Echahid Hamma Lakhdar University - El Oued

Faculty of Exact Sciences

Department of Computer Science *2nd master's degree*

Duration: 1 hour 30 minutes *Date 22/01/2026*

Natural Language Processing (NLP) Exam

Questions

1. Name three types of corpora and give an example of each.
   1. General English Corpora
   Brown Corpus: First million-word electronic corpus
   British National Corpus (BNC): 100 million words of British English
   Corpus of Contemporary American English (COCA): 1 billion words
   2. Web-Scale Corpora
   Common Crawl: Hundreds of billions of web pages
   Google Books Ngrams: Millions of books scanned
   Wikipedia Dump: Entire Wikipedia content
   3. Specialized Corpora
   PubMed: Biomedical literature
   Legal Case Law: Court decisions and statutes
   Twitter/Reddit Data: Social media text
2. What are the main goals of text preprocessing? List at least three common preprocessing steps.
   Text data in its raw form often contains inconsistencies, errors, and noise that can negatively impact model training. Cleaning text data is essential because NLP algorithms rely on understanding word usage and semantic relationships within the text.
   Three common preprocessing steps :
   - Data Cleaning: Removing special characters, numbers, and extra spaces.
   - Lowercasing: Converting text to lowercase to avoid duplicates due to case differences.
   - Tokenization: Splitting text into smaller units called tokens (words, sentences).
3. How does lemmatization help with vocabulary size reduction?
   Normalize word forms Groups different forms of the same word together, reducing vocabulary size and improving word frequency analysis.
4. What is *Named Entity Recognition (NER)*? What types of entities can spaCy identify?
   identifies real-world objects (i.e., anything that can be denoted with a proper name) and classifies them into predefined categories.
   spaCy can identify the following:
   - PERSON: people (existing or fictional).
   - LOC: locations.

- ORG: organizations such as companies, agencies, institutions, organizations, etc.
- GPE: countries, cities, states.

5. What are word embeddings? Name two popular word embedding models
   Word embeddings are dense vector representations of words, capturing semantic relationships.
   Name two popular word embedding models
   - Word2Vec: A model that learns vector representations of words based on their context.
   - GloVe: A model based on the global co-occurrence of words in a corpus.

Exercice 1

Given the following short movie reviews, each labeled with a genre, either comedy or action:

Class	Movie reviews
Comedy	fun, friends, laugh
Action	fast, furious, shoot
Comedy	friends, fly, fast, fun
Action	furious, shoot, fun
Action	fly, fast, shoot, laugh

And a new movie review D: fast, family, shoot, fly
Compute the most likely class for D. Assume a Naive Bayes classifier.

Solution

New review **D:** fast, family, shoot, fly

Vocabulary = {fun, friends, laugh, fast, furious, shoot, fly, family} → 8 unique words.

Total documents = 5
Comedy = 2 documents
Action = 3 documents

P(Comedy) = 2/5 = 0.4
P(Action) = 3/5 = 0.6
Comedy: = 7 words total
Action: = 10 words total

Comedy probabilities:
P(fast| C)=(1+1)/15=2/15
P(family| C)=(0+1)/15=1/15
P(shoot| C)=(0+1)/15=1/15
P(fly| C)=(1+1)/15=2/15

$P(D|\ Comedy) = (2/15) \times (1/15) \times (1/15) \times (2/15) = 4/50625 \approx 7.90 \times 10^{-5}$

Action probabilities:
$P(fast|\ A) = (2+1)/18 = 3/18 = 1/6$
$P(family|\ A) = (0+1)/18 = 1/18$
$P(shoot|\ A) = (3+1)/18 = 4/18 = 2/9$
$P(fly|\ A) = (1+1)/18 = 2/18 = 1/9$

$P(D|\ Action) = (1/6) \times (1/18) \times (2/9) \times (1/9) = 2/8748 = 1//4374 \approx 2.29 \times 10^{-4}$

$P(Comedy|\ D) = 0.4 \times 7.90 \times 10^{-5} = 3.16 \times 10{-}5$

$P(Action|\ D) = 0.6 \times 2.29 \times 10^{-4} = 1.374 \times 10{-}4$

$P(Action|\ D) > P(Comedy|\ D) 1.374 \times 10{-}4 > 3.16 \times 10{-}5$

The most likely class is: Action

Exercice 2

Solution

1. In a Recurrent Neural Network (RNN) used for language modeling explain why hidden state $h^{(t)}$ depends on both: $x^{(t)}$ (the current input) and $h^{(t-1)}$ (the previous hidden state)

(RNN) remember previous information using hidden states and connect it to the current task.
x(t) represents **what's happening now** in the sequence
h(t-1) serves as a **compressed memory** of all previous inputs {x(1), x(2), ..., x(t-1)}

2. Why can't bidirectional RNNs be used for language modeling?
   They are not applicable to Language Modeling, because in Language Modeling only left context is available.

Exercice 3

Implement a function called RuleBasedSentimentAnalyzer() that analyzes the sentiment of a given text using hand-crafted rules, and lexicons, that handle negation and determine sentiment label : "POSITIVE" , "NEGATIVE" or "NEUTRAL".

Solution

```
import re
from collections import defaultdict


class SentimentLexicon:
    def __init__(self):
        self.positive_words = {
```

```
            'good', 'great', 'excellent', 'amazing', 'wonderful', 'fantastic',
            'awesome', 'brilliant', 'love', 'like', 'nice', 'perfect', 'best',
            'beautiful', 'outstanding', 'superb', 'terrific', 'fabulous',
            'marvelous', 'exceptional', 'pleasant', 'delightful', 'satisfied',
            'happy', 'pleased', 'content', 'joyful', 'ecstatic', 'thrilled'
        }

        self.negative_words = {
            'bad', 'terrible', 'awful', 'horrible', 'worst', 'hate', 'dislike',
            'ugly', 'poor', 'disappointing', 'unpleasant', 'disgusting',
            'annoying', 'frustrating', 'angry', 'mad', 'upset', 'sad',
            'depressing', 'miserable', 'horrific', 'dreadful', 'abysmal',
            'atrocious', 'appalling', 'lousy', 'rubbish', 'garbage', 'trash'
        }

        self.negation_words = {
            'not', "n't", 'no', 'never', 'nothing', 'none', 'nobody',
            'nowhere', 'neither', 'nor', 'cannot', 'without'
        }

        self.intensifiers = {
            'very': 1.5, 'extremely': 2.0, 'really': 1.3, 'quite': 1.2,
            'absolutely': 2.0, 'completely': 1.8, 'totally': 1.7,
            'utterly': 2.0, 'highly': 1.5, 'super': 1.4, 'incredibly': 1.8
        }

        self.diminishers = {
            'slightly': 0.7, 'somewhat': 0.8, 'barely': 0.6,
            'hardly': 0.5, 'almost': 0.9, 'partially': 0.8
        }


def RuleBasedSentimentAnalyzer(text, lexicon):

    # Step 1: Clean and tokenize the text
    def clean_and_tokenize(text):
```

```
    # Convert to lowercase
    text = text.lower()
    # Remove punctuation except apostrophes for contractions
    text = re.sub(r'[^\w\s\']', ' ', text)
    # Tokenize by splitting on whitespace
    tokens = text.split()
    return tokens

tokens = clean_and_tokenize(text)

# Initialize variables
score = 0.0
components = []
sentiment_words_count = 0

# Process tokens
for i, token in enumerate(tokens):
    current_modifier = 1.0
    current_word_type = None
    current_word = token

    # Check for intensifiers/diminishers in previous positions (window of 2)
    for j in range(max(0, i-2), i):
        if tokens[j] in lexicon.intensifiers:
            current_modifier *= lexicon.intensifiers[tokens[j]]
            components.append(('intensifier', tokens[j], lexicon.intensifiers[tokens[j]]))
        elif tokens[j] in lexicon.diminishers:
            current_modifier *= lexicon.diminishers[tokens[j]]
            components.append(('diminisher', tokens[j], lexicon.diminishers[tokens[j]]))

    # Check for negation in previous positions (window of 3)
    negated = False
    for j in range(max(0, i-3), i):
        if tokens[j] in lexicon.negation_words:
            negated = True
            components.append(('negation', tokens[j], -1.0))
```

```
            break

    # Check if current word is a sentiment word
    if token in lexicon.positive_words:
        current_word_type = 'positive'
        base_score = 1.0
        sentiment_words_count += 1
    elif token in lexicon.negative_words:
        current_word_type = 'negative'
        base_score = -1.0
        sentiment_words_count += 1

    # Apply negation if needed
    if current_word_type:
        if negated:
            base_score *= -1.0  # Flip sentiment if negated
            current_modifier *= 0.7  # Reduce strength when negated

        # Apply modifier
        word_score = base_score * current_modifier
        score += word_score

        # Record component
        if negated:
            components.append((f'negated_{current_word_type}', token, word_score))
        else:
            components.append((current_word_type, token, word_score))

# Calculate confidence
confidence = min(1.0, abs(score) / (sentiment_words_count + 1e-6))

# Determine sentiment label
if score > 0.5:
    sentiment = "POSITIVE"
elif score < -0.5:
    sentiment = "NEGATIVE"
```

```
else:
    sentiment = "NEUTRAL"

return {
    'sentiment': sentiment,
    'score': round(score, 2),
    'confidence': round(confidence, 2),
    'components': components
}
```