**



Humans as Subjective Sensors



Text Mining Landscape



Natural Language Processing

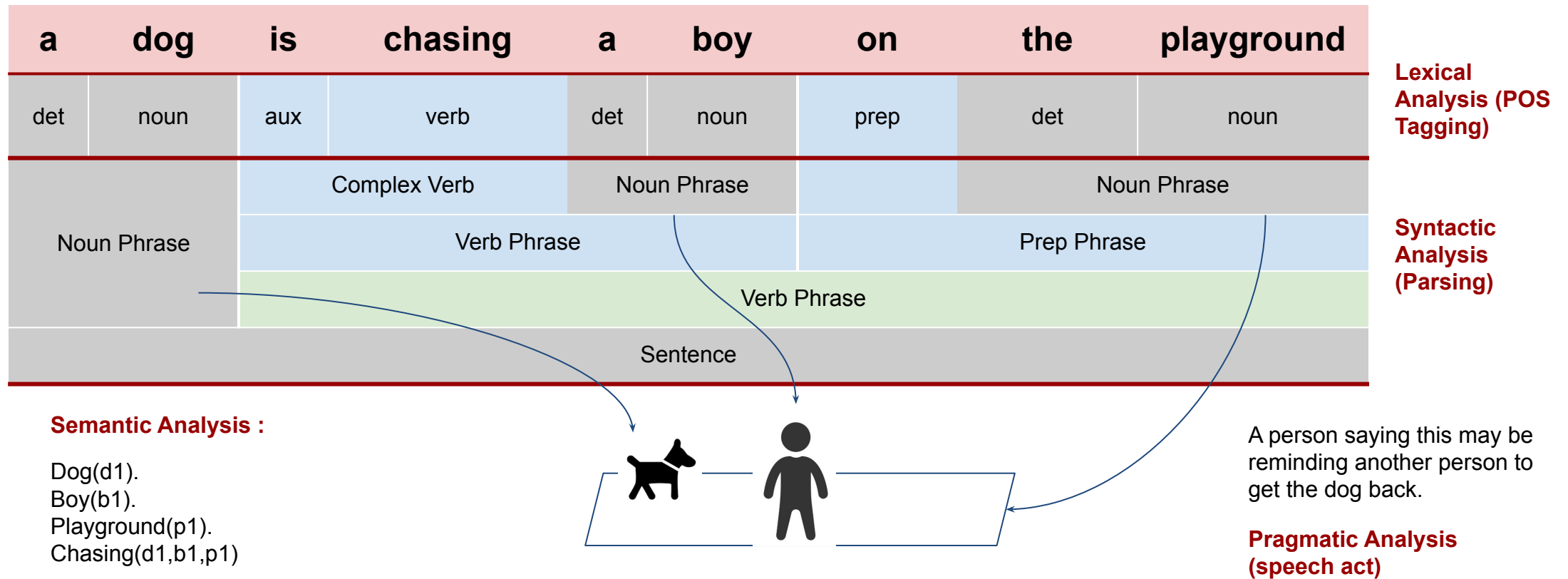
Definition

- ➡ NLP is a research area of computer science, artificial intelligence, and computational linguistics, concerned with the interactions between computers and human natural languages.
- ➡ Helps computers understand, interpret, and manipulate human language. Not only understand the word, but also how these words are interconnected into a meaningful information



Natural Language Processing

Basic Concepts



Natural Language Processing

Challenges

- Language is ambiguous → need context to explain
 - The same word can mean something else (homograph)
 - Bank - Sloping land (especially the slope beside a body of water)
 - Bank - A financial institution that accepts deposits and channels the money into lending activities
 - Different words means the same (synonyms)
- Human errors - misspellings, typos, abbreviations, social languages, etc.
- Each language is different in terms of structure, vocabulary, etc
- Special requirements: regulation and privacy related to legal, diplomatic, medical



Natural Language Processing

Implementation Packages

- Apache OpenNLP: Machine learning toolkit that provides tokenization, sentence segmentation, part of speech tagging, named entity extraction, chunking, parsing, coreference resolution, and so on.
- Natural Language Toolkit (NLTK): the most popular python library for NLP, consisting of: classification, tokenization, stemming, tagging, parsing, and others.
- Stanford NLP: used for part of speech tagging, named entity recognizer, coreference resolution, sentiment analysis, etc.
- MALLET: is a JAVA package consists of Latent Dirichlet Allocation, document classification, clustering, topic modeling, information extraction, and others.



Lab Preparation

Python Library Required

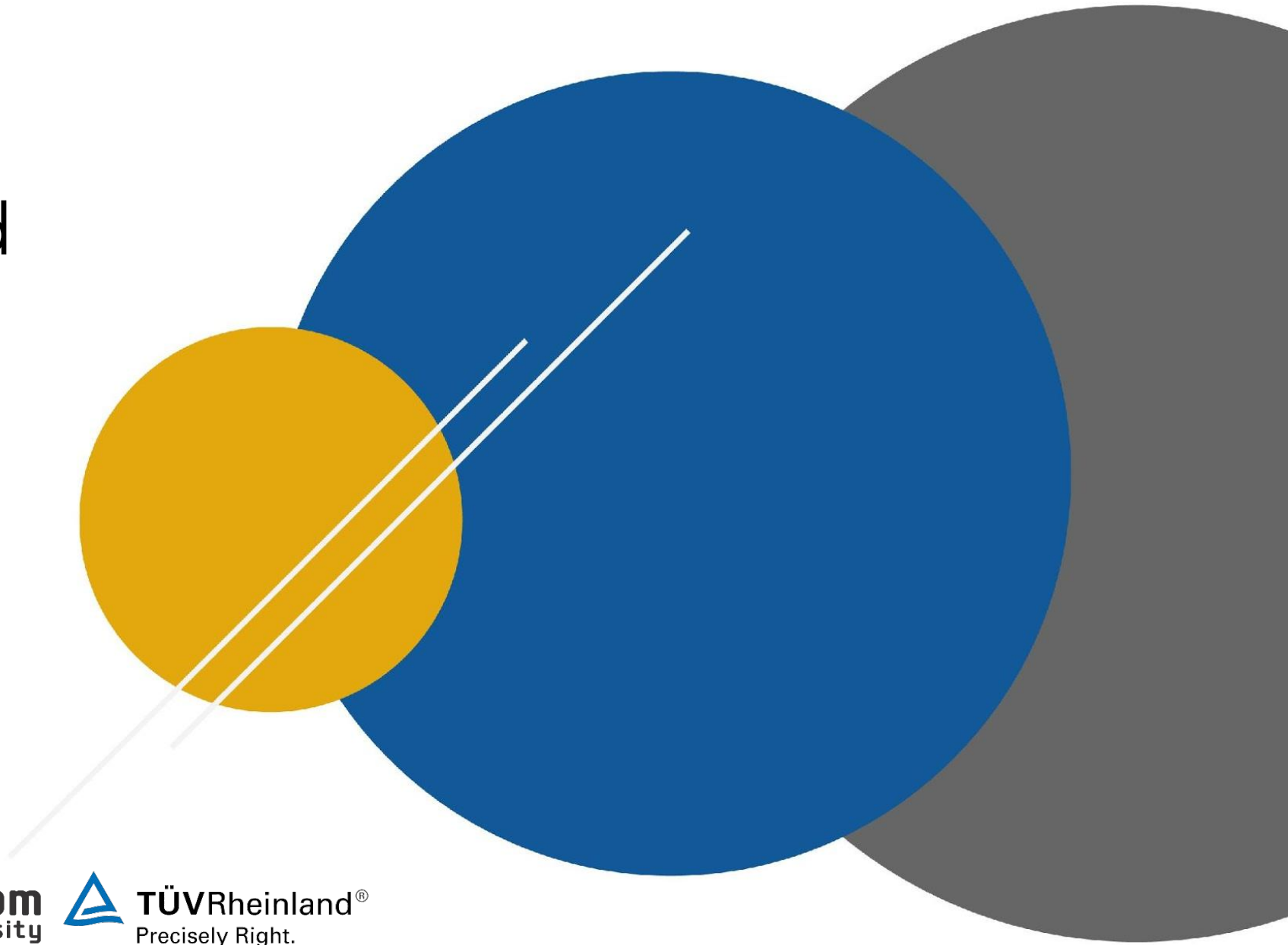
In this training we will use below python library

- nltk: the most popular NLP library in the python ecosystem
- BeautifulSoup4: library for extracting data from HTML and XML documents
- pandas: library for data manipulation and analysis
- scikit-learn: python machine learning library
- matplotlib: library for 2-dimensional plotting
- sastrawi: stemmer for Indonesian Language, ported from PHP
- gensim: python library focused on analyzing plain-text documents for semantic structure



LAB 01

Installation and Configuration



Lab Description

What you will learn:

- Anaconda installation
- Create and manage anaconda environment
- Install python and the required packages
- Run Jupyter Notebook

Requirement :

- Anaconda
- Jupyter Notebook
- Python 3 library
 - numpy
 - pandas
 - scikit-learn
 - nltk
 - matplotlib
 - beautifulsoup4
 - sastrawi



STEP 01

Install Anaconda

Install Anaconda according to your OS.

Installer can be downloaded at:

www.anaconda.com/download

 Windows |  macOS |  Linux

Anaconda 2019.03 for Windows Installer

Python 3.7 version

Download

64-Bit Graphical Installer (662 MB)

32-Bit Graphical Installer (546 MB)

Python 2.7 version

Download

64-Bit Graphical Installer (587 MB)

32-Bit Graphical Installer (493 MB)



STEP 02

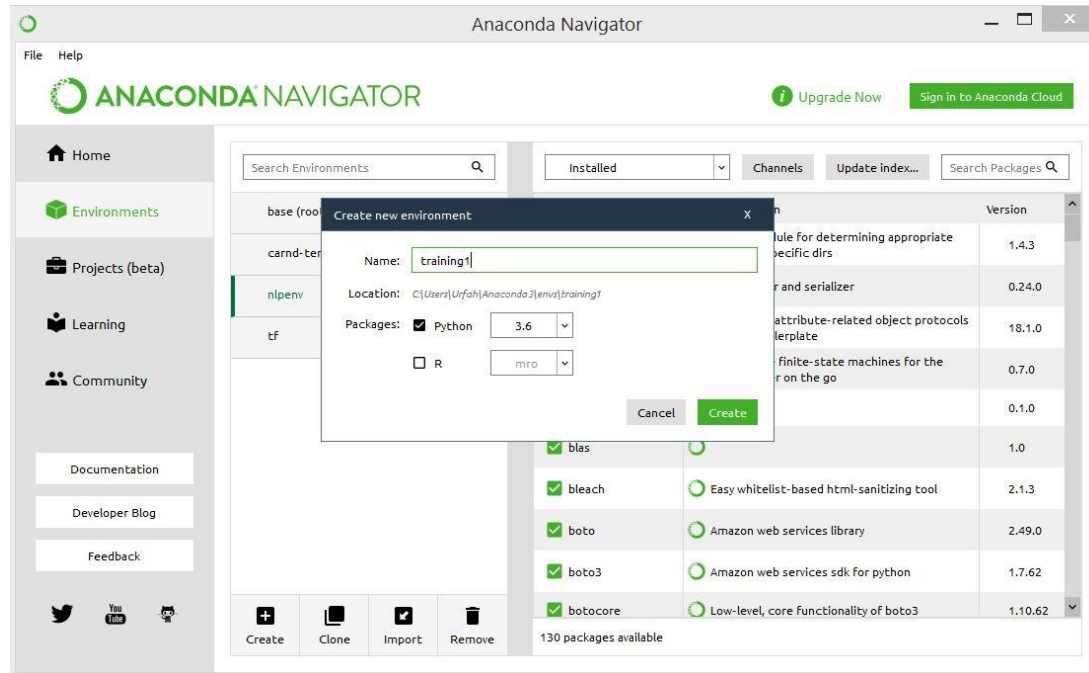
Create Environment

- Conda environment is basically a certain directory that contains all the packages we install. We can have several conda environments with different package versions. For example we need to run python 2 in one environment, and python 3 in another.
- By default, Anaconda creates one main environment called base (root).
- There are two ways to create and manage the environment: by using Anaconda Navigator (Graphical User Interface) or Anaconda prompt (Command Line Interface)
- To create an environment through the GUI, run Anaconda Navigator



STEP 02

Create Environment



- Choose **Environment**, and click **Create**
- Name the environment, for example **training1**
- Choose python version



STEP 03

Install Package

The screenshot shows the Anaconda Navigator interface. In the top right, there is a dropdown menu labeled 'Not installed' which is circled in red. Below this, the 'Install Packages' dialog box is open, showing a list of 32 packages to be installed. The packages are listed in a table with columns: Name, Unlink, Link, and Channel. The packages listed are: beautifulsoup4, matplotlib, nltk, numpy, pandas, and scikit-learn. The 'Apply' button is highlighted in green.

Name	Unlink	Link	Channel
1 beautifulsoup4	-	4.6.1	defaults
2 matplotlib	-	2.2.2	defaults
3 nltk	-	3.3.0	defaults
4 numpy	-	1.15.0	defaults
5 pandas	-	0.23.3	defaults
6 scikit-learn	-	0.19.1	defaults

* Indicates the package is a dependency of a selected packages

Cancel Apply

- In the drop down list, select Not Installed, then check the packages that we will install
- Click **Apply** button



STEP 04

Install Package Using pip

- Sastrawi package installed through CLI, by using pip
- Follow this steps :
 - Open Anaconda Prompt
 - Activate environment with command activate <environment name>
 - Install Sastrawi by using command pip install Sastrawi



```

Anaconda Prompt

(base) C:\Users>activate training1

(training1) C:\Users>pip install Sastrawi
Collecting Sastrawi

```

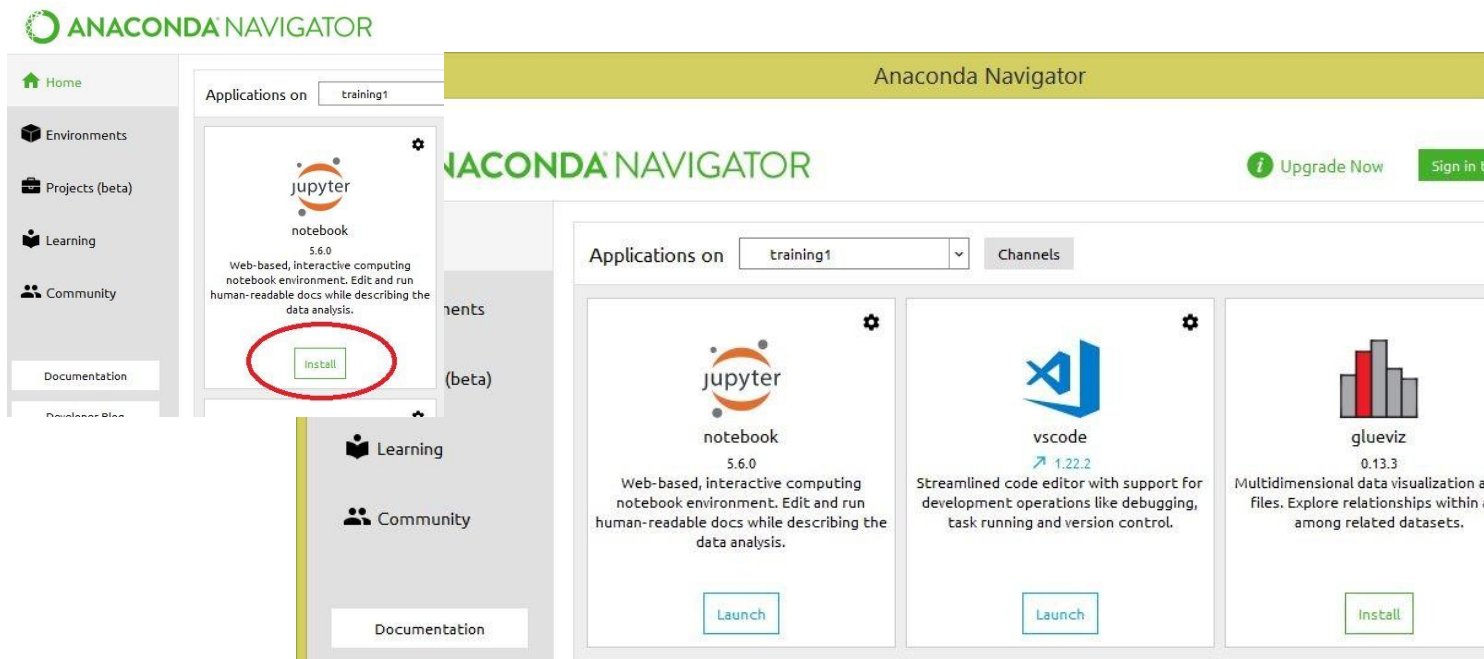


STEP 05

Run Jupyter Notebook

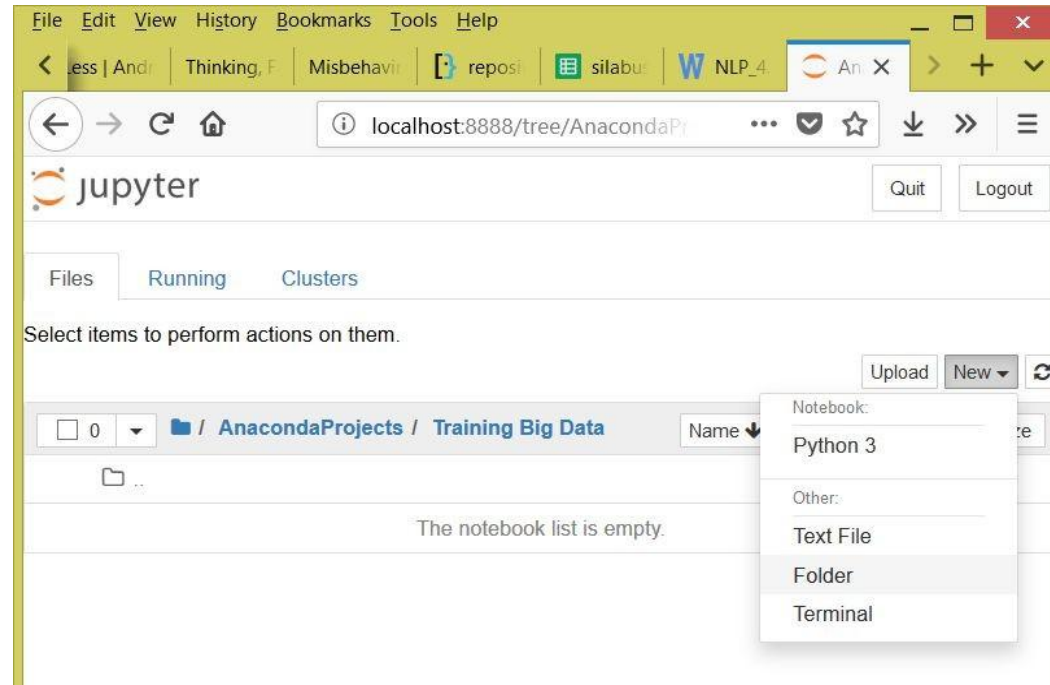
Install Jupyter Notebook through Anaconda

Navigator in Home menu, and click Launch



STEP 05

Run Jupyter Notebook

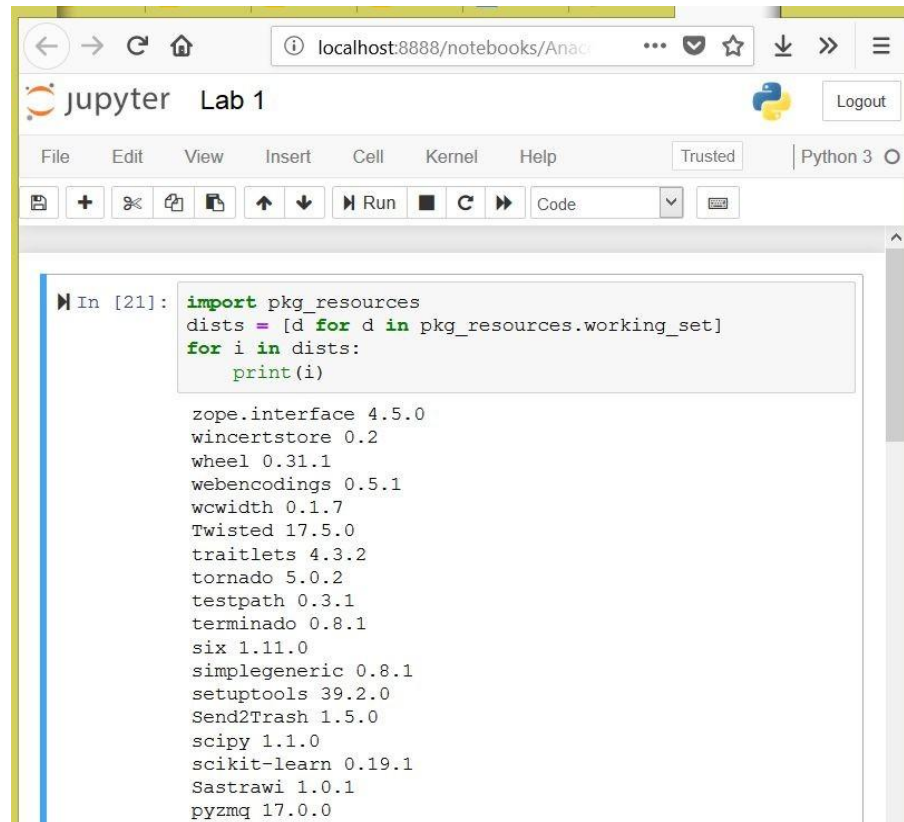


- Jupyter notebook will be opened as a tab in your browser
- You can create new folder or notebook by clicking New



STEP 06

Check Package Version



```
In [21]: import pkg_resources
dists = [d for d in pkg_resources.working_set]
for i in dists:
    print(i)

zope.interface 4.5.0
wincertstore 0.2
wheel 0.31.1
webencodings 0.5.1
wcwidth 0.1.7
Twisted 17.5.0
traitlets 4.3.2
tornado 5.0.2
testpath 0.3.1
terminado 0.8.1
six 1.11.0
simplegeneric 0.8.1
setuptools 39.2.0
Send2Trash 1.5.0
scipy 1.1.0
scikit-learn 0.19.1
Sastrawi 1.0.1
pymq 17.0.0
```

We can check the version of all package installed in our current environment, with the following code :

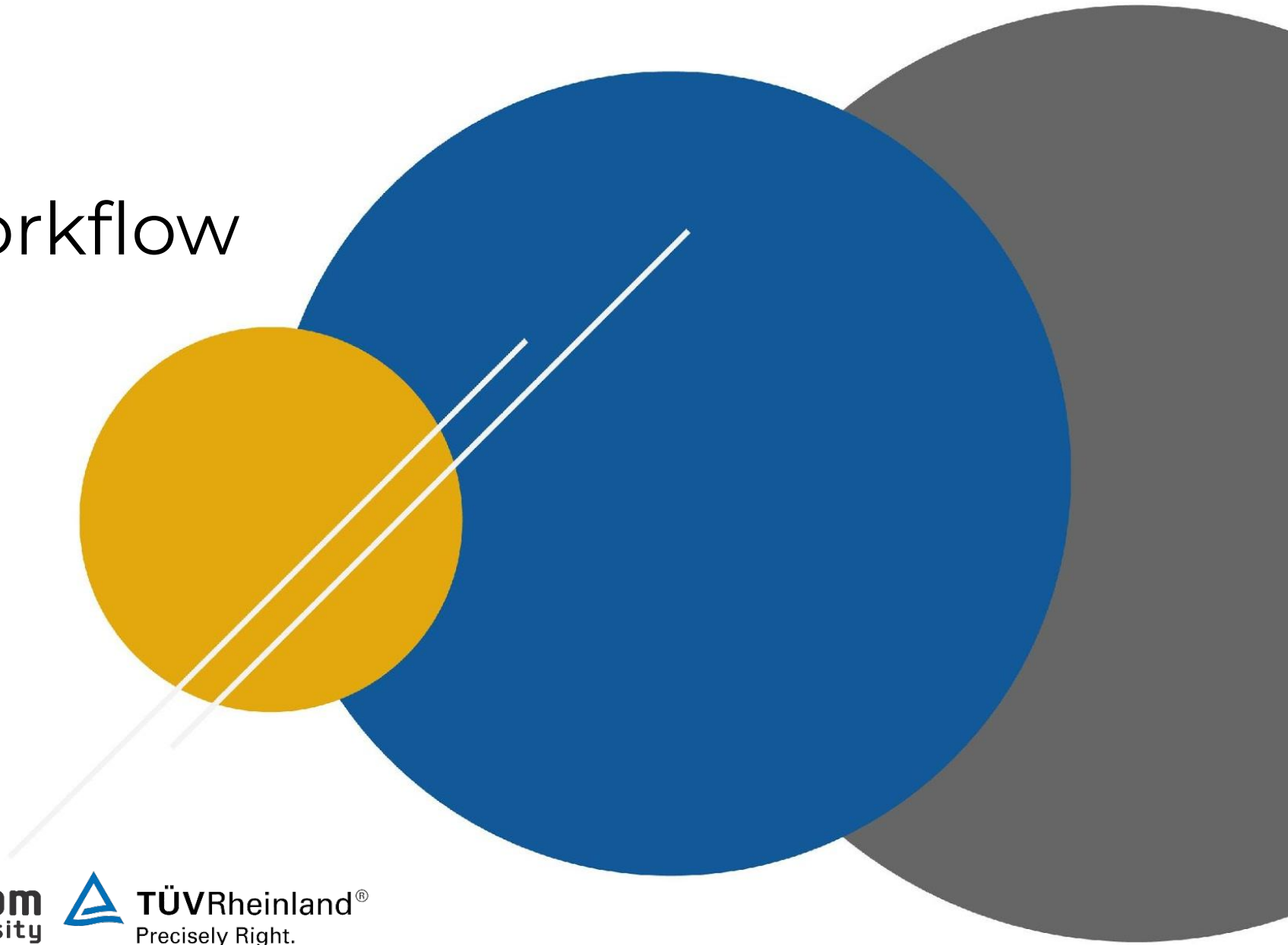
```
import pkg_resources
dists = [d for d in pkg_resources.working_set]
for i in dists:
    print(i)
```

Click **Run** to execute

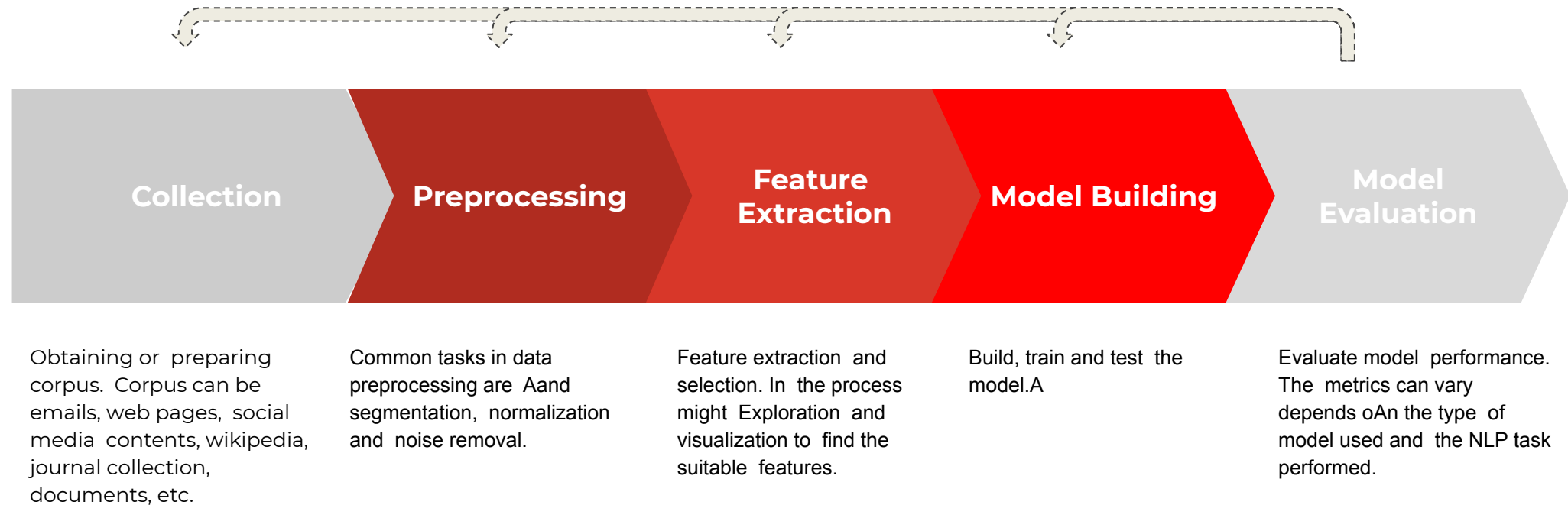


Chapter 2

Text Mining Workflow

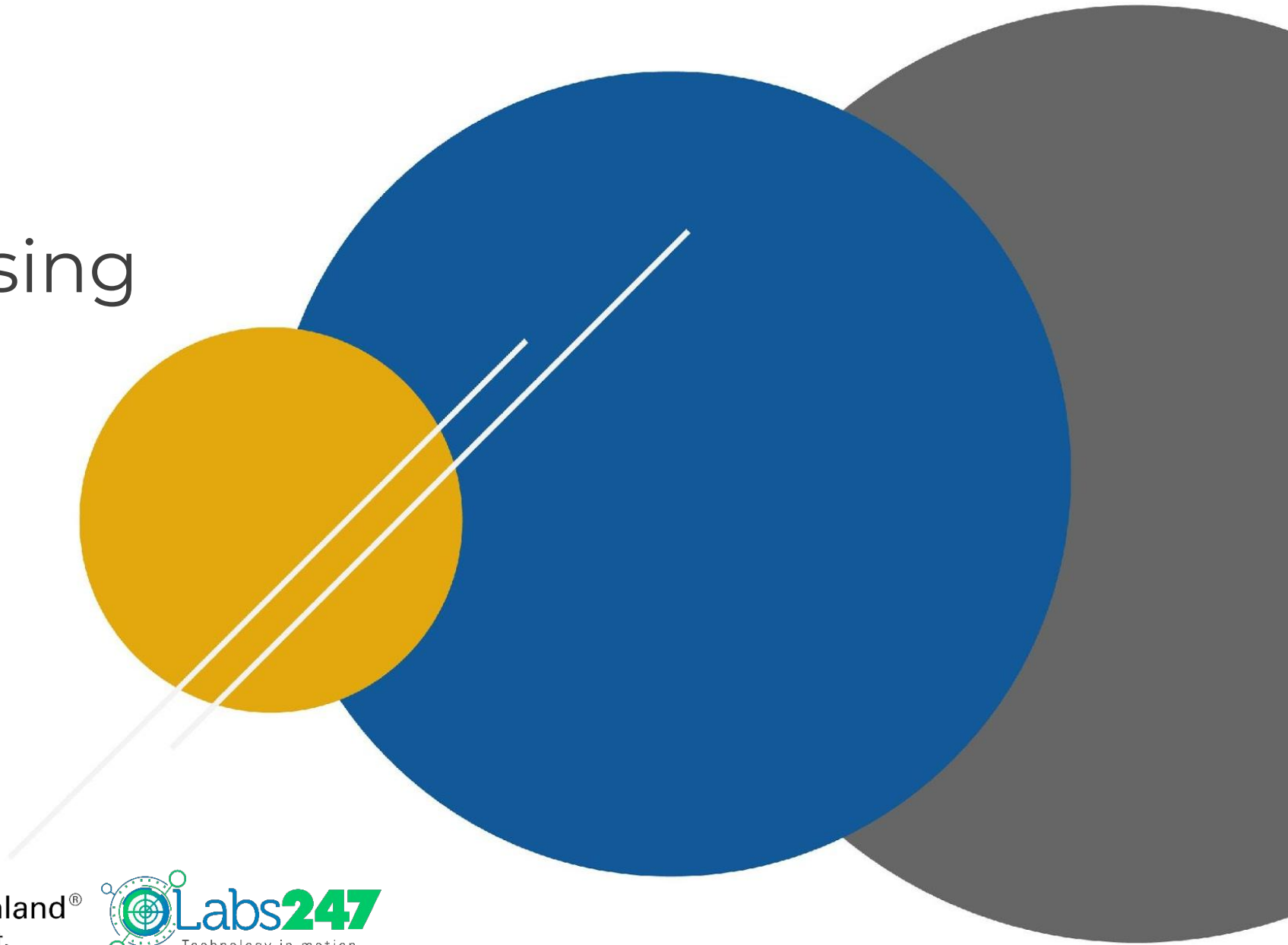


Common Text Mining Workflow



Chapter 03

Text Preprocessing



Text Preprocessing

Common Tasks

Some of the most common tasks in the preprocessing stage are:

- Tokenization
- Noise Removal
 - Stop Words Removal
- Normalization
 - Stemming & Lemmatization
 - Object Standardization



Tokenization

- The process of cutting text into smaller units, called tokens.
- Tokens can be words, keywords, phrases, symbols, or even sentences.
- Challenges in tokenization depends on the type of language.
- Languages such as English is referred to as space-delimited as most of the words are separated from each other by white spaces.
- Languages such as Chinese are referred to as unsegmented as words do not have clear boundaries. Tokenizing unsegmented language sentences requires additional lexical and morphological information.



Noise Removal

Every piece of text that is not relevant to the context and final output can be considered as noise. For example:

- Stopword: commonly used words from a language - for example: are, me, from, in, etc.
- URL, link, tag
- Social media entities (mention, hashtag)
- Punctuation



Noise Removal

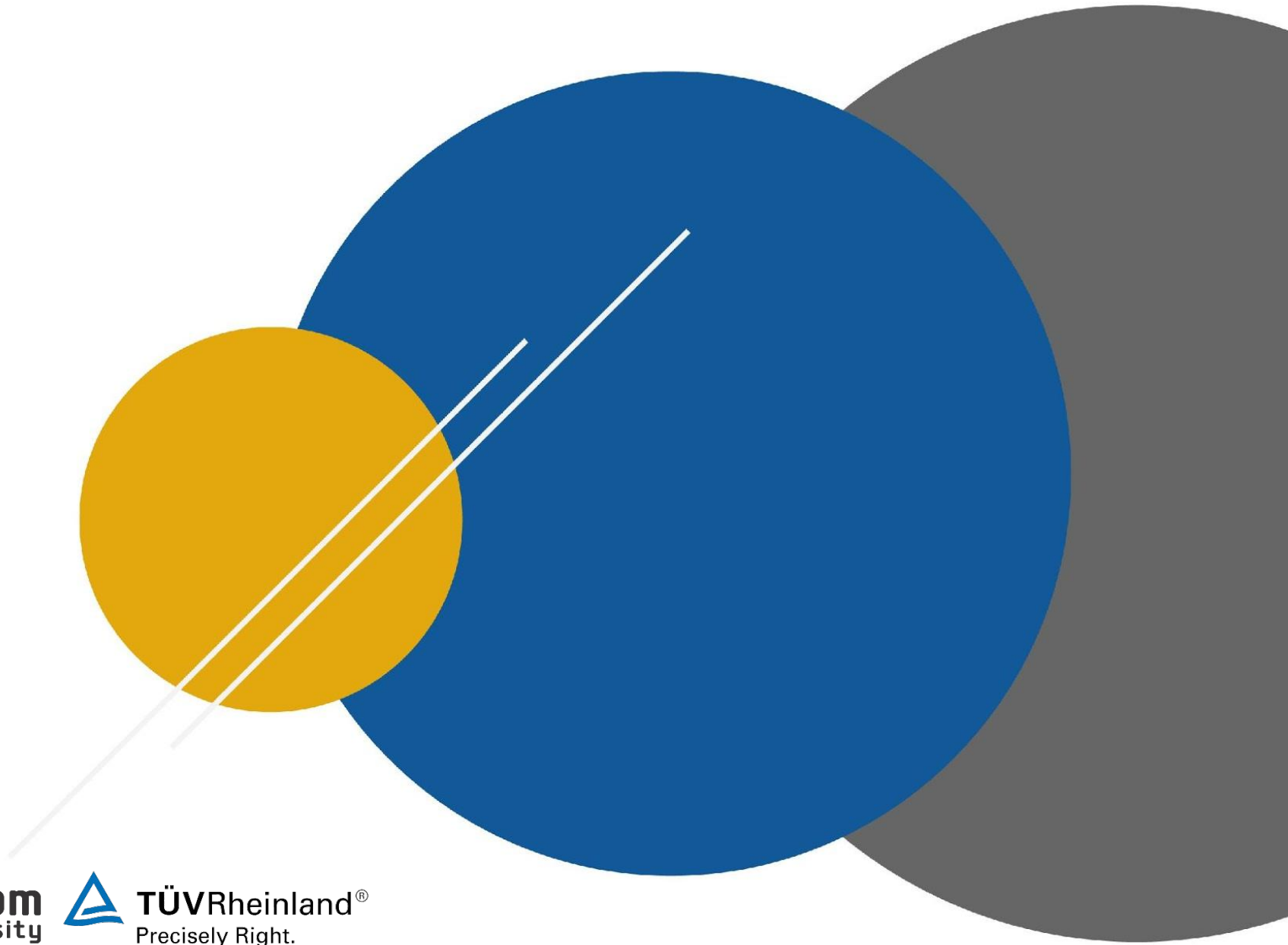
Approach

- Preparing a noise dictionary, and iterate each token (word), eliminating tokens that appear in the dictionary. For example: stop word list.
- Using regular expression (regex)
- Combination of both



LAB 02

Tokenization & Cleansing



Lab Description

- A simple sentence is used as a dataset, for example

```
Budi dan Badu bermain bola di sekolah
```

- Breaking sentences into words using the word_tokenize function in the library
- Elements considered as noise are non alphabetical token, such as numbers and punctuations. It will be deleted by using simple regular expressions
- Besides that, it also learned how to make functions in python



STEP 01

Tokenization and Cleansing

Required code

```
string01 = "Budi dan Badu bermain bola di sekolah"

def tokenize_clean(text):

    #tokenisasi
    tokens = [word.lower() for sent in nltk.sent_tokenize(text) for word
               in nltk.word_tokenize(sent)]

    #clean token from numeric and other character like punctuation
    filtered_tokens = []
    for token in tokens:
        if re.search('[a-zA-Z]', token):
            filtered_tokens.append(token)

    return filtered_tokens

result01 = tokenize_clean(string01)
print(string01)
print(result01)
```



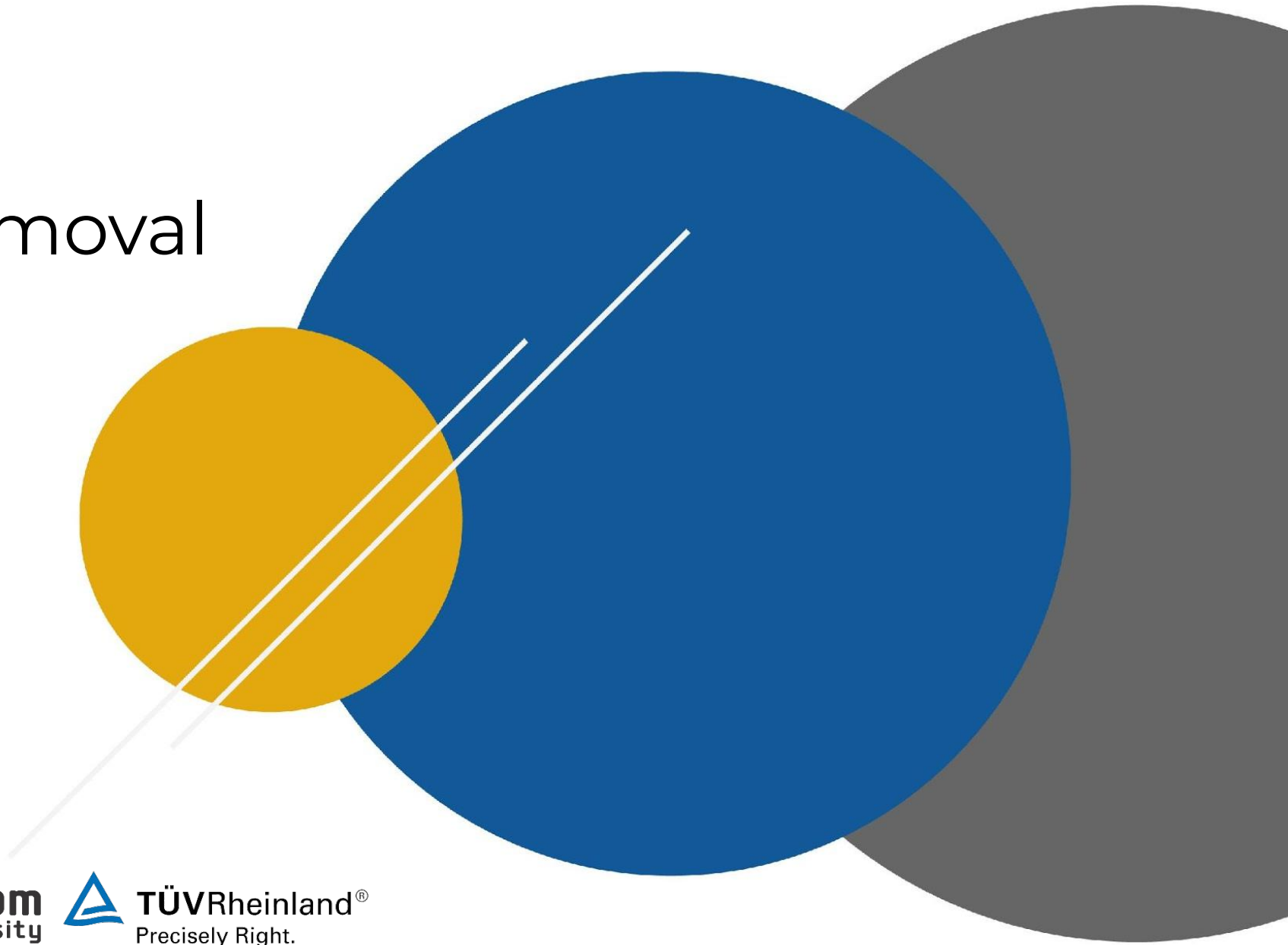
Stop Words Removal

- Stop words are words in a sentence that are considered unimportant, which if omitted will not change the meaning or value of the sentence.
- In most cases, stop word needs to be removed / cleaned so that the results of the analysis are more accurate.
- Stop words are usually words that don't have meaning on its own, for example conjunctions such as 'and', 'then', 'or', etc.
- Stop words depend on the language and domain of the problem to be resolved. There is no universal criteria in determining stop words.



LAB 03

Stop Words Removal



Lab Description

- A simple sentence is used as a dataset, for example

```
Budi dan Badu bermain bola di sekolah
```

- Breaking sentences into words using the word_tokenize function in the library
- Elements considered as noise are non alphabetical token, such as numbers and punctuations. It will be deleted by using simple regular expressions
- Create a function to remove stop words by using the Indonesian stop words reference provided by nltk library



STEP 01

Get Indonesian Stop Words

- Type the following code and click Run

```
stopwords = nltk.corpus.stopwords.words('indonesian')
```



STEP 02

Create remove_stopwords Function

Type the following
code and click Run

```
def remove_stopwords(tokenized_text):  
  
    cleaned_token = []  
    for token in tokenized_text:  
        if token not in stopwords:  
            cleaned_token.append(token)  
  
    return cleaned_token
```



STEP 03

Process Data

Type the following
code and click Run

```
string01 = "Budi dan Badu bermain bola di sekolah"  
string02 = "Apakah Romi dan Julia saling mencintai saat mereka  
berjumpa di persimpangan jalan?"  
  
result01 = tokenize_clean(string01)  
result = remove_stopwords(result01)  
print(string01)  
print(result01)  
print(result)
```



Normalization

- A series of tasks to process text into a form with certain standards
- For example: changing all letters to lowercase, changing numbers into letters, stemming, changing abbreviations into their original words, etc.
- The aim is to improve the quality of the text so that the next process can perform better
- Important processes in text normalization are stemming and lemmatization



Stemming & Lemmatization

- ➡ Stemming and Lemmatization is the process of changing the word into a common base form (stem).
- ➡ Stemming is done by cutting off the end or the beginning of the word, taking into account a list of common prefixes and suffixes in an inflected word. For example:
 - studying**ing** → study
 - studies**es** → studi



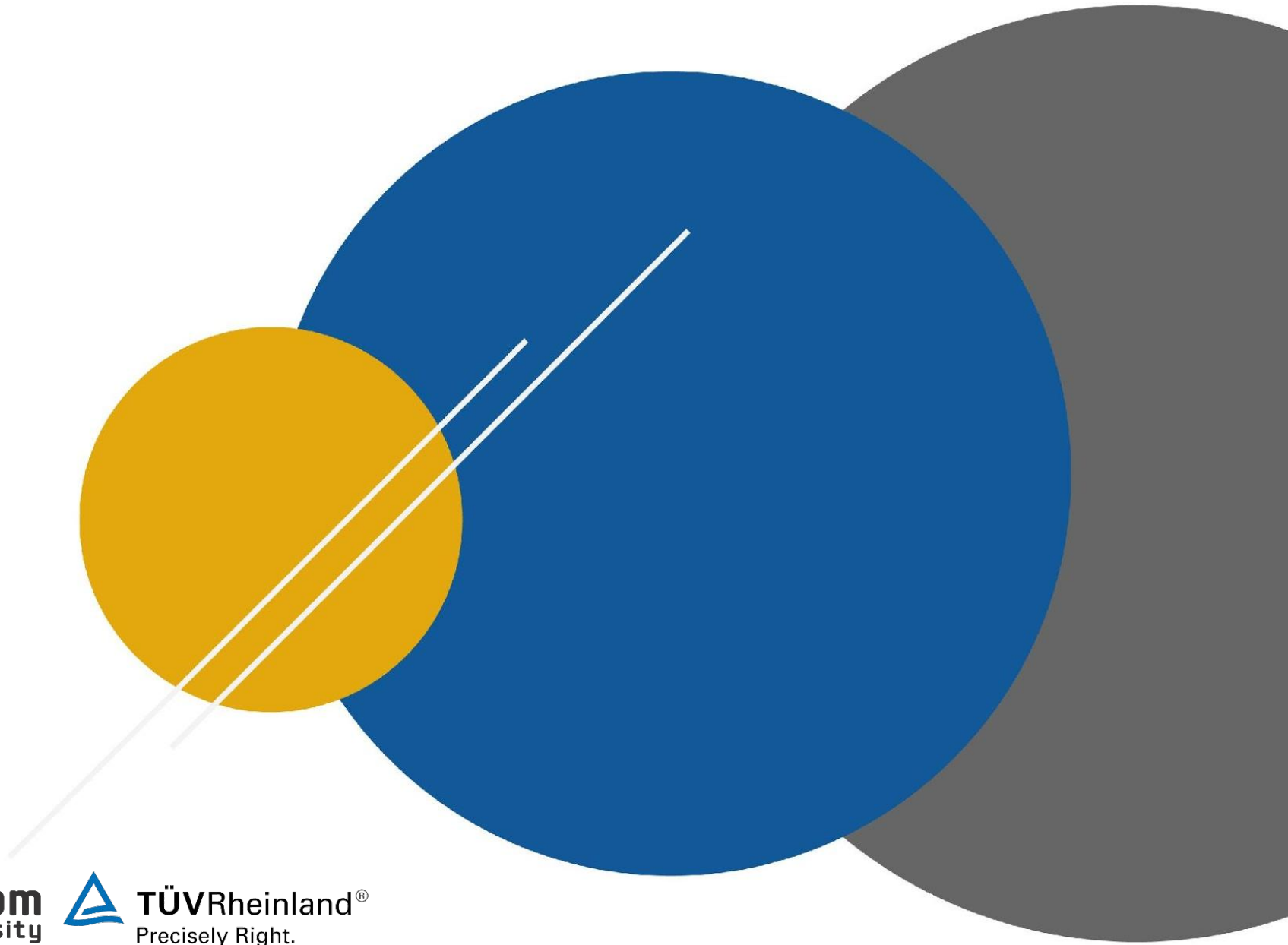
Stemming & Lemmatization

- Lemmatization takes into consideration the morphological analysis of the words. To do so, it is necessary to have detailed dictionaries which the algorithm can look through to link the form back to its lemma. For example:
 - studying → study
 - studies → study
- Lemmatization is usually done for languages that have a change of word form, for example in English: go-went-gone, etc. In bahasa Indonesia, stemming and lemmatization are usually considered the same process.



LAB 04

Stemming and Lemmatization



Lab Description

- A simple sentence is used as a dataset, for example

```
Budi dan Badu bermain bola di sekolah
```

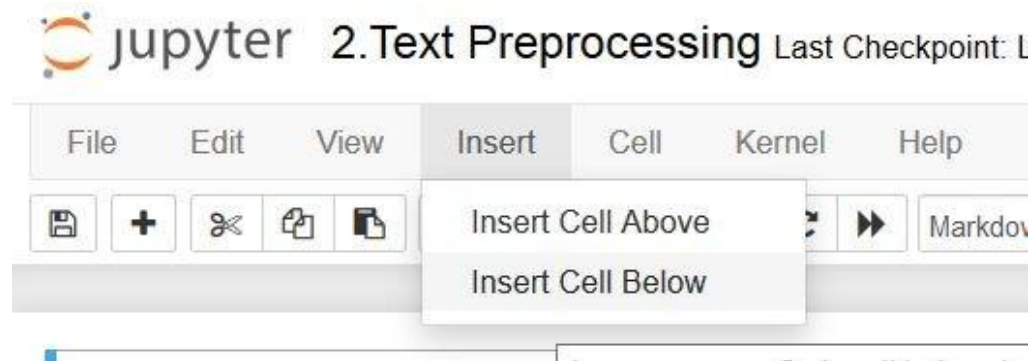
- Create a function to do stemming a words by using StemmerFactory function in Sastrawi package library
- Preprocess data by using function created from previous lab



STEP 01

Insert Cell

- Insert new cell by choosing Insert → Insert Cell



STEP 02

Create Stemming Function

Create stemming_text function by using this code

```
#stem using Sastrawi StemmerFactory
factory = StemmerFactory()
stemmer = factory.create_stemmer()

stems = []
for token in tokenized_text:
    stems.append(stemmer.stem(token))
```



STEP 03

Process Data

Use the following code to process data

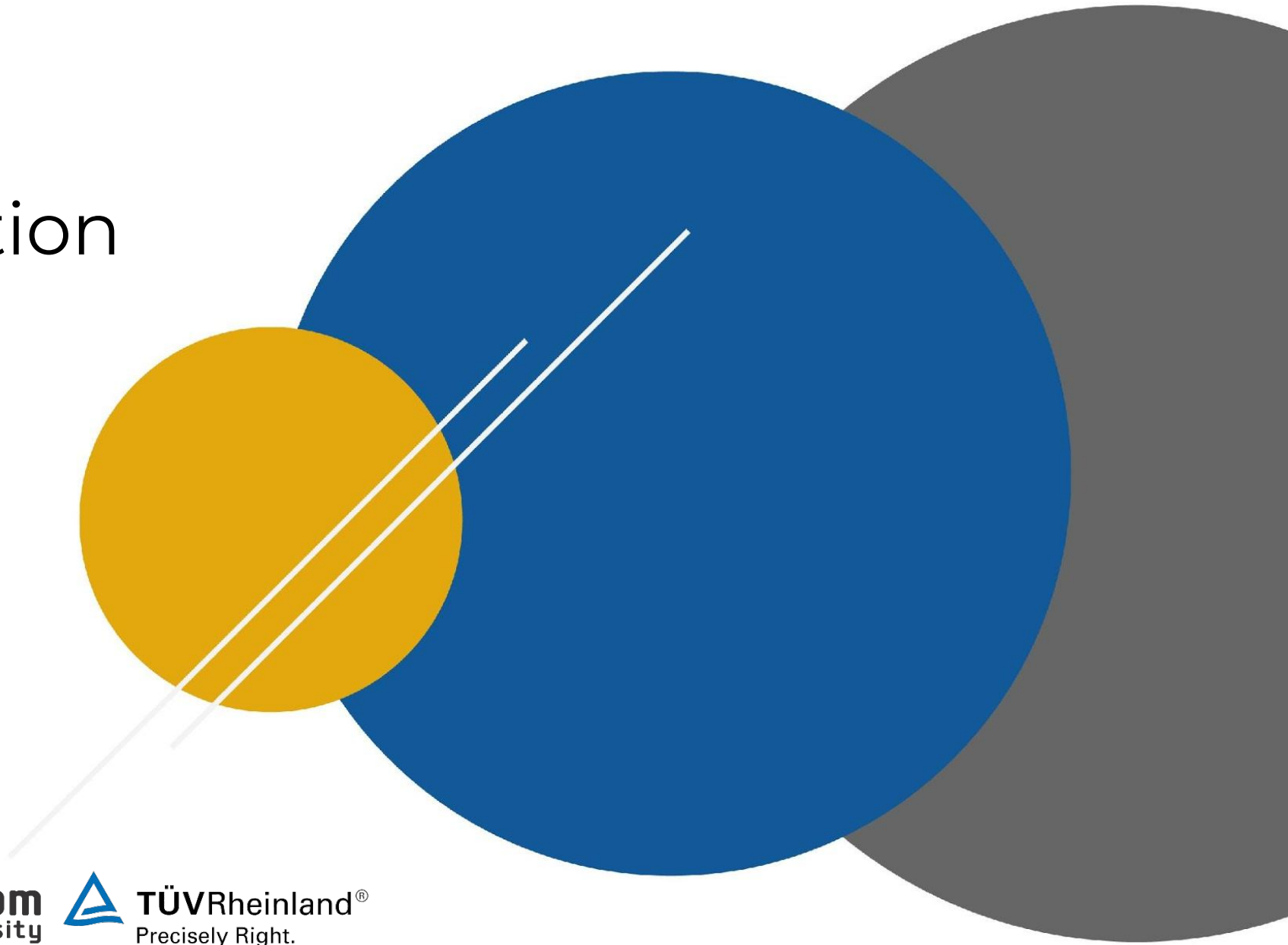
```
# create dataset
string01 = "Budi dan Badu bermain bola di sekolah"
string02 = "Apakah Romi dan Julia saling mencintai saat mereka berjumpa di persimpangan jalan?"

# preprocess and view the result
result01 = tokenize_clean(string01)
result = remove_stopwords(result01)
stem_result = stemming_text(result)
print(string01)
print(result01)
print(result)
print(stem_result)
```



Chapter 4

Feature Extraction



Feature Extraction

- ➡ The process of converting text into a set of features prior to analysis
- ➡ The type of features depend on the model and what method will be used in the mining/machine learning process
- ➡ Some feature engineering techniques in NLP:
 - Syntactic parsing
 - Entity parsing
 - Vectorization



Feature Extraction

Syntactic Parsing

- Syntactic parsing is the process of determining sentence structure based on a certain grammar and lexicon.
- The structure of the sentence includes word level, word class level, phrase level, element level, and clausal level.
- Some important attributes of text syntax are :
 - Dependency Grammar and
 - Part of Speech Tags.



Syntactic Parsing

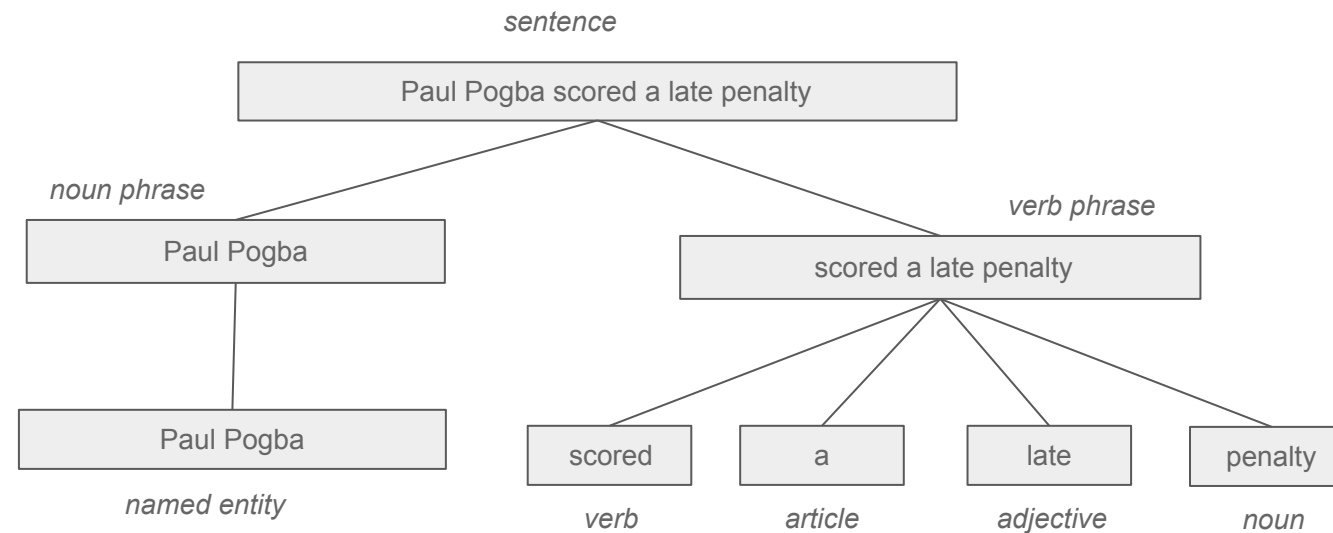
Part of Speech Tagging

- ➡ POS (Part-of-Speech) Tags are a way of categorizing word classes, such as nouns, verbs, adjectives, etc.
- ➡ POS Tagger is an application that is capable of automatically performing POS tag annotations for each word in a document.
- ➡ POS tagging produces a list of tuples, where each tuple is in the form of (words, tags) pairs.
- ➡ Tags are labels that indicate whether a word is a noun, adjective, verb, and so on.
- ➡ There are several POS tagset or POS tag naming methods, the most popular tagset is Penn TreeBank tagset.



Part of Speech Tagging

Simple Example



Part of Speech Tagging

Usage

- POS tagging is usually done before the chunking process, or phrases extraction from a sentence.
- POS Tagging is also used for sentence structure analysis and word sense disambiguation. For example:
 - can - We can help you
 - can - It kept in a can
- By knowing the word class in a sentence, it's easier to determine its meaning.



Entity Parsing

Named Entity Recognition

- The task of identifying the names of all the people, organizations and geographic locations in a text, as well as time, currency and percentage expressions
- Build knowledge from text, by extracting information such as
 - Names (people, organizations, locations, objects, etc.)
 - Temporal expression (calendar dates, times of day, durations, etc.)
 - Numerical expressions (money, percentage, etc.)
- Knowledge base built with NER is widely used in technologies such as smart assistants, machine translation, indexing in information retrieval, classification, automatic summarization, etc.



Named Entity Recognition

Methods

■ Rule Based

- Using a data dictionary consisting of the name of the country, city, company, etc
- Uses predefined language dependent rules based on linguistics which helps in the identification of named entities in a document.
- Constraints: requires the ability to define the rules that are usually carried out by linguists and have a large dependence on the language used.

■ Machine Learning

- Using statistical classification models and machine learning algorithms
- Constraint : require annotated corpora for the domain of interest. The construction of the annotated corpora for a new domain is a time-consuming task and requires effort by the human experts to produce it.

■ Hybrid

- Combine both methods by taking advantage of each method used.



Vectorization

- Transforming text into representations that are 'understood' by machines, which is numeric vector (or array), so that they can be used by various analytics and machine learning algorithms
- There are 2 types of text vectorization, which are:
 - **Bag of words** (or bag of n-grams), represents words as a discrete element of a vector (or array) → element of a bag
 - **Word embeddings** : represent (or embed) words in a continuous vector space in which words with similar meanings are mapped closer to each other. New words in application texts that were missing in training texts can still be classified through similar words.



Vectorization Methods

Type	Vectorization Method	Function	Considerations
Bag of Words	Frequency	Counts term frequencies	Most frequent words not always most informative
	One-Hot Encoding	Binarizes term occurrence (0, 1)	All words equidistant, so normalization extra important
	TF-IDF	Normalizes term frequencies across documents	Moderately frequent terms may not be representative of document topics
Word Embeddings	Distributed Representations	Context-based, continuous term similarity encoding	Performance intensive; difficult to scale without additional tools (e.g., Tensorflow)



Bag of Words

Definition

- Text representation that indicate the appearance of a token / word in a document.
- Called a bag because it does not care about the structure or sequence in the text. *
- The main components of BoW are:
 - Vocabulary or a collection of known words based on text input
 - A measure of the presence of the known words
- The complexity of BoW techniques depends on how to build the vocabulary and the scoring method



Bag of Words

Scoring Method

- Binary: one-hot encoding vector
- Counts: count the number of times a word appears in each document
- Frequency: number of occurrences of words in a document to the total number of words in the document (count / total words)
- TF-IDF: frequency and relevance of words in a corpus



Bag of Words

Example

*It was the best of times,
it was the worst of times,
it was the age of wisdom*

The vocabulary:

{“it”, “was”, “the”, “best”, “of”, “times”, “worst”, “age”, “wisdom”}

If we treat each sentence as separate document, the BoW vectors are:

“it was the best of times”	[1, 1, 1, 1, 1, 1, 0, 0, 0, 0]
"it was the worst of times"	[1, 1, 1, 0, 1, 1, 1, 0, 0, 0]
"it was the age of wisdom"	[1, 1, 1, 0, 1, 0, 0, 1, 1, 0]



One-hot Encoding

Definition

- A representation of categorical variables as binary vectors.
- Each integer value is represented as a binary vector that is all zero values except the index of the integer, which is marked with a 1.
- For example, if we have words {boy, chase, dog, playground} :

boy	{1, 0, 0, 0}
chase	{0, 1, 0, 0}
dog	{0, 0, 1, 0}
playground	{0, 0, 0, 1}



TF-IDF

- TF-IDF : Term Frequency - Inverse Document Frequency
- A way to determine the topic of a document based on the words or terms in the document
- TF-IDF calculates relevance, not just frequency
- The weight calculation in TF-IDF uses a statistical method, which evaluates how important a term is to a document
- The greater the TF-IDF value of a word or term, the rarer the word, the more relevant a word to a document



TF-IDF

Usage

- What is the use of TF-IDF?
 - Categorize text, automatically create tags or keywords for a document.
 - Determine the order of documents in search results (document relevance to a term)
 - Fix / add stop-word lists
- Difference between TF-IDF and sentiment analysis?
 - Sentiment analysis classifies text based on 'positive', 'negative' or 'neutral' opinion values.
 - TF-IDF classifies text based on its contents.



TF-IDF

Term Frequency

- Term Frequency (TF) calculates the frequency of occurrence of a word or term (T) in a document (D).
- There's a possibility that a word has a greater occurrence value on a different document, because the length of the document is different
- TF calculation formula:

$$\text{TF}(t) = (\text{Number of occurrences of the word } t) / (\text{Total number of words})$$



TF-IDF

Inverse Document Frequency

- IDF calculates how important a word or term and document is
- It is known that certain terms, such as "are", "from", and "that", may appear many times but does not have a large influence. Therefore it is necessary to reduce the weight for those words and increase the weight for the rare ones, with the following calculations:

$$IDF(t) = \log_e \left(\frac{\text{Number of documents}}{\text{Number of documents containing the term } t} \right)$$



TF-IDF

Example

- Suppose a document has 100 words with the appearance of the word cat 3 times. TF for cats is:

$$3/100 = 0.03$$

- If there are 10 million documents and the word cat appears in 1000 documents, then the IDF:

$$\log (10,000,000 / 1,000) = 4$$

- The TF-IDF weight for the word cat is

$$0.03 \times 4 = 0.12$$



TF-IDF

Update dan Maintenance

- In most cases, the processed document grow continuously, so the value of tf-idf needs to be updated to include the new documents.
- However, TF-IDF is calculated against a certain corpus, so the tf-idf matrix cannot be updated incrementally.
- Several approaches can be taken to overcome this, including:
 - Perform tf-idf calculations when needed. If there is a new document, the terms in the documents are calculated for the tf-idf value
 - Update regularly, when new documents reach a certain amount / time, the drawback is that there may be terms that will be ignored because they are not yet included in the vocabulary



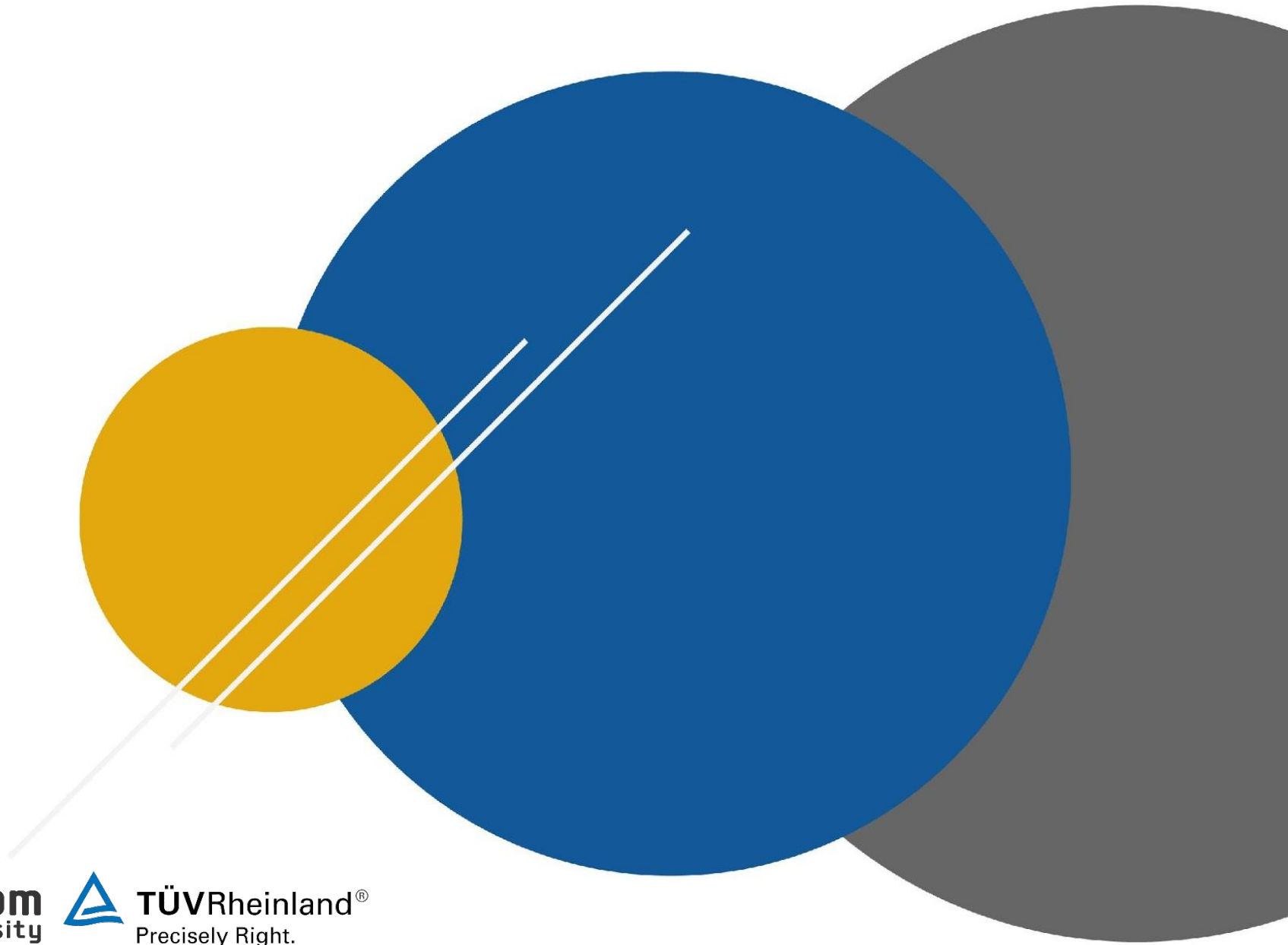
Issues in BoW

- **Vocabulary:** requires good design, especially for managing size because it will affect the sparsity of document representation
- **Sparsity:** The vector formed is a sparse vector, which is a vector with majority elements null or 0. This sparse representation is more difficult and less efficient to model, both in terms of computational (storage complexity and computation time) as well as information (model a little information in a very large space)
- **Meaning:** eliminating the word order results in the loss of context and meaning of words in the text (semantics). Context and meaning are very useful in modeling, for instance to distinguish different meaning of words due to different arrangement, to determine synonyms, and so on



LAB 05

TF - IDF



Lab Description

What you will learn:

- Run TF-IDF function

Requirement :

- `tokenize_and_stem` function from previous labs



STEP 01

Data Input

■ Create dataset

```
from sklearn.feature_extraction.text import TfidfVectorizer

#we will use dummy document for input, with 1 sentence per document

files = []
files.append("Sekelompok ibu dan kaum perempuan duduk beralaskan rumput lapangan sambil fokus menganyam bambu yang ia genggam di tangan.")
files.append("Sebagian besar masyarakat rupanya tak mau melewatkan waktu begitu saja untuk meratapi erupsi.")
files.append("Lombok memang memiliki sejuta pesona yang mampu menyedot perhatian orang untuk datang berwisata.")
files.append("Perempuan yang bergelut di dunia kerelawanan akan belajar caranya bertanggung jawab bagi sendiri dan orang lain.")
files.append("Kami berkoordinasi dan melapor pada posko relawan, kami berkomitmen siap membantu dengan siaga 24 jam")
```



STEP 02

Corpus Preparation

```
#prepare corpus, load it into dictionary
token_dict = {}
i = 0
for t in files:
    filename = "file" + str(i)
    token_dict[filename] = t
    i = i + 1

#use stop words bahasa indonesia from nltk corpus
stopwords = nltk.corpus.stopwords.words('indonesian')

#perform tf-idf vectorization, use tokenize_and_stem we create in previous lab
tfidf = TfidfVectorizer(tokenizer=tokenize_and_stem, stop_words=stopwords)
tfs = tfidf.fit_transform(token_dict.values())
```



STEP 03

TF-IDF Transformation

- We test by using a new sentence, what are the tokens produced and what is the tf-idf value
- Show how many token produced dan tf-idf value

```
str1 = 'Di kejauhan tampak seorang relawan pria dari Lombok sedang berjalan.'  
response = tfidf.transform([str1])  
  
#show result  
feature_names = tfidf.get_feature_names()  
for col in response.nonzero()[1]:  
    print (feature_names[col], ' - ', response[0, col])
```



Word Embeddings

Definition

- A distributed word representation, which is dense, low-dimensional, and real-valued representation of word. A word representation is a mathematical object associated with each word, often a vector.



Word Embeddings

Why

- Word embeddings overcomes BoW problems
 - Represents words in dense real number vectors
 - Includes sentence context, determines the meaning of words by looking at the context
 - Includes information on the similarity of words in their representation: similar words are represented by similar vectors
- Vector values are learned using neural networks, so this word embedding method is often associated with deep learning
- Popular word embedding algorithms: Word2Vec, GloVe



Word Embeddings

Word2Vec

- Word2Vec is a neural network with 2 layers, text as input and vectors as output
- Developed by Mikolov et. al. at Google in 2013
- Determine the meaning of a word by using other words around it (its context)
- When a word w appears in a text, the context of w is the words before and after w (usually in a specified window size)
- For example:

...Menlu yang menghadiri dan membuka *konferensi* Afro-Asia mengharapkan kerjasama yang baik...

...bahwa tema yang diusung dalam *konferensi* tahun ini adalah penguatan Ekonomi...

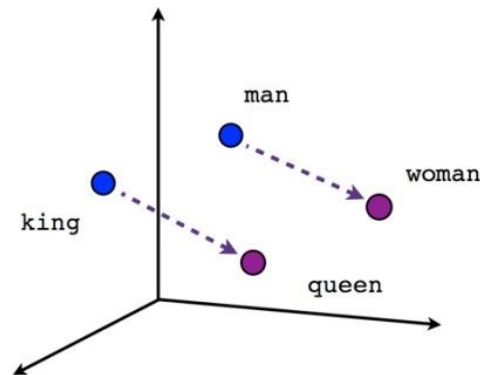
...Wagub membuka Seminar Nasional dan *Konferensi* Daerah Ikatan Apoteker Indonesia...

- This context is called local context

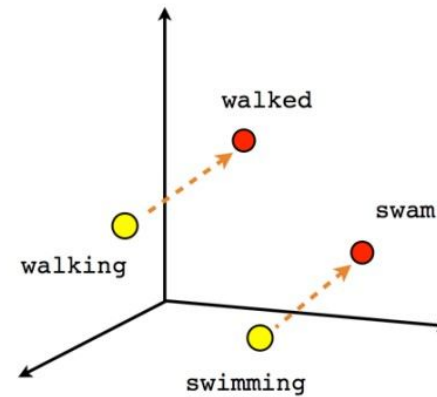


Wor2Vec

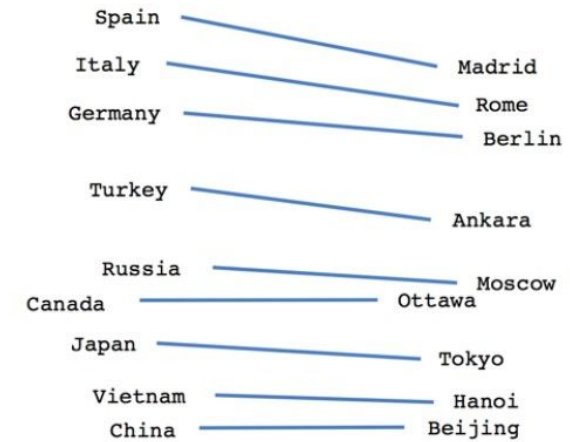
Word Similarity



Male-Female



Verb tense



Country-Capital



Word2Vec

Advantage

- The key benefit of the approach is that high-quality word embeddings can be learned efficiently (low space and time complexity), allowing larger embeddings to be learned (more dimensions) from much larger corpora of text (billions of words).



Google Word2Vec

- Pre-trained model
- 300 dimensional vector
- 3 million words and phrases
- Dataset : Google News (300 billion words)
- Further info:
<https://code.google.com/archive/p/word2vec/>



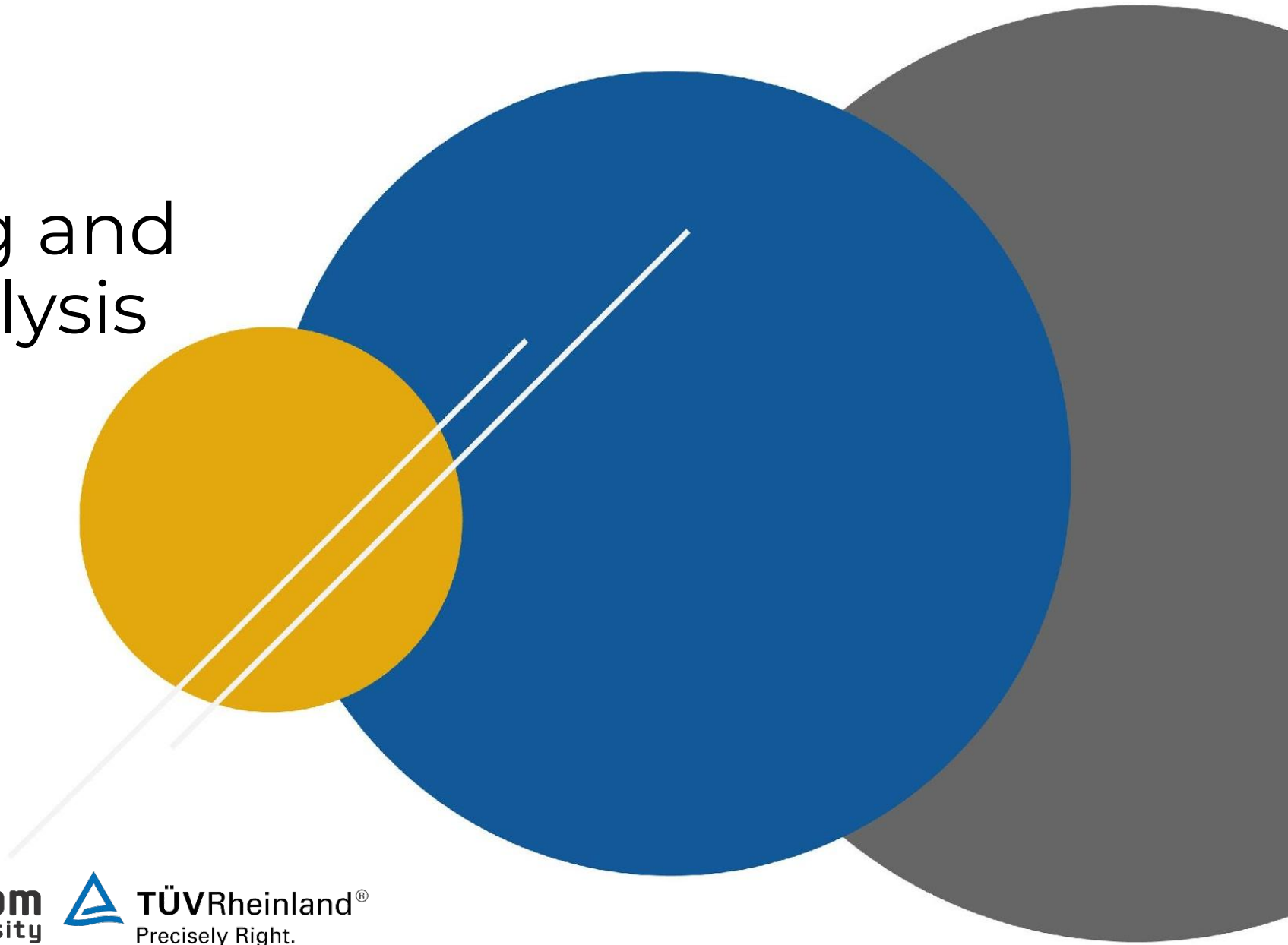
Other Word Embedding

- GloVe
- FastText
- LDA2Vec
- StarSpace
- Poincare embeddings



Chapter 5

Opinion Mining and Sentiment Analysis



Definition

- Information types in text : facts and opinions.
 - Facts : objective expressions about something.
 - Opinions : subjective expressions that describe people's sentiments, appraisals, and feelings toward a subject or topic.
- Sentiment analysis : analysis process to obtain subjective information of a topic.



Sentiment Analysis

Use Case Examples

- Opinions in the social and geopolitical context
- Business and e-commerce applications, such as product reviews and movie ratings
- Predicting stock prices based on people opinion about the companies and resources
- Determine areas of product that need to be improved by summarizing product reviews
- Customer preference



Opinion Representation

- **Opinion holder:** Whose opinion is this?
- **Opinion target:** What is this opinion about? e.g., a product, a service, an individual, an organization, an event, or a topic → also called entity. An entity can have many feature (aspect).
- **Opinion context:** Under what situation (e.g., time, location) was the opinion expressed?
- **Opinion sentiment:** What does the opinion tell us about the opinion holder's feeling ? Positive, negative and neutral are called opinion orientation (also called sentiment orientation or polarity)
- Liu (2012) formulated the formal definition : An opinion is a quadruple, (g, s, h, t), where g is the opinion (or sentiment) target, s is the sentiment about the target, h is the opinion holder, and t is the time when the opinion was expressed.



Sentiment Analysis

Level

- Document-level Sentiment Analysis : determine whether a whole document, message, etc, is overall positive or negative
- Sentence-level Sentiment Analysis : determine the sentiment of each sentence within the document
- Aspect or Topic based Sentiment Analysis : identify not only positive or negative sentence, but also the specific topic/feature that is being referred as positive or negative. There may be more than 1 aspects in a sentence :
 - e.g : I love the display of the new phone but the battery life is terrible.



Sentiment Analysis

Process

- Opinion Mining
 - Entity extraction and categorization
 - Aspect extraction and categorization
 - Opinion holder extraction and categorization
 - Time extraction and standardization
 - Sentiment classification
 - Opinion quadruple generation: Produce all opinion (g, s, h, t) expressed in document d based on the results of the above tasks.
- Opinion Summarization
 - Opinions are subjective. An opinion from a single person (unless a VIP) is often not sufficient for action. We need opinions from many people, and thus the need for opinion summarization.



Document Sentiment Classification

Techniques

- Supervised learning : any existing supervised learning methods can be applied; e.g. Bayesian classifications, Support Vector Machine, etc.
- Unsupervised learning : using opinion words and phrases. Liu (1992) explain the algorithm which contains 3 steps:
 - Extract phrase containing adjective or adverbs
 - Estimate the semantic orientation/polarity
 - Given a review, the algorithm computes the average opinion orientation of all phrases in the review, and classifies the review as recommended if the average is positive, not recommended otherwise.



Sentiment Analysis

Important Features

- Terms and their frequency : Individual words or n-grams and their frequency counts. Word positions may also be considered. The TF-IDF weighting scheme may be applied too. These features have been shown quite effective in sentiment classification
- Part of speech : adjectives may be treated as special features
- Opinion words and phrases : words that are commonly used to express positive or negative sentiments. For example, beautiful, wonderful, good are positive opinion words, and bad, poor, and terrible are negative opinion words.
- Negations : important because their appearances often change the opinion orientation
- Syntactic dependency : word dependency based features generated from parsing or dependency trees



Sentiment Analysis

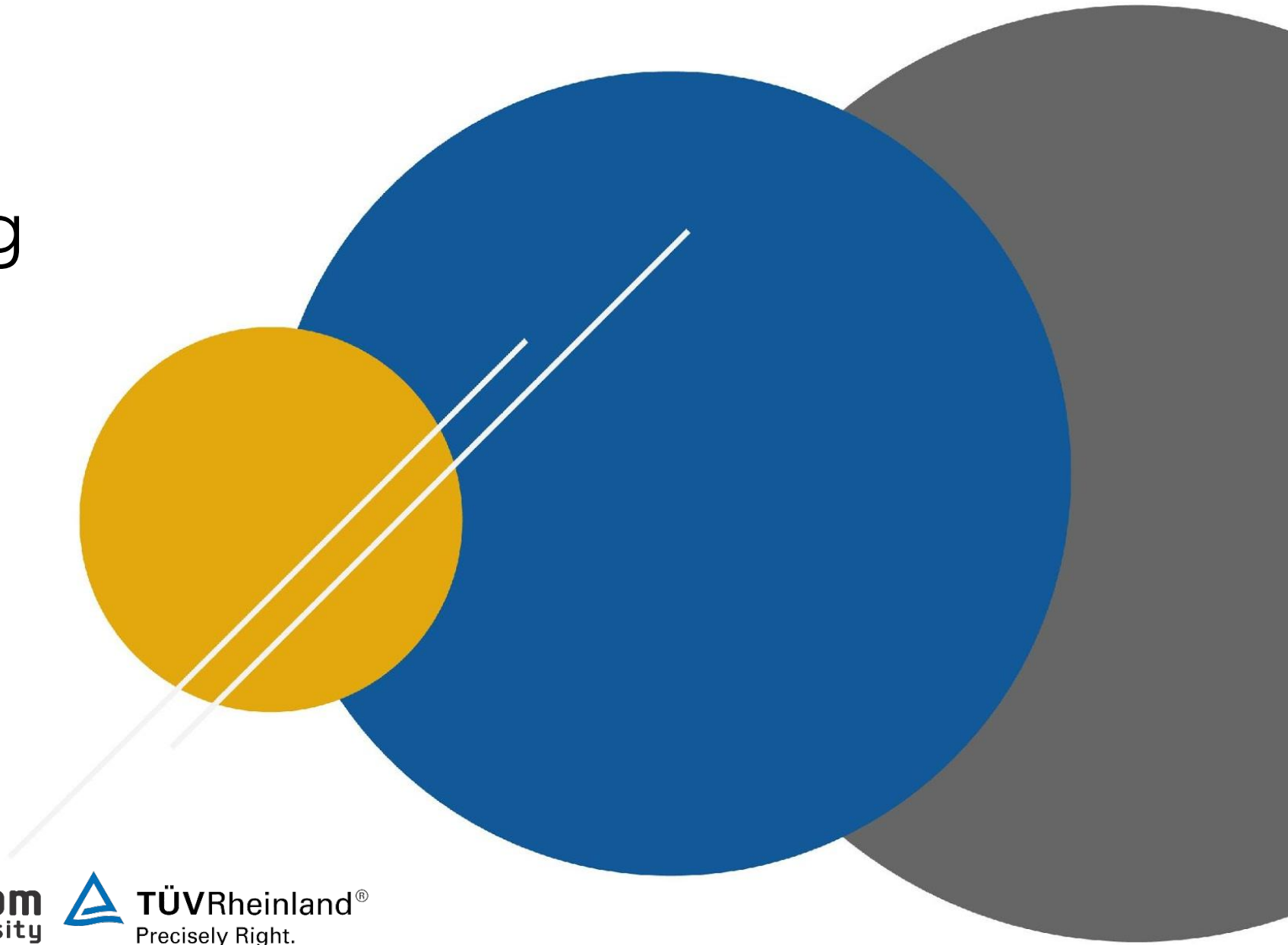
Challenges

- A positive or negative sentiment word may have opposite orientations in different application domains.
- A sentence containing sentiment words may not express any sentiment. Question sentences and conditional sentences are two important types, e.g., “Can you tell me which camera is good?” and “If I can find a good camera in the shop, I will buy it.”
- Not all conditional or interrogative sentences express no sentiments, e.g., “Does anyone know how to repair this terrible printer” and “If you are looking for a good car, get Toyota.”
- Sarcastic sentences with or without sentiment words are hard to deal with, e.g., “What a great car! It stopped working in two days.”
- Many sentences without sentiment words can also imply opinions. e.g. “This washer uses a lot of water” implies a negative sentiment.



Chapter 6

Topic Modelling



Topic Modelling

- Topic modeling is an unsupervised machine learning way to organize text information such that related pieces of text can be identified.
- Topic Modelling is basically a document clustering where documents and words are clustered simultaneously
- Topic modelling problem :
 - Known : Text/document collections (corpus) and The number of topics
 - Unknown : The actual topics and topic distribution in each document
- Topic modelling used in:
 - Discovering hidden topical patterns that are present across the collection
 - Annotating documents according to these topics
 - Using these annotations to organize, search and summarize texts



Topic Modelling

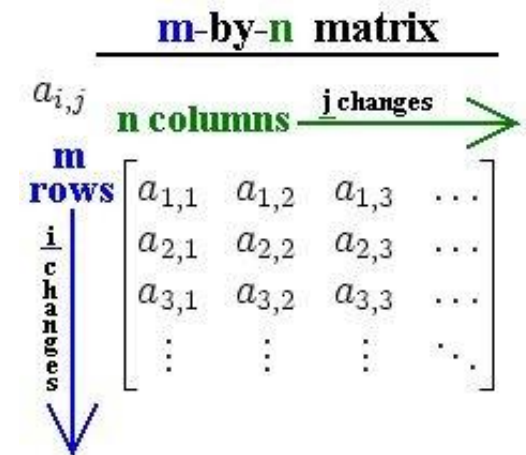
- Basic assumptions:
 - A document consists of a mixture of topics
 - A topic is a collection of words
- Topic = latent semantic concepts
- Different Approaches
 - Latent Semantic Analysis/Indexing (LSA/LSI) → linear algebra
 - Probabilistic Latent Semantic Analysis (PLSA) → probabilistics
 - Latent Dirichlet Allocation (LDA) → probabilistics



Latent Semantic Analysis

- Decomposing documents-words matrix into documents-topics and topics-words by using Singular Value Decomposition (SVD)
- Given m documents and n words in our vocabulary, we can construct an m -by- n matrix $A \rightarrow$ sparse word-document co-occurrence matrix
 - Simplest form of LSA uses raw count, where $a_{i,j}$ is the number of times the j -th word appeared in the i -th document
 - More advanced LSA often uses TF-IDF to for $a_{i,j}$ value
- SVD decompose matrix A into 3 matrices where:
 - A is an $m \times n$ matrix
 - U is an $m \times m$ orthogonal matrix
 - S is an $n \times n$ diagonal matrix
 - V is an $n \times n$ orthogonal matrix

$$A = USV^T$$

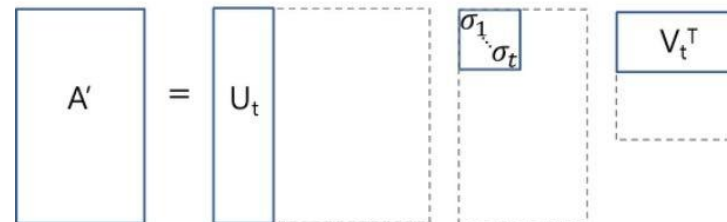


Latent Semantic Analysis

- Since A most likely sparse, we need to perform dimensionality reduction using truncated SVD

$$A \approx U_t S_t V_t^T$$

- This will keep the t most significant dimensions in the transformed space.

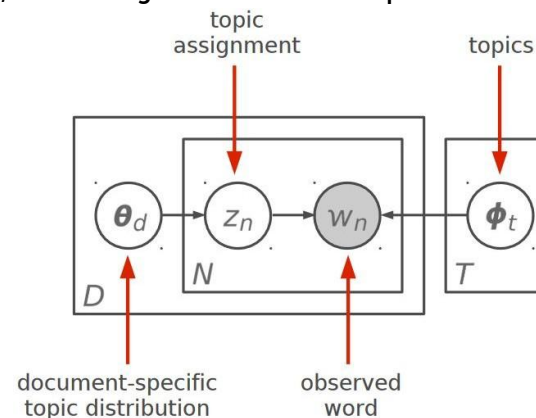


- LSA is quick and efficient, but has some shortcomings:
 - Lack of interpretable embeddings
 - Need for really large set of documents and vocabulary to get accurate results
 - Less efficient representation



Probabilistic Latent Semantic Analysis

- PLSA uses probabilistic method instead of SVD
- The basic idea : find probabilistic model $P(D,W)$ such that for any document d and word w , $P(d,w)$ corresponds to that entry in the document-term matrix.
- PLSA assumptions:
 - given a document d , topic z is present in that document with probability $P(z|d)$
 - given a topic z , word w is drawn from z with probability $P(w|z)$
- As its name implies, PLSA just adds a probabilistic treatment of topics and words on top of LSA.



PLSA

Limitations

- PLSA is more flexible than LSA, but still has some limitations :
 - The number of parameters grows linearly with the size of training documents → The model is prone to overfitting
 - Not a well-defined generative model - no way of generalizing to new, unseen documents



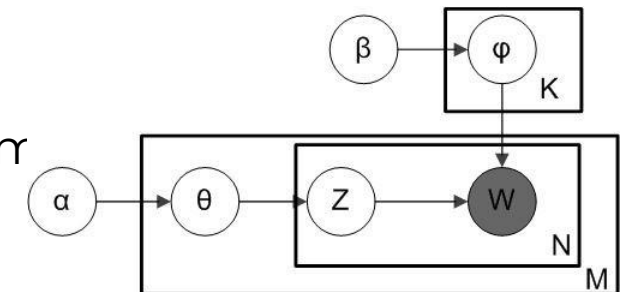
Latent Dirichlet Allocation

- LDA is a Bayesian version of PLSA. It uses dirichlet priors for the document-topic and word-topic distributions, leading to better generalization.
- Dirichlet : a probability distribution but it is not sampling from the space of real numbers. Instead it is sampling over a probability simplex.
- Probability simplex : a group of numbers that add up to 1. For example:
 - (0.6, 0.4)
 - (0.1, 0.1, 0.8)
 - (0.05, 0.2, 0.15, 0.1, 0.3, 0.2)
- The numbers represent probabilities over K distinct categories. In the above examples, K is 2, 3, and 6 respectively.



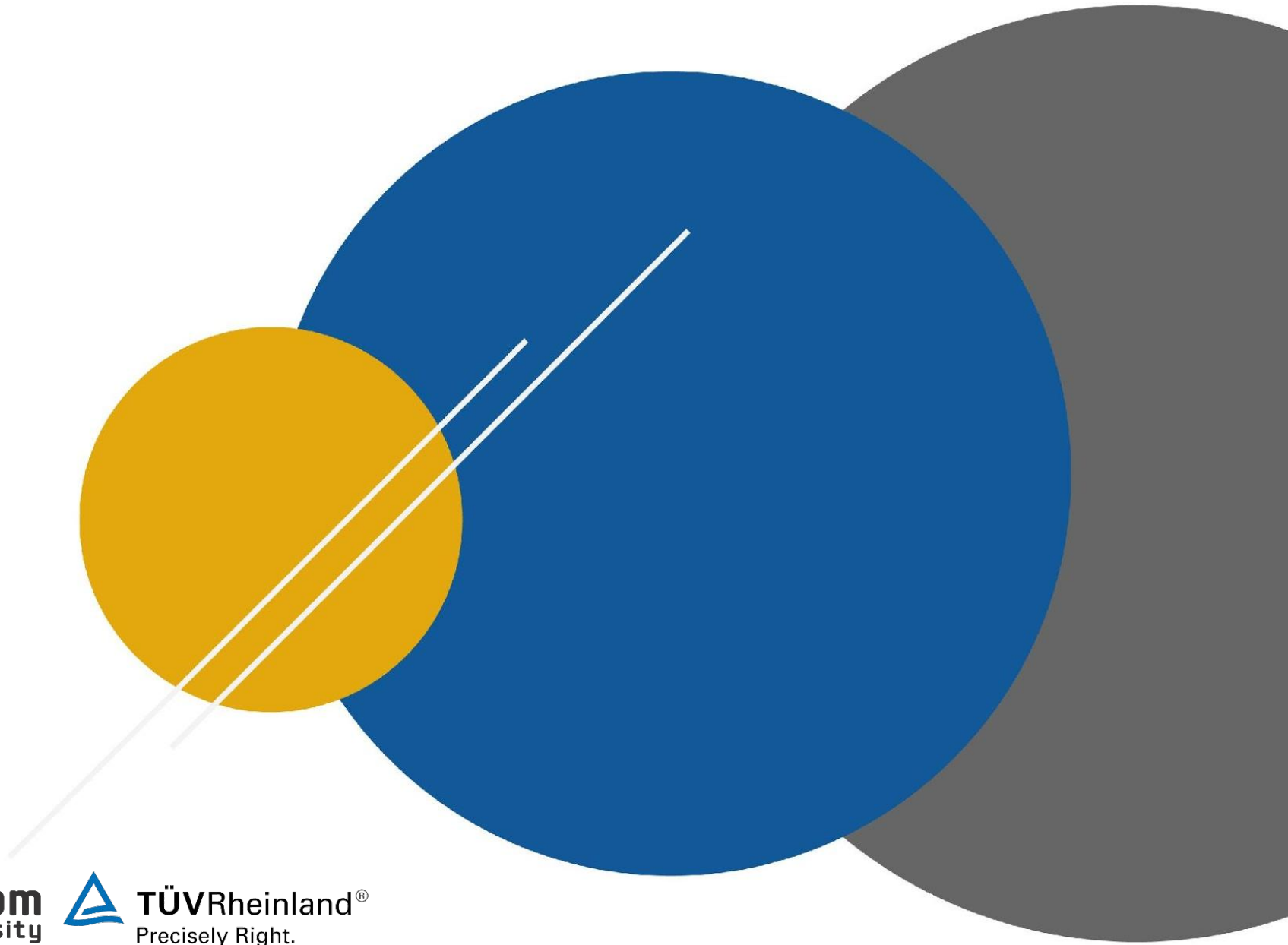
Latent Dirichlet Allocation Model

- From a dirichlet distribution $\text{Dir}(\alpha)$, draw a random sample representing the topic distribution θ of a particular document.
- From θ , we select a particular topic Z based on the distribution.
- From another dirichlet distribution $\text{Dir}(\beta)$, select a random sample representing the word distribution ϕ of the topic Z . From ϕ , we choose the word w .
- LDA typically works better than pLSA because it can generalize to new documents easily.
- Some limitations:
- Needs relatively large memory and processing time
- The model is difficult to explain



Chapter 7

Wrapping It All Together



Text Clustering

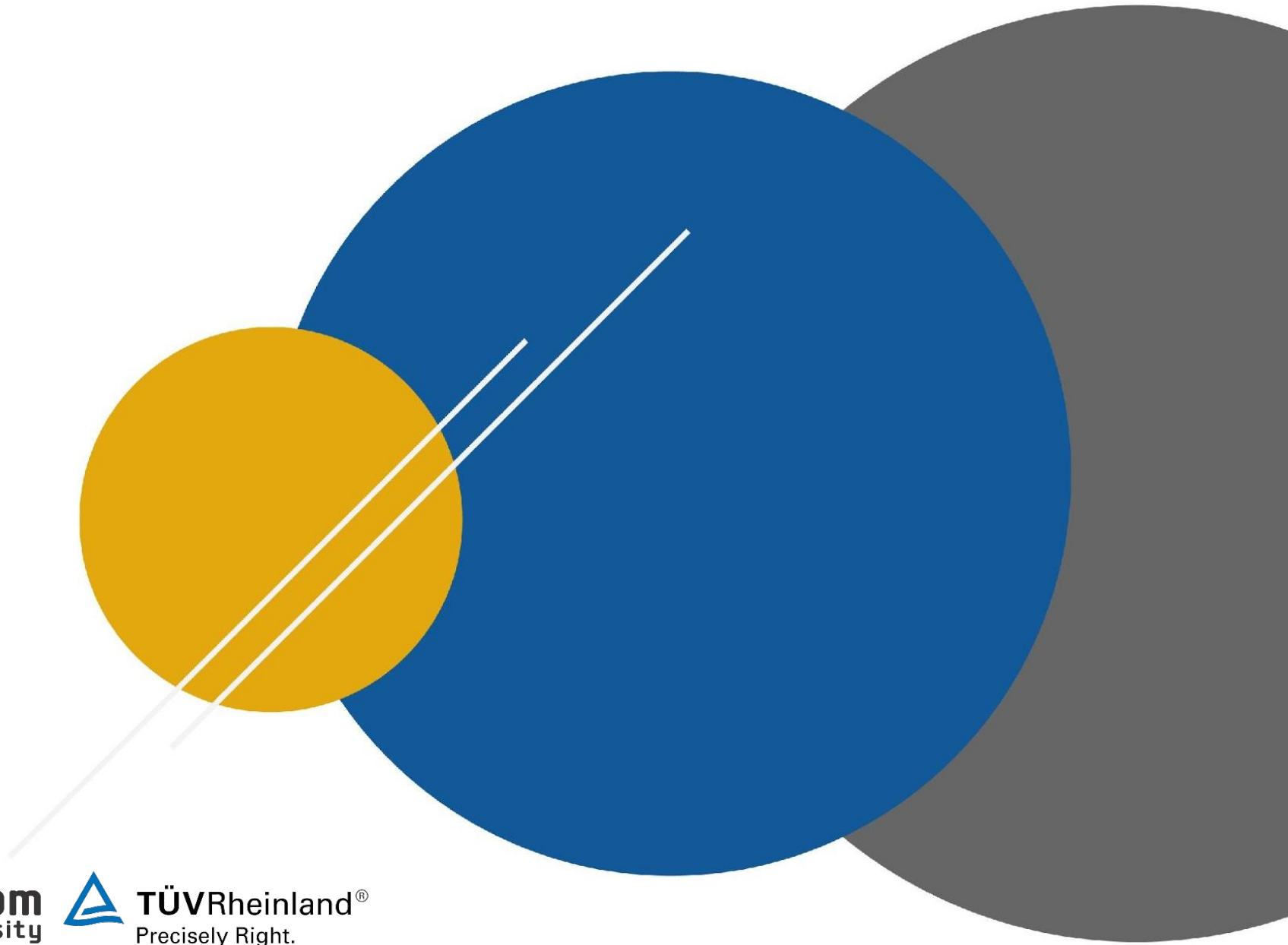
Process Flow

- We will demonstrate the end-to-end process by performing document clustering.
- The process flow that will be used are as follow:
 - Text preprocessing, including text cleanup and text normalization
 - Vector Representation / Feature Extraction : using TF-IDF
 - Building model : using K-Means
 - Visualization
 - Model evaluation



LAB 06

Text Clustering



Lab Description

What you will learn:

- How to create a document clustering program by using real dataset
- Implement tokenization, stemming and cleansing
- K-Means implementation
- Visualization by using matplotlib



STEP 01

Library

- Import all required library and click Run

```
import nltk
from Sastrawi.Stemmer.StemmerFactory import StemmerFactory

import re
import pandas as pd
from bs4 import BeautifulSoup
import matplotlib.pyplot as plt

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.cluster import KMeans
from sklearn.externals import joblib
from sklearn.manifold import MDS
```



STEP 02

Data Input

- Type the following code and click Run

```
#load titles
titles = open('Judul Berita.txt').read().split('\n')

#load articles
article = open('Berita.txt', encoding="utf8").read().split('BERHENTI
DISINI')
```

- Show sample data

```
len(titles)  titles[:5]

article[:5]  len(article)
```



STEP 03

Parsing Articles

- Parsing articles from html format using beautifulsoup package

```
article_clean = [] for text in article:  
    text = BeautifulSoup(text, 'html.parser').getText()  
    article_clean.append(text)  
article = article_clean  
print(article)
```



STEP 04

Tokenization dan Stemming

- Do tokenization, stemming and cleansing, like in the Lab 03

```
def tokenize_and_stem(text):  
    #tokenization and change to lowercase  
    tokens = [word.lower() for sent in nltk.sent_tokenize(text) for word in  
nltk.word_tokenize(sent)]  
  
    #clean token from number and non alphabetical character such as  
punctuation, etc.  
    filtered_tokens = []  
    for token in tokens:  
        if re.search('[a-zA-Z]', token):  
            filtered_tokens.append(token)  
  
    #clean stop words  
    stopwords = nltk.corpus.stopwords.words('indonesian')  
    cleaned_token = []  
    for token in filtered_tokens:  
        if token not in stopwords:  
            cleaned_token.append(token)  
    ...
```



STEP 05

Tokenization dan Stemming (cont.)

- Do tokenization, stemming and cleansing, like in the Lab 03 (cont.)

```
...  
  
#stem using Sastrawi StemmerFactory  
factory = StemmerFactory()  
stemmer = factory.create_stemmer()  
stems = [stemmer.stem(t) for t in cleaned_token]  
  
return stems
```

- Show sample data

```
vocab_frame = pd.DataFrame({'words': totalvocab_tokenized}, index =  
totalvocab_stemmed)  
print('ada ' + str(vocab_frame.shape[0]) + ' kata di vocab_frame')  
print(vocab_frame.head())
```



STEP 06

TF-IDF

■ Calculate TF-IDF matrix

```
tfidf_vectorizer = TfidfVectorizer(max_df=0.8,  
max_features=200000,min_df=0.2,  
                                use_idf=True, tokenizer=tokenize_and_stem,  
                                ngram_range=(1,3))  
  
#fit the vectorizer to article  
tfidf_matrix = tfidf_vectorizer.fit_transform(article)
```

■ Show matrix

```
print(tfidf_matrix.shape)  
print(tfidf_matrix)
```



STEP 07

K-Means Modelling

- Do K-Means Modeling, in this case we use the number of clusters = 3

```
num_clusters = 3
km = KMeans(n_clusters=num_clusters, random_state=1000)
km.fit(tfidf_matrix)
```

- Create DataFrame with the format: sequence - title - cluster

```
#urutan
ranks = [i for i in range(1, len(titles)+1)]
#cluster with k-means
clusters = km.labels_.tolist()

news = { 'title': titles, 'rank': ranks, 'article': article, 'cluster': clusters }
frame = pd.DataFrame(news, index = [clusters] , columns = ['rank', 'title', 'cluster'])

#show dataframe
print(frame)
frame['cluster'].value_counts()
```



STEP 08

Data Exploration

- Displays the results of clustering and top term per cluster to determine the label

```
print("Top terms per cluster:")
#sort cluster centers based on its proximity to its centroid
order_centroids = km.cluster_centers_.argsort()[:, :-1]

for i in range(num_clusters):
    print("Cluster %d words:" % i, end='')

    for ind in order_centroids[i, :6]: #replace 6 with n words per cluster
        print(' %s' % vocab_frame.ix[terms[ind].split('
')]
        .values.tolist()[0][0].encode('utf-8', 'ignore'), end=',')
    print() #add whitespace
    print() #add whitespace

    print("Cluster %d titles:" % i, end='')
    for title in frame.ix[i]['title'].values.tolist():
        print(' %s,' % title, end='')
        print() #add whitespace
    print()
```



STEP 09

Visualization

- Visualize of the results of clustering with MDS

```
similarity_distance = 1 - cosine_similarity(tfidf_matrix)

mds = MDS(n_components=2, dissimilarity="precomputed",
random_state=1) mds.fit_transform(similarity_distance)
# shape (n_components, n_samples)
xs, ys = pos[:, 0], pos[:, 1]
```

- From step 06 it can be seen that the 3 clusters formed are: economy, sports and crime. Color set and label cluster

```
#set color with dictionary
cluster_colors = {0: '#1b9e77', 1: '#d95f02', 2: '#7570b3'}

#dictionary for cluster name (chart legend)
cluster_names = {0: 'Olahraga', 1: 'Ekonomi', 2: 'Kriminal'}
```



STEP 09

Visualization

- Set matplotlib to display charts inline

```
matplotlib inline
```

- Type the following code

```
#create data frame that has the result of the MDS plus the cluster
numbers and titles
df = pd.DataFrame(dict(x=xs, y=ys, label=clusters, title=titles))
groups = df.groupby('label')

# set up plot
fig, ax = plt.subplots(figsize=(17, 9)) # set size
# ax.margins(0.05) # Optional, just adds 5% padding to the
autoscaling
```

