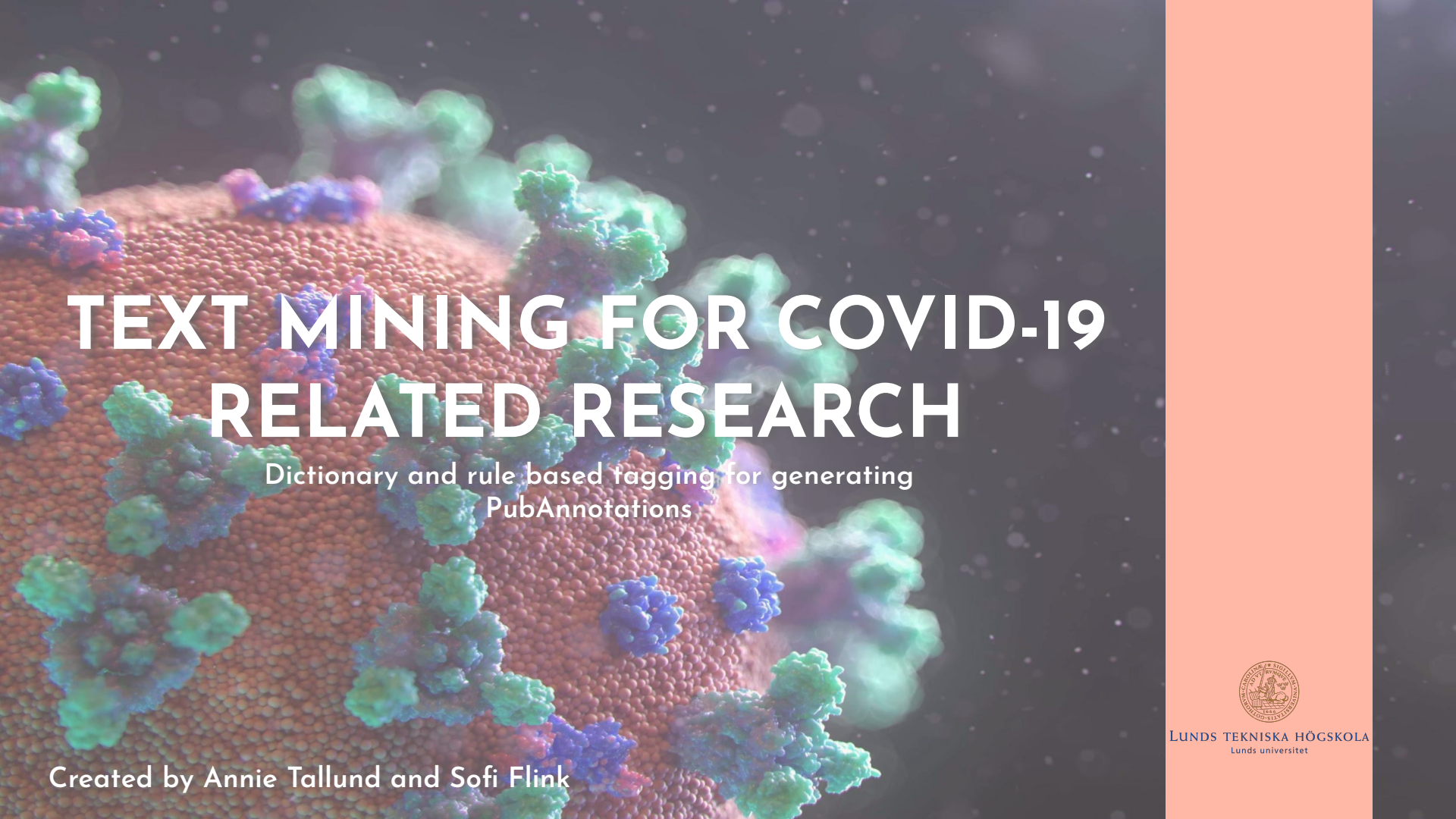
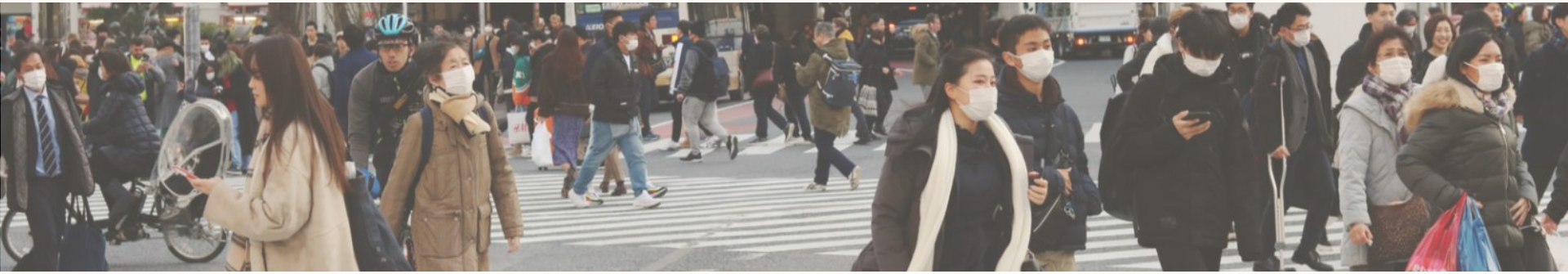


TEXT MINING FOR COVID-19 RELATED RESEARCH

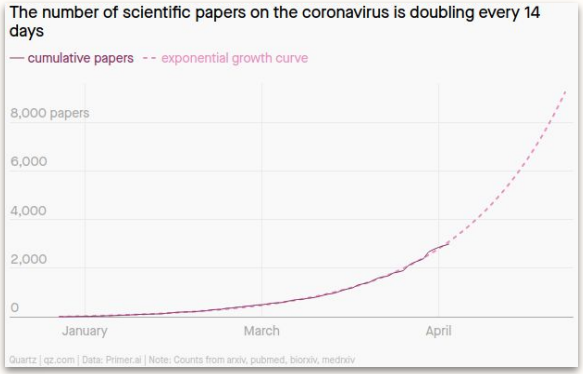
Dictionary and rule based tagging for generating PubAnnotations

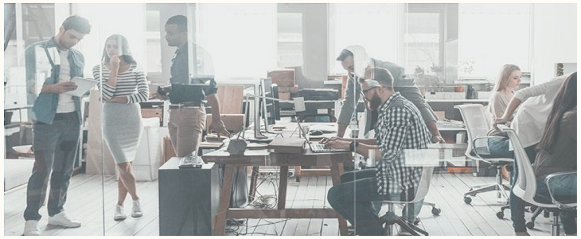
Created by Annie Tallund and Sofi Flink

LUNDS TEKNISKA HÖGSKOLA
Lunds universitet



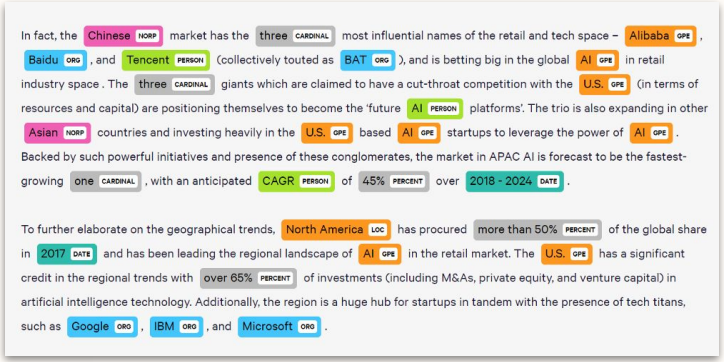
PROBLEM INTRODUCTION

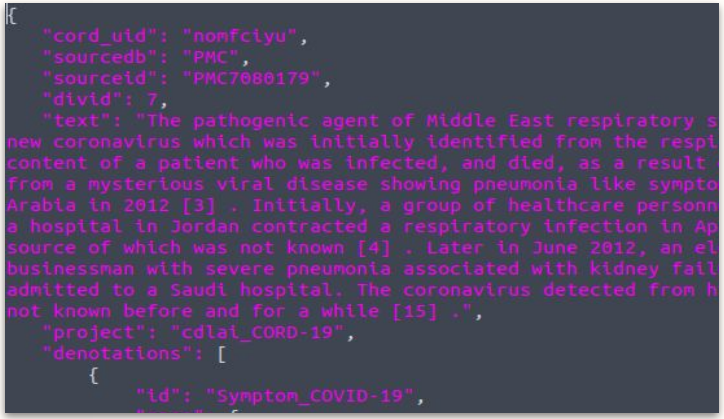


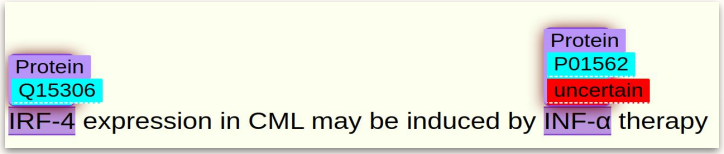


- Unmanageable amount of research articles
- COVID-19 Open Research Dataset Challenge (CORD-19 Open Research Dataset) on KAGGLE
- 59,000 scholarly articles

PROJECT DESCRIPTION







NAMED ENTITY RECOGNITION (NER)

- Locate and classify
- Dictionaries, rules, models
- Generate PubAnnotations to share result with others

IMPLEMENTATION

INPUT DATA

- CORD-19 data set
- JSON- and CSV-files
- Dictionaries with words to tag
  - 'Disease_COVID-19'
  - 'Symptom_COVID-19'
  - 'Virus_SARS-CoV-2'
- Folder names used as argument
  - '.../data/directories/'

```
def __load_vocabulary(self, file_path, word_class):
    """
    Opens a vocabulary/dictionary containing the wo
    is added to a dictionary using the word class c
    """
    vocab_list = [row.strip() for row in
                  open(file_path)]
    self.vocabs_col_dict.update({word_class:
                                 vocab_list})
    self.word_classes.add(word_class)

def __load_patterns(self):
    """
    Save patterns in same fashion as vocabularies/d
    Pattern 1. 'chemical_antiviral' tags all words
    """
    self.patterns_dict = {'chemical_antiviral':
                          r'(?i)\b\S*vir\b'
                          }
    for word_class in self.patterns_dict:
        self.word_classes.add(word_class)
```

DATA STRUCTURES

- Python dictionaries
- Metadata represented in a list, map index and 'paper_ids'/'sha'

TAGGING WITH DICTIONARIES AND RULES

- One paragraph a time
- Find spans with regex
- Different priorities

```
def __tag_paragraph(self, paragraph):
    """
    For a paragraph, iterate through all vocabularies and patterns and tag using corresponding regex-pattern.
    """
    self.paragraph_matches.clear()
    case_insensitive_regex = r'(?i)'
    hyphen_or_whitespace_or_both_regex = r'(\s|\-\s?)'
    opt_plural_regex = r'(es|s)?'
    boundary_regex = r'\b'
    for vocabulary in self.vocabs_col_dict:
        for word in self.vocabs_col_dict[vocabulary]:
            pattern = case_insensitive_regex + boundary_regex
            if True in [character in word for character in string.whitespace]:
                composite_words = word.split()
                for composite_word in composite_words:
                    if composite_word == composite_words[-1]:
                        pattern += composite_word + opt_plural_regex + boundary_regex
                    else:
                        pattern += composite_word + hyphen_or_whitespace_or_both_regex
            else:
                pattern += word + opt_plural_regex
            self.__tag_pattern(pattern, paragraph, vocabulary)

    for word_class in self.patterns_dict:
        self.__tag_pattern(self.patterns_dict[word_class], paragraph, word_class)
```

```
def is_longest_match(self, new_match, prev_match):
    """
    Returns true if new match is equally long or longer than prev_match for a common span and if words don't match.
    """
    new_word_match = new_match.group(0)
    prev_word_match = prev_match.group(0)
    if prev_word_match == new_word_match:
        return True
    new_pattern = new_match.re
    prev_pattern = prev_match.re
    is_span_overlap = ((prev_match.start() <= new_match.end() and prev_match.end() >= new_match.start()) or
                       (prev_match.start() >= new_match.end() and prev_match.end() <= new_match.start()))
    shortest_word = min(new_word_match, prev_word_match, key=len)
    if shortest_word == new_word_match:
        prev_tagged = re.search(new_pattern, prev_word_match)
        if prev_tagged and is_span_overlap:
            return False
    else:
        prev_tagged = re.search(prev_pattern, new_word_match)
        if prev_tagged and is_span_overlap:
            del self.paragraph_matches[prev_match]
            return True
    return True
```

- Find longest span
- Update dictionary
- Not time efficient

LUNDS TEKNISKA HÖGSKOLA
Lunds universitet

EVALUATION

```
CLASS: Virus_SARS-CoV-2
Total: 46
True Positives: 33
False Positives: 8
False Negatives: 13
CLASS: Disease_COVID-19
Total: 13
True Positives: 5
False Positives: 1
False Negatives: 8
CLASS: Symptom_COVID-19
Total: 28
True Positives: 17
False Positives: 8
False Negatives: 11
```

- Test set:
  - 10 articles from PubMed
  - Gold standard corpus
- Precision and Recall
- Missing words in dictionaries

EVALUATION RESULT

	Disease_ COVID-19	Symptom_ Covid-19	Virus_SARS -CoV-2	Micro	Macro
Precision Score	0.8	0.68	0.79	0.75	0.76
Recall Score	0.31	0.61	0.72	0.62	0.54

Total entities	87
True positives	54
False positives	18
False negatives	33

Harmonic Mean	0.63

Lunds tekniska högskola
Lunds universitet

DIFFICULTIES

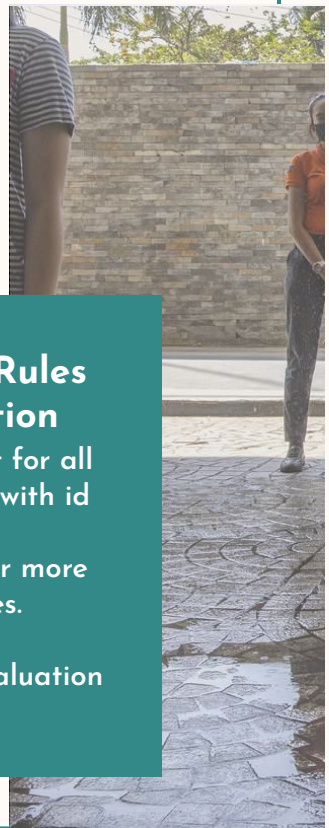
Experience

- No background in text mining
- First time encountering PubAnnotation format

Project

- Finding edge cases
- Different conceptions on problem description between project participants
- A lot of manual work for error checking

FUTURE ITERATIONS

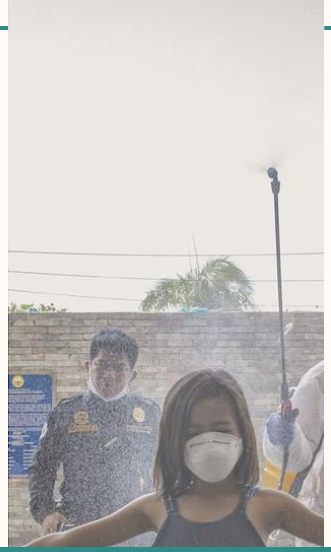


Dictionary-Rules Optimization

Use ordered list for all words mapped with id

Add support for more dictionaries.

Continue the evaluation



Run-time Optimization

More efficient algorithms

Reduce iterations

Exclude nonsense words (stop words)

User Interface

Simplify use for non-programmers

BEFORE WE ROUND OFF. . .

Credits
11

Special thanks to:

- Supervisors Sonja Aits and Pierre Nugues
  - Guidance
- Salma
  - Gold standard PubAnnotations
- Each and every participant of project
  - Support and knowledge

LUNDS TEKNISKA HÖGSKOLA
Lunds universitet

QUESTION TIME

Annie Tallund
Lunds Tekniska Högskola
D4
an0284ta-s@student.lu.se

Sofi Flink
Lunds Tekniska Högskola
C4
bmp13sfl@student.lu.se

Thank you for listening

Does anyone have any questions?