Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

472867-HS2025-0: NLP and Text Mining

03 – Text Mining with WE

murat.sariyar@bfh.ch

Video Resources

Suggestions:

https://www.youtube.com/watch?v=5HQCNAsSO-s (Sentiment analysis)

https://www.youtube.com/watch?v=x1u5TotQ0G0 (TF-IDF)

https://www.youtube.com/watch?v=hVM8qGRTaOA (Word embeddings)

A little bit boring but high-quality content:

https://www.youtube.com/watch?v=viZrOnJclY0&t=80s (Word embeddings)

Goals for today

You are able

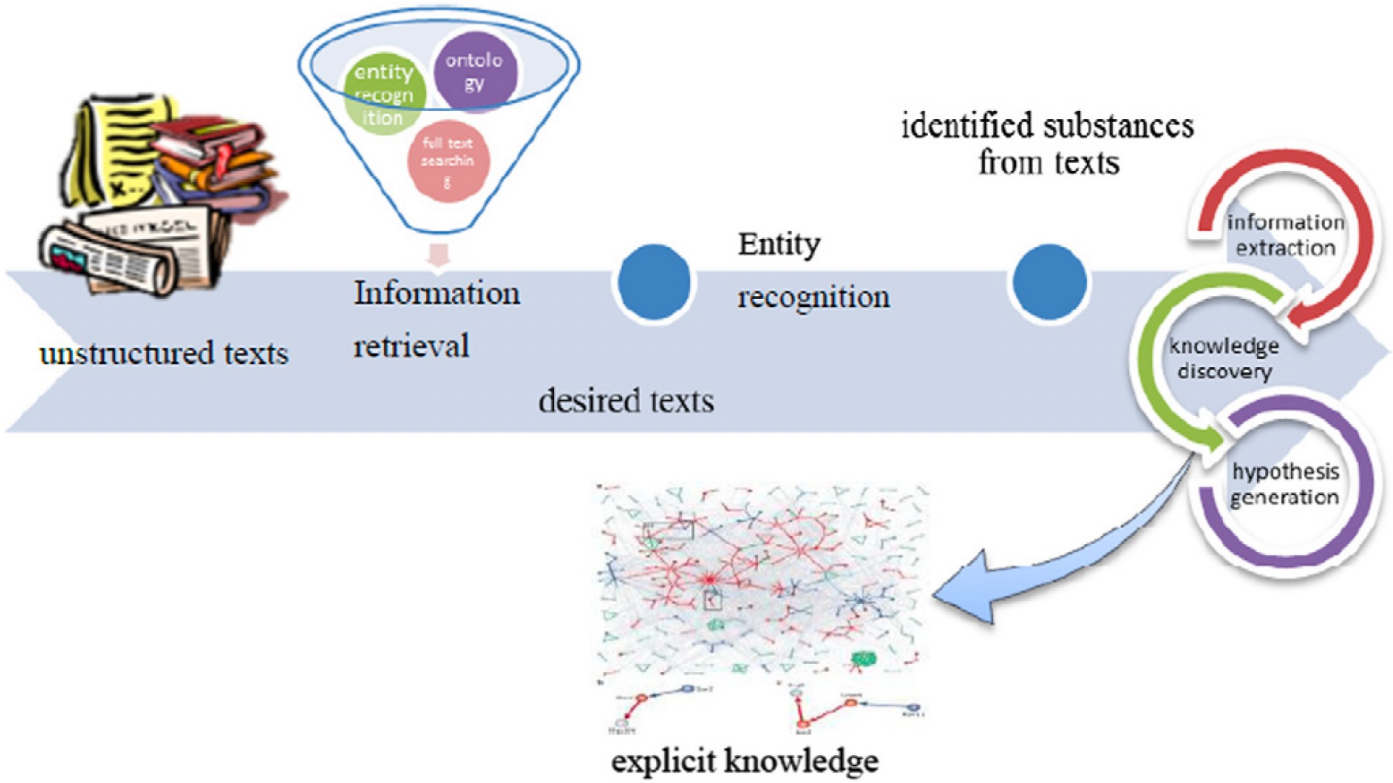
to define what text mining essentially is

to define what sentiment analysis essentially is

to apply supervised sentiment analysis using word embeddings

Text Mining in Medicine

Biomedical text mining



Biomedical text mining

The goal of text mining is to derive implicit knowledge that hides in unstructured text and present it in an explicit form. General tasks of biomedical text mining include *information retrieval*, *NER* and *relation extraction, knowledge discovery* and *hypothesis generation*.

Examplary applications: identifying malignant tumor related biomedical mentions (genes, proteins, etc.), finding relationships among biomedical entities (protein–protein, gene–disease, etc.), extracting knowledge from text and generating hypotheses.

Current biomedical named entity recognition technique falls into three major categories: dictionary-based approaches, rule-based approaches and machine learning approaches.

https://www.sciencedirect.com/science/article/pii/S1532046412001712

Concrete Use Cases for Text Mining

Adverse event surveillance

Decision support alerts

Automatic summaries

ICD-10 code mapping

SNOMED CT code mapping

…

NER and Entity linking

Named-Entity Recognition: is a subtask of information extraction that seeks to locate and classify named entities mentioned in unstructured text into pre-defined categories such as person names, organizations, locations, medical codes, time expressions, quantities, monetary values, percentages, etc. $[\text{Jim}]_{\text{Person}}$ bought 300 shares of $[\text{Acme Corp.}]_{\text{Organization}}$ in $[2006]_{\text{Time}}$.

Entity linking is different from named-entity recognition (NER) in that NER identifies the occurrence of a named entity in text but it does not identify which specific entity it is. In entity linking, each named entity is linked to a unique identifier, e.g. SNOMED CT ID.

Mining frequently equals entity linking or classification

Peculiarities in Patient Records

Complete sentences are rare: a sentence may contain only a number of adverbs such as "fever, sweating, trouble breathing"

Auxiliary verbs such as be, is and are are rarely used.

Patient is mentioned in abbreviated form or not at all

The subject of sentences is often omitted

According to Pakhomov et al. (2005) there are 30% non-word tokens, abbreviations, acronyms, misspellings, wrongly used grammar etc. in clinical text, which is a good indication of the noisiness of clinical text.

Excercise

You are analyzing two short medical reports that describe symptoms of different patients:

- **Report 1**: "Patient exhibits fever, cough, and shortness of breath."
- **Report 2**: "Patient has fever, sore throat, and fatigue."

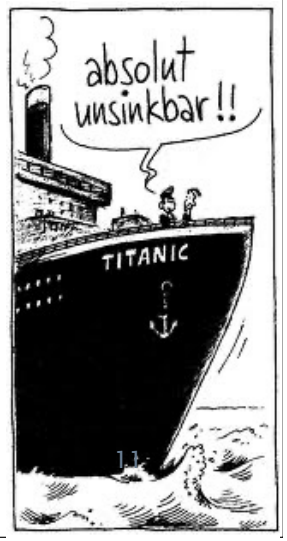
Symptom	Frequency in Report 1	Frequency in Report 2
fever	1	1
cough	1	0
shortness of breath	1	0
sore throat	0	1
fatigue	0	1

1. Calculate the cosine similarity between the symptoms of the reports.
2. Is cosine similarity the same as the Euclidian distance when the vectors are normalized?
3. Would the results profit from using SNOMED CT?

Homework assignment

(All, 10 minutes)

Solve the exercise on slide 10.



Cosine similarity and Euclidian distance

1. Start with the Euclidean distance formula for normalized vectors:

$$d_{\text{Euc}}(\mathbf{u}, \mathbf{v}) = \|\mathbf{u} - \mathbf{v}\|$$

   Squaring both sides, we get:

$$d^2_{\text{Euc}}(\mathbf{u}, \mathbf{v}) = \|\mathbf{u} - \mathbf{v}\|^2$$

2. Expand $\|\mathbf{u} - \mathbf{v}\|^2$ using the identity for the norm of a difference:

$$\|\mathbf{u} - \mathbf{v}\|^2 = (\mathbf{u} - \mathbf{v}) \cdot (\mathbf{u} - \mathbf{v}) = \mathbf{u} \cdot \mathbf{u} - 2\mathbf{u} \cdot \mathbf{v} + \mathbf{v} \cdot \mathbf{v}$$

   Since $\|\mathbf{u}\| = 1$ and $\|\mathbf{v}\| = 1$, we know that $\mathbf{u} \cdot \mathbf{u} = 1$ and $\mathbf{v} \cdot \mathbf{v} = 1$, so this simplifies to:

$$d^2_{\text{Euc}}(\mathbf{u}, \mathbf{v}) = 1 - 2\mathbf{u} \cdot \mathbf{v} + 1 = 2 - 2\mathbf{u} \cdot \mathbf{v}$$

3. Now, recall that for normalized vectors, $\mathbf{u} \cdot \mathbf{v} = \cos(\theta)$, where θ is the angle between the two vectors. Substituting this into the equation, we get:

$$d^2_{\text{Euc}}(\mathbf{u}, \mathbf{v}) = 2 - 2\cos(\theta)$$

4. Taking the square root of both sides, the Euclidean distance between normalized vectors becomes:

$$d_{\text{Euc}}(\mathbf{u}, \mathbf{v}) = \sqrt{2 - 2\cos(\theta)}$$

Using SNOMED CT

SNOMED CT contains over 300,000 concepts related to clinical terms, with many different synonyms for the same concept. For example, a disease like **myocardial infarction** might be referred to as: Heart attack, MI, Myocardial infarction

SNOMED CT links these terms to a single concept identifier, which allows the text mining system to recognize different phrases or spellings as referring to the same medical condition. For example, in clinical text:

"The patient suffered a heart attack."

"History of MI was noted."

Both sentences would be linked to the same SNOMED CT concept for myocardial infarction (SNOMED CT Concept ID: 22298006), allowing for more accurate data extraction.

Text Mining: Supervised Sentiment Analysis

What is sentiment analysis?

Building models to evaluate the sentiment of some text. These sentiments can be binary (positive and negative) or much more differentiated (see below)

We also measure how sentiments vary in different directions (**polarity**).

If our training data is human generated, we cannot exceed human-level performance

Types of sentiments: emotions, intent detection, tone (e.g., sarcasm and irony), subjectvity versus objectivity (an opinion or rather a statement of a fact)

Applications: product satisfcation, social media sentiment, customer interaction via chatbots.
And what are applications in the biomedical domain?

Sentiment Analysis Process

https://medium.com/@tomyuz/a-sentiment-analysis-approach-to-predicting-stock-returns-d5ca8b75a42

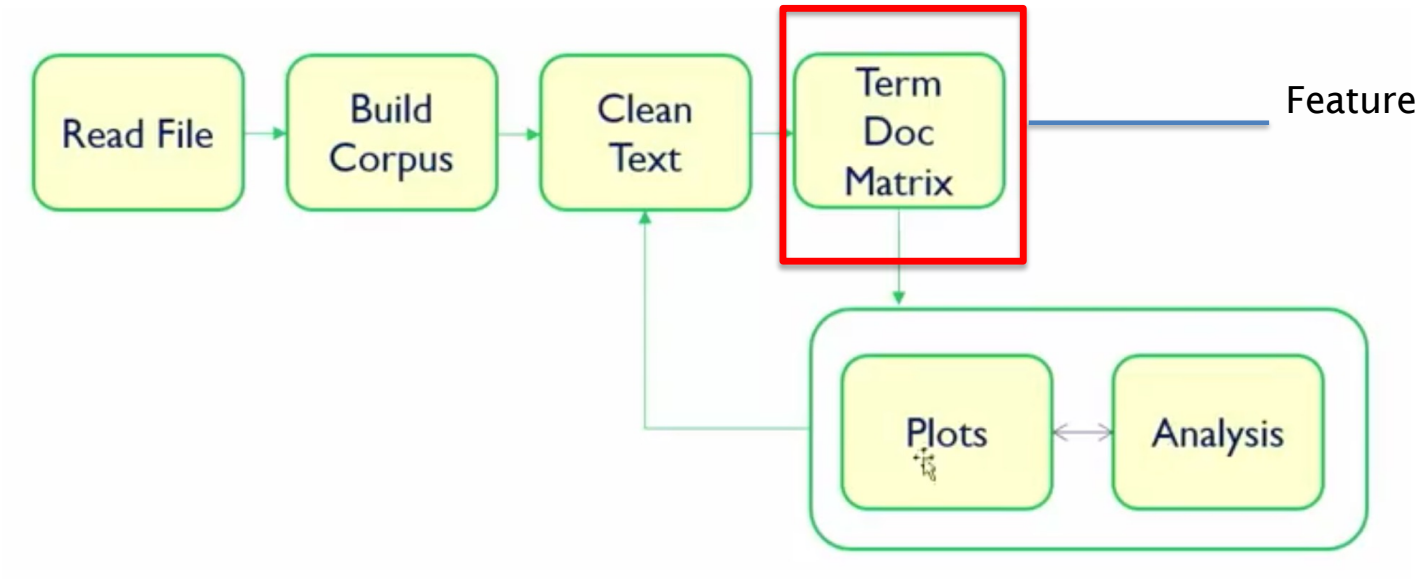
Training sentiment models

1. The document dataset is split into train and test datasets (Why at the start?)
2. Preprocesss the data by stripping unwanted characters, removing stop words, and performing other common preprocessing steps.
3. The text is converted to a numeric vector representation in order to extract the features.
4. Train the model with a machine learning alogirithm
5. Make predictions on the test data

Differences to the use of sentiment lexicons

1. Tokenization is necessary, but is frequently part of a vectorizer.
2. Conversion to a numeric vector representation is necessary.
3. Training data with labels are required.

General process for text mining



Feature extraction for text mining

From raw text → numerical representation. ML models require vectors, not raw words

None: when using sentiment lexicons (only data cleansing is required)

Bag of Words Model: Represent each document as frequency of words, disregarding grammar and even word order but keeping multiplicity.

$$f(t,d) = \text{count of term } t \text{ in document } d \qquad \mathbf{v}_d = (f(t_1,d), f(t_2,d), \ldots, f(t_n,d))$$

Document-term matrix: Combine BoW for multiple docs $M_{ij} = f(t_j, d_i)$

TF-IDF: balances frequency with rarity

Document-term matrix

	Term-Document Matrix				
Document/ Terms	**Investment**	**Profit**	**Happy**	**Success**	**…**
Doc 1	10	4	3	4	
Doc 2	7	2	2		
Doc 3			2	6	
Doc 4	1	5	3		
Doc 5		6		2	
Doc 6	4		2		
…					

TF-IDF

$$\mathrm{TF}(t,d) = \frac{f(t,d)}{\sum_k f(t_k,d)}$$

$$\mathrm{IDF}(t,D) = \log \frac{|D|}{1 + |\{d \in D : t \in d\}|}$$

$$\text{TF-IDF}(t,d,D) = \mathrm{TF}(t,d) \cdot \mathrm{IDF}(t,D)$$

Applications: Search engines (ranking), Text classification, Keyword extraction

Example:

- pain" appears in all documents → IDF ≈ 0 → low weight
- "morphine" appears once → high IDF → high weight

Naive Bayes from Scikit-Learn

`MultinomialNB` implements the naive Bayes algorithm for multinomially distributed data, and is one of the two classic naive Bayes variants used in text classification (where the data are typically represented as word vector counts, although tf-idf vectors are also known to work well in practice). The distribution is parametrized by vectors $\theta_y = (\theta_{y1}, \ldots, \theta_{yn})$ for each class y, where n is the number of features (in text classification, the size of the vocabulary) and θ_{yi} is the probability $P(x_i \mid y)$ of feature i appearing in a sample belonging to class y.

The parameters θ_y is estimated by a smoothed version of maximum likelihood, i.e. relative frequency counting:

$$\hat{\theta}_{yi} = \frac{N_{yi} + \alpha}{N_y + \alpha n}$$

where $N_{yi} = \sum_{x \in T} x_i$ is the number of times feature i appears in a sample of class y in the training set T, and $N_y = \sum_{i=1}^{n} N_{yi}$ is the total count of all features for class y.

The smoothing priors $\alpha \geq 0$ accounts for features not present in the learning samples and prevents zero probabilities in further computations. Setting $\alpha = 1$ is called Laplace smoothing, while $\alpha < 1$ is called Lidstone smoothing.

https://scikit-learn.org/stable/modules/naive_bayes.html#multinomial-naive-bayes

Naive Bayes from Scikit-Learn

```
import numpy as np
rng = np.random.RandomState(1)
X = rng.randint(5, size=(6, 100))
y = np.array([1, 2, 3, 4, 5, 6])
from sklearn.naive_bayes import MultinomialNB
clf = MultinomialNB()
clf.fit(X, y)
print(clf.predict(X[2:3]))
```

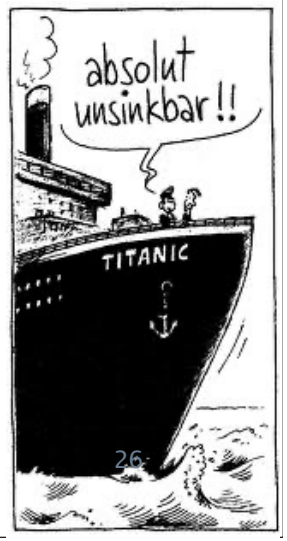
Evaluation metrics

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.25)

classifier.fit(X_train, y_train)
preds = classifier.predict(X_test)
from sklearn.metrics import accuracy_score, precision_score, classification_report,
confusion_matrix

print("Accuracy:", accuracy_score(y_test, preds))
print("Precision:", precision_score(y_test, preds))
print(classification_report(y_test, preds))
print(confusion_matrix(y_test, preds))
```

Homework assignment

(All, 10 minutes)

Make the code in Sentiment-NB.jpynb in the materials of week 3 run on your machine. The three data text files are in the folder data of week 3. Why is padding the text not useful here?



Text Mining with Word Embeddings

Problems of bag-of-words

It is static and is not really looking at the context, e.g. buy, sell, and pay offer differing perspectives on the same underlying purchasing event (word relatedness)

Semantic Similiraty is missing: cat is similar to dog and cold is the opposite of hot

Connotations are missing: happy as positive and sad as negative

Words can have multiple meanings, e.g. mouse is a pet or a tool (word sense disambiguation)

⇒ In general, sequentiality and relations used in ontologies are missing: hypernymy (IS-A), antonymy (opposites) and meronymy (part-whole relations)

Word Embeddings

Unlike the vectors we've seen so far, embeddings are short, with number of dimensions d ranging from 50-1000. These d dimensions don't have a clear interpretation. And the vectors are dense: instead of vector entries being sparse, mostly-zero counts or functions of counts, **the values will be real-valued numbers that can be negative**.

It turns out that dense vectors work better in every NLP task than sparse vectors. While we don't completely understand all the reasons for this, we have some intuitions. Representing words as 300-dimensional dense vectors requires our classifiers to learn far fewer weights than if we represented words as 50,000-dimensional vectors, and **the smaller parameter space possibly helps with generalization and avoiding overfitting**. Dense vectors may also do a better job of capturing synonymy. For example, in a sparse vector representation, dimensions for synonyms like car and automobile dimension are distinct and unrelated.

Embedding Example

King	-	man	+	woman	=	Queen
1	-	0.2	+	0.2	=	1
0	-	0	+	0	=	0
0	-	0	+	0	=	0
1	-	0.3	+	0.2	=	0.9
-1	-	1	+	1	=	-1

Embedding Example

```
import spacy
import numpy as np

nlp = spacy.load("en_core_web_md")  # or "en_core_web_lg"

def v(w): return nlp(w).vector
def cos(a,b): return float(np.dot(a,b)/(np.linalg.norm(a)*np.linalg.norm(b)))

king, man, woman, queen = map(v, ["king","man","woman","queen"])
analogy = king - man + woman

print("cos(king, queen)     =", round(cos(king, queen), 3))
print("cos(analogy, queen)  =", round(cos(analogy, queen), 3))
```

Embedding Example

```
# nearest neighbors to the analogy (excluding the query words)
vocab = [w for w in nlp.vocab if w.has_vector and w.is_alpha and w.is_lower]
sims = []
for w in vocab:
    if w.text in {"king","man","woman"}:
        continue
    sims.append((w.text, cos(analogy, w.vector)))
print(sorted(sims, key=lambda x: x[1], reverse=True)[:10])
```

cos(king, queen) = 0.611 cos(analogy, queen) = 0.618 [('queen', 0.6178014278411865), ('and', 0.38990047574043274), ('that', 0.38483577966690063), ('havin', 0.3667822182178497 3), ('where', 0.33859238028526306), ('she', 0.3244561851024627 7), ('they', 0.3206636309623718), ('somethin', 0.3095314800739288 3), ('there', 0.305420696735382 1), ('should', 0.298373043537139 9)]

Keras Word Embeddings

Here, embeddings are produced on the fly with the (supervised learning) task, and can be initialized by word2vec or similar other embeddings

Each input integer is used as the index to access a table that contains all possible vectors. That is the reason why you need to specify the size of the vocabulary as the first argument.

Hope to see you soon: [0, 1, 2, 3, 4]

Nice to see you again: [5, 1, 2, 3, 6]

Embedding(7, 2, input_length=5)

7 is the number of distinct words in the training set, i.e., Vocab size

2 indicates the size of the embedding vectors

5 size of each input sequence

Undertstanding keras-Embeddings (Lookup Table)

index	Embedding
0	[1.2, 3.1]
1	[0.1, 4.2]
2	[1.0, 3.1]
3	[0.3, 2.1]
4	[2.2, 1.4]
5	[0.7, 1.7]
6	[4.1, 2.0]

```
[[0.7, 1.7], [0.1, 4.2], [1.0, 3.1], [0.3, 2.1], [4.1, 2.0]]
```

Problem of using Word Embeddings in regression models

Each feature is usually a single scalar value (e.g., height, weight, or age), and the model assumes a one-to-one relationship between each feature and the target variable.

Solutions:

1. Averaging Embeddings across all words in a sentence
2. Pooling Techniques: Max-Pooling
3. Dimension reduction: t-SNE or PCA
4. Embedding Aggregation through attention mechanism
5. Use Advanced Models: neural networks or tree-based models

What would you prefer?

Embeddings in keras (1)

```
from numpy import array
from tensorflow.keras.preprocessing.text import one_hot
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten,Embedding,Dense
# Define 10 restaurant reviews
reviews =['Never coming back!', 'horrible service', 'rude waitress', 'cold food',
          'horrible food!', 'awesome', 'awesome services!', 'rocks', 'poor work',
          'couldn\'t have done better' ]

#Define labels
labels = array([1,1,1,1,1,0,0,0,0,0])

Vocab_size = 50
encoded_reviews = [one_hot(d,Vocab_size) for d in reviews]
print(f'encoded reviews: {encoded_reviews}')
```

Embeddings in keras (2)

```
max_length = 4
padded_reviews = pad_sequences(encoded_reviews,maxlen=max_length,padding='post')
print(padded_reviews)
```

```
[[18 39 17 0]
[27 27 0 0]
[ 5 19 0 0]
[41 29 0 0]
[27 29 0 0]
[ 2 0 0 0]
[ 2 1 0 0]
[49 0 0 0]
[26 9 0 0]
[ 6 9 11 21]]
```

Embeddings in keras (3)

```
model = Sequential()
embedding_layer = Embedding(input_dim=Vocab_size,output_dim=8,input_length=max_length)
model.add(embedding_layer)
model.add(Flatten())
model.add(Dense(1,activation='sigmoid'))
model.compile(optimizer='adam',loss='binary_crossentropy',metrics=['acc'])
print(model.summary())
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 4, 8)              400
_________________________________________________________________
flatten (Flatten)            (None, 32)                0
_________________________________________________________________
dense (Dense)                (None, 1)                 33
=================================================================
Total params: 433
Trainable params: 433
Non-trainable params: 0
_________________________________________________________________
None
```

Homework assignment

(All, 20 minutes)

Tweak the neural network in section "Neural Network" of Sentiment-NB.jpynb such that the accuracy is above 80%. Is GlobalAveragePooling1D useful here?