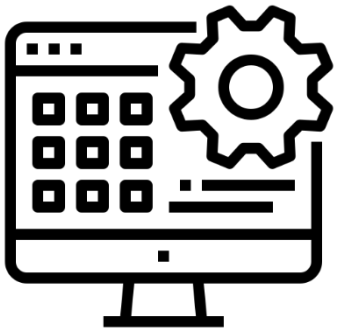
# Material Summary: Working with Text

## 1. Text Processing

### 1.1 Text Data

- Documents, **written in plain text**
  - News, tweets, blog posts, poems, books, legal documents, etc.
  - May also be auto-generated (i.e., server logs)
- Objective
  - Preprocess the **text data so that it's structured**
  - Algorithms can analyze a **table of numbers, not plain text**
  - This is especially **true for machine learning algorithms**
- Applications
  - **Sentiment Analysis**
  - **Grouping Texts** – similar topics, similar authors
  - **Classification** (e.g., spam / fake news prevention)
  - **Text Summarization**



### 1.2 Character Frequencies

- Reading is **simple**:
  - **Open the file**
  - **Read it**
  - **Close it**

```
text = ""
with open("alice.txt", "r", encoding = "utf-8") as f:
    text = f.read()
print(len(text))
```

- A string is a **collection of characters**
  - There are several ways to count them, the easiest being by using a library: **collections.Counter**

```
from collections import Counter
char_counter = Counter(text)
```

  - Most common characters: **"etaoin shrdlu"**

```
char_counter.most_common(20)
```

  - Similarly, most common words: **split by all non-word characters**

```
import re
word_counter = Counter(re.split("\W+", text))
```

### 1.3 Preparing Text Data

- Before we start working with the text, we have to "**normalize**" and **clean up** the messy data

- Remove all **non-letter characters**
  - Numbers, punctuation, whitespace, etc.
  - If needed, apply **additional rules**
  - For example: if we're looking at tweets, **@mention** means a username and we may want to get rid of it
- Transform **all characters to lowercase**
- Remove "**stopwords**"
- Words that are **too frequent** in all documents and **don't contain much information** such as: "the", "a", "is"
- Perform **stemming**
- Extract the stems of all words: "connected", "connection"
- "connecting" should all point to "connect"

### 1.4 Stopwords and Stemming: NLTK

- **NLTK** is a **library for working with natural language**
  - Contains **all frequently used algorithms and corpora**
  - Installation is as usual, using **conda**:

```
conda install nltk
```

- Getting and removing **stopwords**
  - Download the words first

```
import nltk
nltk.download("stopwords")
from nltk.corpus import stopwords
stop = set(stopwords.words("english"))
sentence = "this is a foo bar sentence"
print([w for w in sentence.lower().split() if w not in stop])
```

- Stemming
  - Porter's algorithm (includes many "manual" rules)

```
from nltk.stem.porter import *
stemmer = PorterStemmer()
words = ["caresses", "flies", "dies", "seizing", "itemization",
"sensational", "traditional", "reference", "plotted"]
print([stemmer.stem(word) for word in words])
```

### 1.5 TF – IDF

- **T**erm **f**requency – **i**nverse **d**ocument **f**requency
  - A common method **to preprocess the text**

$$w_{x,y} = tf_{x,y} \times \log\left(\frac{N}{df_x}\right)$$

**TF-IDF**

Term $x$ within document $y$

$tf_{x,y}$ = frequency of $x$ in $y$

$df_x$ = number of documents containing $x$

$N$ = total number of documents

  - **High score**: **rare, specific words**

        - **Hypothesis**: these may be better related to the topic
        - **Note**: This may also include misspelled words and / or names
    - **Low score**: **words that occur in nearly all documents**

### 1.6 Using TF – IDF

- Read the "**20 newsgroups**" dataset (from scikit-learn)

```
from sklearn.datasets import fetch_20newsgroups
# Download only some categories to speed up the process
newsgroups = fetch_20newsgroups()
```

- Initialize the algorithm (docs) and compute the matrix

```
tfidf = TfidfVectorizer(input = "content", analyzer = "word",
    ngram_range = (1, 4), min_df = 0, stop_words = stop,
sublinear_tf = True)
tfidf_matrix = tfidf.fit_transform(newsgroups.data)
```

- Get all **feature names**

```
feature_names = tfidf.get_feature_names()
```

- Get the IDF for each word / n-gram in one document

```
doc = 0 # Change the index to view another document
feature_index = tfidf_matrix[doc, :].nonzero()[1]
tfidf_scores = zip(feature_index, [tfidf_matrix[doc, x] for x in
feature_index])
for w, s in [(feature_names[i], s) for (i, s) in tfidf_scores]:
    print(w, s)
```