Natural Language Processing

Arabic Language as a Use Case

Yasser Hifny

Outline

Introduction to Arabic NLP

Why Arabic NLP is Unique

- Spoken by over **400 million people** across 22 countries.
- Rich historical literary tradition combined with vibrant social media presence.
- Poses distinct computational challenges compared to Indo-European languages.

Introduction to Arabic NLP

Why Arabic NLP is Unique

- Spoken by over **400 million people** across 22 countries.
- Rich historical literary tradition combined with vibrant social media presence.
- Poses distinct computational challenges compared to Indo-European languages.

Core Challenges

- **Diglossia:** Coexistence of Modern Standard Arabic (MSA) and Dialects.
- **Morphological Richness:** High degree of inflection and derivation.
- **Orthographic Complexity:** Omission of diacritics, script variations.

Diglossia & Language Varieties

Modern Standard Arabic (MSA)

- Formal language of news, literature, education, and official affairs.
- Standardized grammar and vocabulary.
- High availability of annotated datasets.

Dialectal Arabic (DA)

- Spoken vernaculars used in daily communication and social media.
- Lack of standardized spelling rules.
- Significant regional variations (Levantine, Egyptian, Gulf, Maghrebi).

Classical Arabic (CA)

Historical register found in religious texts and classical literature; rigid syntax and vocabulary.

Arabic Script & Orthography

Right-to-Left (RTL) Rendering

- Script flow requires specialized Unicode handling (BiDi engine).
- Contextual letter shaping: Isolated, Initial, Medial, and Final forms.

Orthographic Variations & Inconsistencies

- **Alef Variants:** Difference between أ, إ, آ, and ا.
- **Yaa vs. Alef Maqsura:** Distinguishing ي and ى.
- **Taa Marbouta vs. Haa:** Distinguishing ة and ه.
- **Arabizi:** Chat Arabic written with Latin characters and numerical substitutions (e.g., 3 for *Ayin*, 7 for *Haa*).

Diacritization (Tashkeel) Challenges

Vocalization in Arabic

Arabic text is usually written **undiacritized**. Short vowels are represented by diacritical marks placed above or below consonants.

Diacritic	Name	Phonetic Value
◌َ	*Fatha*	Short /a/
◌ُ	*Damma*	Short /u/
◌ِ	*Kasra*	Short /i/
◌ْ	*Sukun*	No vowel (vowel-less)
◌ّ	*Shadda*	Gemination (consonant doubling)

Semantic Ambiguity

Word عَلَم (*'alama* - taught), عِلْم (*'ilm* - science), عَلَم (*'alam* - flag).

Arabic Morphology - Derivational System

Root-and-Pattern Architecture

Arabic morphology is primarily **non-concatenative**. Words are generated by interweaving a root into a morphological pattern (*Wazn*).

Example: Root K-T-B (ك-ت-ب) - Writing

- **Pattern *Fa'ala*:** كَتَبَ (*Kataba* - He wrote)
- **Pattern *Faa'il*:** كَاتِب (*Kaatib* - Writer)
- **Pattern *Maf'ool*:** مَكْتُوب (*Maktoob* - Letter/Written)
- **Pattern *Maf'al*:** مَكْتَب (*Maktab* - Desk/Office)
- **Pattern *Mafta'a*:** مَكْتَبَة (*Maktabah* - Library)

Arabic Morphology - Inflection & Agglutination

Clitics and Agglutination

A single Arabic surface word form can represent a complete sentence with multiple clitics attached.

Structure of an Agglutinated Arabic Word

$$\text{Word} = \text{Prefixes} + \text{Stem} + \text{Suffixes}$$

Deconstruction Example: وسيكتبونها (*wa-sa-yaktuboonaha*)

- **Conjunction (*wa*):** و (And)
- **Future Marker (*sa*):** س (Will)
- **Imperfect Prefix (*ya*):** ي (He/They)
- **Root Stem (*ktb*):** كتب (Write)
- **Plural Suffix (*oon*):** ون (They - masculine)
- **Object Pronoun (*ha*):** ها (It / Her)
- **Translation:** "And they will write it"

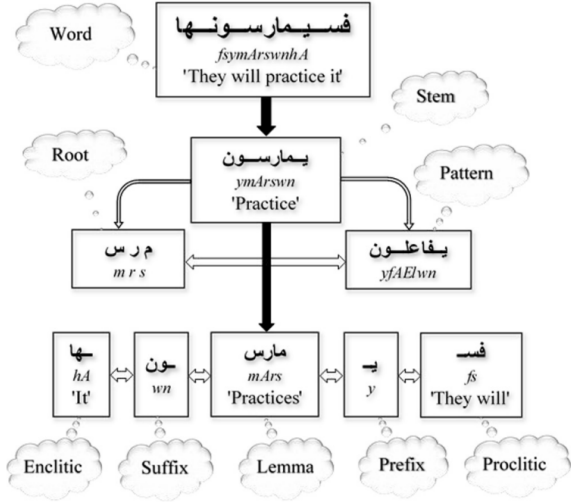

Figure: Word Representations in Alkhalil.

Text Preprocessing Pipeline

Step-by-Step Arabic Normalization

1. **Noise Removal:** Clean HTML tags, URLs, user handles, non-Arabic script.
2. **Diacritic Stripping:** Remove optional diacritics to unify vocabulary.
3. **Letter Normalization:**
   - Map {أ, إ, آ} → ا
   - Map ى → ي
   - Map ة → ه (Optional depending on downstream task)
4. **Kashida Removal:** Strip lengthening characters (ـ).
5. **Punctuation & Digit Handling:** Standardize Eastern/Western Arabic numerals.

Removing Arabic Diacritics Using RegEx

Arabic diacritics are Unicode combining marks.
A regular expression can identify the Arabic diacritic range and remove these characters.

Python Example

```
import re

def remove_diacritics(text):
    pattern = r'[\u0610-\u061A\u064B-\u065F\u0670\u06D6-\u06ED]'
    return re.sub(pattern, '', text)
```

Example

Example

Input:

الْجَدِيدُ الْكِتَابُ

Output:

الجديد الكتاب

Transliteration Schemes

Why Transliteration Matters

Enables lossless mapping of Arabic characters to ASCII representation for computation and display in environments lacking RTL support.

Buckwalter Transliteration

- Strictly 1-to-1 mapping.
- Uses ASCII symbols ($, *, >, <).
- Standard in legacy NLP tools.

Safe Buckwalter / Habash-Soudi-Buckwalter (HSB)

- Eliminates problematic punctuation characters.
- Uses clear alphanumeric representations.

Arabic Tokenization: Word-Level

Definition

Split text on whitespace and punctuation; each resulting unit is treated as one token.

المدرسة إلى وسيذهبون

[المدرسة, إلى, وسيذهبون]

Advantages

- Preserves whole-word meaning
- Simple, fast, easy to inspect

Drawbacks for Arabic

- وسيذهبون bundles conjunction + future + verb + plural into one token
- Massive vocabulary; high OOV rate

Arabic Tokenization: Character-Level

Definition

Split text into individual Unicode characters (optionally including diacritics).

كتّاب

becomes

[ك, ت, ا, ب]

Advantages

- Tiny, closed vocabulary (∼36 letters)
- Zero OOV tokens

Drawbacks

- Discards word-level meaning
- Long sequences → higher compute cost

Arabic Tokenization: Byte-Pair Encoding (BPE)

Definition

Start from characters and iteratively merge the most frequent adjacent pair into a new subword unit, until a target vocabulary size is reached.

- A middle ground between word-level and character-level tokenization
- Learns common Arabic affixes and clitics (e.g. ال, و, ب) as reusable subword units
- Keeps frequent whole words intact while still handling rare/unseen words gracefully
- Used by AraBERT, AraGPT2, and most Arabic transformer models

Why it matters for Arabic

Balances vocabulary size against Arabic's rich agglutinative morphology.

BPE Training Example: Learning Merge Rules

Toy Corpus (word : frequency)

الكتب (3), كتب (3), الكتاب (4), كتاب (5),

Each word starts fully split into characters, with _ marking word end.

Step	Most Frequent Pair	Merge
1	ت+ك (count 15)	كت → كت
2	كت+ا (count 9)	كا → كا
3	ب+كا (count 9)	كاب → كاب
4	ا+ل (count 7)	ال → ال

Result: كاب_ stays whole; كتب_ remains split as ب+كت, since its merge count never wins a round.

BPE Application: Tokenization

Applying Learned Merges to a New (Unseen) Word

Merge rules are applied in the order they were learned.

مكتب (*office* – not in training corpus)

- Start: ب ت ك م
- Apply rule 1 (ك+ت→كت): ب كت م
- Rules 2–4 don't match → no further merges

[ب, كت, م]

Key Point

An unseen word still tokenizes gracefully into a mix of learned subwords and single characters – never an OOV error.

BPE Application: Detokenization

Reversing Tokenization

Concatenate subword tokens in order and drop any end-of-word markers to recover the original text.

[ب, كت, م] $\xrightarrow{\text{concatenate}}$ مكتب

- Tokens within a word are joined directly, with no spaces
- A boundary marker (e.g. _ or ##) tells the detokenizer where word boundaries fall
- Full sentence reconstruction: join detokenized words with single spaces, restoring original spacing/punctuation

BPE Training Example: Corpus Initialization

The Corpus

Consider a small training corpus consisting of 5 word types with their respective frequencies:

- `low`: 5
- `lower`: 2
- `new`: 6
- `newer`: 3
- `wide`: 4

Initial Representation (Characters)

We add an end-of-word symbol (_) and split words into individual characters:

- 5: `l o w _`
- 2: `l o w e r _`
- 6: `n e w _`
- 3: `n e w e r _`

BPE Training Example: Counting Pairs (Iteration 1)

Step 1: Count Adjacent Pair Frequencies

Scanning the corpus for the most frequent bigram pair:

- (n, e): $6 + 3 = 9$
- (e, w): $6 + 3 = 9$
- (l, o): $5 + 2 = 7$
- (o, w): $5 + 2 = 7$
- (w, i): 4
- (i, d): 4
- (d, e): 4
- (e, r): $2 + 3 = 5$

First Merge Rule

The most frequent pairs are (n, e) and (e, w) with frequency 9. We choose **(n, e) $\rightarrow$ ne** as our first merge rule and update the vocabulary.

BPE Training Example: Updating the Corpus (Iteration 2)

Corpus After First Merge

Replacing `n e` with `ne`:

- 5: `l o w _`
- 2: `l o w e r _`
- 6: `ne w _`
- 3: `ne w e r _`
- 4: `w i d e _`

Step 2: Recounting Pairs

Recalculating frequencies yields the next highest pair:

- (`ne`, `w`): $6 + 3 = 9$ (Highest frequency)
- (`l`, `o`): 7
- (`o`, `w`): 7
- (`e`, `r`): 5

BPE Training Example: Iteration 3 & Final Outcome

Corpus After Second Merge

Replacing `ne w` with `new`:

- 5: `l o w _`
- 2: `l o w e r _`
- 6: `new _`
- 3: `new e r _`
- 4: `w i d e _`

Next Iteration Count

The pair (`l`, `o`) now has a frequency of $5 + 2 = 7$, making it the next candidate: **(`l`, `o`) $\rightarrow$ `lo`**.

Summary of Learned Rules

By continuing this iterative process based on corpus statistics, BPE progressively builds subword units like `ne`, `new`, `lo`, `low`, and `er` to handle vocabulary efficiently.

Hands-on Lab - Preprocessing Pipeline

Lab Deliverables

Students will implement a custom Python-based Arabic preprocessor:

1. Build regex modules for orthographic normalization.
2. Implement a Buckwalter bidirectional encoder/decoder.
3. Measure the impact of normalization on vocabulary size and out-of-vocabulary (OOV) rates using a raw Arabic news dataset (SANAD).
4. Implement subword BPE tokenization algorithm using the SANAD dataset.

Summary

1. Arabic NLP

Arabic presents challenges related to morphology, agglutination, orthography, diacritics, and dialect variation.

2. Tokenization

Three important representations are:

- Word-level tokens
- Character-level tokens
- Subword tokens such as BPE

3. Preprocessing

A basic Arabic preprocessing pipeline includes:

$$\boxed{\text{Normalization} \rightarrow \text{Diacritic Removal} \rightarrow \text{Tokenization}}$$

Thank You

Questions and Discussion