Tool Development For dictionary

Indian Languages native script lexicon generators

i. IIT Madras Parser
ii. CMUDICT english Parser

IIT Madras Parser

Installation Procudre for IITM Parser

Step 1: sudo apt-get install festival, valgrind
Step 2: cd unified_parser/unified
make
step 3: gcc *.c -o executable # install inbuilt bison flex
Step 4: valgrind ./unified-parser word 1 0 0 0

CMUDICT

Step 1: Install g2p_en , jamo
Step 2: run test.py missing module will be installed during run
Step 3: convert.py

convert.py

```
import re
from cmu2ilsl import *
f=open("/home/dhamaraiselvi/CMUDICT/voicebot_hieng", "r", encoding="utf8")
lines=f.readlines()

file_write = open("/home/dhamaraiselvi/CMUDICT/lexicon_voicebot_hieng.txt","w")

for line in lines:

    out_put = cmu2iitm(line)
```

```
    file_write.write("%s\t%s\n" % (line, out_put))
    print(line+ " " + line)
file_write.close()
```

cmu2ilsl.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import time
import sys
from phoneme_tokenizer import PhonemeTokenizer
g2p="g2p_en"
tokenizer = PhonemeTokenizer(g2p_type=g2p)

def g2p_en_lex(line):
    tokens = tokenizer.text2tokens(line)
    return(tokens)


def cmu2iitm(word):
    dict = {

    'AA0':'aa',
    'AA1':'aa',
    'AA2':'aa',
    'AA':'aa',
    'AE0':'ae',
    'AE1':'ae',
    'AE2':'ae',
    'AE':'ae',
    'AH0':'a',
    'AH1':'a',
    'AH2':'a',
    'AH':'a',
```

'AO0':'ao',
'AO1':'ao',
'AO2':'ao',
'AO':'ao',
'AW0':'au',
'AW1':'au',
'AW2':'au',
'AW':'au',
'AY0':'ai',
'AY1':'ai',
'AY2':'ai',
'AY':'ai',
'B':'b',
'CH':'c',
'C' : 'c',
'D':'dx',
'DH':'d',
'EH0':'ea',
'EH1':'ea',
'EH2':'ea',
'EH':'ea',
'ER0':'er',
'ER1':'er',
'ER2':'er',
'ER':'er',
'EY0':'ei',
'EY1':'ei',
'EY2':'ei',
'EY':'ei',
'F':'f',
'G':'g',
'HH':'h',
'IH0':'i',
'IH1':'i',
'IH2':'i',

'IH':'i',
'IY0':'ii',
'IY1':'ii',
'IY2':'ii',
'IY':'ii',
'JH':'j',
'K':'k',
'L':'l',
'M':'m',
'N':'n',
'NG':'ng',
'OW0':'oo',
'OW1':'oo',
'OW2':'oo',
'OW':'oo',
'OY0':'oi',
'OY1':'oi',
'OY2':'oi',
'OY':'oi',
'P':'p',
'R':'r',
'S':'s',
'SH':'sh',
'T':'tx',
'TH':'t',
'UH0':'u',
'UH1':'u',
'UH2':'u',
'UH' : 'u',
'UW0':'uu',
'UW1':'uu',
'UW2':'uu',
'UW':'uu',
'V':'w',
'W':'w',

```
    'Y':'y',
    'Z':'z',
    'ZH':'z',
    'H' : 'h',
    'J':'j',
    'Q' : 'q'
    }


    #i="anirban"
    lex=g2p_en_lex(word)
    print(lex)
    #newl.append(lex)

    line = (" ".join(lex) + "\n")
    print(line)

    output_text = []
    line = line.rstrip('\n')
    f1 = line.split()
    [output_text.append(str(dict[each])) for each in f1]
    #output_text = ' '.join(output_text)
    print(line, "--->", output_text)
    return(output_text)
```

phoneme_tokenizer.py

```
import logging
from pathlib import Path
import re
from typing import Iterable
from typing import List
from typing import Optional
from typing import Union
import warnings
```

```
import g2p_en
import jamo
from typeguard import check_argument_types

from abs_tokenizer import AbsTokenizer


g2p_choices = [
    None,
    "g2p_en",
    "g2p_en_no_space",
    "pyopenjtalk",
    "pyopenjtalk_kana",
    "pyopenjtalk_accent",
    "pyopenjtalk_accent_with_pause",
    "pyopenjtalk_prosody",
    "pypinyin_g2p",
    "pypinyin_g2p_phone",
    "espeak_ng_arabic",
    "espeak_ng_german",
    "espeak_ng_french",
    "espeak_ng_spanish",
    "espeak_ng_russian",
    "espeak_ng_greek",
    "espeak_ng_finnish",
    "espeak_ng_hungarian",
    "espeak_ng_dutch",
    "espeak_ng_english_us_vits",
    "espeak_ng_hindi",
    "espeak_ng_tamil",
    "g2pk",
    "g2pk_no_space",
    "korean_jaso",
    "korean_jaso_no_space",
```

```
]


def split_by_space(text) -> List[str]:
    if "   " in text:
        text = text.replace("   ", " <space> ")
        return [c.replace("<space>", " ") for c in text.split(" ")]
    else:
        return text.split(" ")


def pyopenjtalk_g2p(text) -> List[str]:
    import pyopenjtalk

    # phones is a str object separated by space
    phones = pyopenjtalk.g2p(text, kana=False)
    phones = phones.split(" ")
    return phones


def pyopenjtalk_g2p_accent(text) -> List[str]:
    import pyopenjtalk
    import re

    phones = []
    for labels in pyopenjtalk.run_frontend(text)[1]:
        p = re.findall(r"\-(.*?)\+.*?\/A:([0-9\-]+).*?\/F:.*?_([0-9]+)",
labels)
        if len(p) == 1:
            phones += [p[0][0], p[0][2], p[0][1]]
    return phones


def pyopenjtalk_g2p_accent_with_pause(text) -> List[str]:
    import pyopenjtalk
```

```
    import re

    phones = []
    for labels in pyopenjtalk.run_frontend(text)[1]:
        if labels.split("-")[1].split("+")[0] == "pau":
            phones += ["pau"]
            continue
        p = re.findall(r"\-(.*?)\+.*?\/A:([0-9\-]+).*?\/F:.*?_([0-9]+)",
labels)
        if len(p) == 1:
            phones += [p[0][0], p[0][2], p[0][1]]
    return phones


def pyopenjtalk_g2p_kana(text) -> List[str]:
    import pyopenjtalk

    kanas = pyopenjtalk.g2p(text, kana=True)
    return list(kanas)


def pyopenjtalk_g2p_prosody(text: str, drop_unvoiced_vowels:
bool = True) -> List[str]:
    """Extract phoneme + prosoody symbol sequence from input
full-context labels.

    The algorithm is based on `Prosodic features control by
symbols as input of
    sequence-to-sequence acoustic modeling for neural TTS`_ with
some r9y9's tweaks.

    Args:
        text (str): Input text.
        drop_unvoiced_vowels (bool): whether to drop unvoiced
vowels.
```

```
    Returns:
        List[str]: List of phoneme + prosody symbols.

    Examples:
        >>> from espnet2.text.phoneme_tokenizer import
pyopenjtalk_g2p_prosody
        >>> pyopenjtalk_g2p_prosody("こんにちは。")
        ['^', 'k', 'o', '[', 'N', 'n', 'i', 'ch', 'i', 'w', 'a', '$']

    .. _`Prosodic features control by symbols as input of sequence-
to-sequence acoustic
        modeling for neural TTS`:
https://doi.org/10.1587/transinf.2020EDP7104

    """
    import pyopenjtalk

    labels = pyopenjtalk.run_frontend(text)[1]
    N = len(labels)

    phones = []
    for n in range(N):
        lab_curr = labels[n]

        # current phoneme
        p3 = re.search(r"\-(.*?)\+", lab_curr).group(1)

        # deal unvoiced vowels as normal vowels
        if drop_unvoiced_vowels and p3 in "AEIOU":
            p3 = p3.lower()

        # deal with sil at the beginning and the end of text
        if p3 == "sil":
```

```
            assert n == 0 or n == N - 1
            if n == 0:
                phones.append("^")
            elif n == N - 1:
                # check question form or not
                e3 = _numeric_feature_by_regex(r"!(\d+)_", lab_curr)
                if e3 == 0:
                    phones.append("$")
                elif e3 == 1:
                    phones.append("?")
            continue
        elif p3 == "pau":
            phones.append("_")
            continue
        else:
            phones.append(p3)

        # accent type and position info (forward or backward)
        a1 = _numeric_feature_by_regex(r"/A:([0-9\-]+)\+",
lab_curr)
        a2 = _numeric_feature_by_regex(r"\+(\d+)\+", lab_curr)
        a3 = _numeric_feature_by_regex(r"\+(\d+)/", lab_curr)

        # number of mora in accent phrase
        f1 = _numeric_feature_by_regex(r"/F:(\d+)_", lab_curr)

        a2_next = _numeric_feature_by_regex(r"\+(\d+)\+", labels[n
+ 1])
        # accent phrase border
        if a3 == 1 and a2_next == 1 and p3 in "aeiouAEIOUNcl":
            phones.append("#")
        # pitch falling
        elif a1 == 0 and a2_next == a2 + 1 and a2 != f1:
            phones.append("]")
        # pitch rising
```

```
        elif a2 == 1 and a2_next == 2:
            phones.append("[")

    return phones


def _numeric_feature_by_regex(regex, s):
    match = re.search(regex, s)
    if match is None:
        return -50
    return int(match.group(1))


def pypinyin_g2p(text) -> List[str]:
    from pypinyin import pinyin
    from pypinyin import Style

    phones = [phone[0] for phone in pinyin(text,
style=Style.TONE3)]
    return phones


def pypinyin_g2p_phone(text) -> List[str]:
    from pypinyin import pinyin
    from pypinyin import Style
    from pypinyin.style._utils import get_finals
    from pypinyin.style._utils import get_initials

    phones = [
        p
        for phone in pinyin(text, style=Style.TONE3)
        for p in [
            get_initials(phone[0], strict=True),
            get_finals(phone[0], strict=True),
        ]
```

```
        if len(p) != 0
    ]
    return phones


class G2p_en:
    """On behalf of g2p_en.G2p.

    g2p_en.G2p isn't pickalable and it can't be copied to the other
processes
    via multiprocessing module.
    As a workaround, g2p_en.G2p is instantiated upon calling this
class.

    """

    def __init__(self, no_space: bool = False):
        self.no_space = no_space
        self.g2p = None

    def __call__(self, text) -> List[str]:
        if self.g2p is None:
            self.g2p = g2p_en.G2p()

        phones = self.g2p(text)
        if self.no_space:
            # remove space which represents word serapater
            phones = list(filter(lambda s: s != " ", phones))
        return phones


class G2pk:
    """On behalf of g2pk.G2p.
```

```
    g2pk.G2p isn't pickalable and it can't be copied to the other processes
    via multiprocessing module.
    As a workaround, g2pk.G2p is instantiated upon calling this class.

    """

    def __init__(
        self, descritive=False, group_vowels=False, to_syl=False, no_space=False
    ):
        self.descritive = descritive
        self.group_vowels = group_vowels
        self.to_syl = to_syl
        self.no_space = no_space
        self.g2p = None

    def __call__(self, text) -> List[str]:
        if self.g2p is None:
            import g2pk

            self.g2p = g2pk.G2p()

        phones = list(
            self.g2p(
                text,
                descriptive=self.descritive,
                group_vowels=self.group_vowels,
                to_syl=self.to_syl,
            )
        )
        if self.no_space:
            # remove space which represents word serapater
            phones = list(filter(lambda s: s != " ", phones))
```

```
        return phones


class Jaso:
    PUNC = "!'(),-.:;?"
    SPACE = " "

    JAMO_LEADS = "".join([chr(_) for _ in range(0x1100,
0x1113)])
    JAMO_VOWELS = "".join([chr(_) for _ in range(0x1161,
0x1176)])
    JAMO_TAILS = "".join([chr(_) for _ in range(0x11A8,
0x11C3)])

    VALID_CHARS = JAMO_LEADS + JAMO_VOWELS +
JAMO_TAILS + PUNC + SPACE

    def __init__(self, space_symbol=" ", no_space=False):
        self.space_symbol = space_symbol
        self.no_space = no_space

    def _text_to_jaso(self, line: str) -> List[str]:
        jasos = list(jamo.hangul_to_jamo(line))
        return jasos

    def _remove_non_korean_characters(self, tokens):
        new_tokens = [token for token in tokens if token in
self.VALID_CHARS]
        return new_tokens

    def __call__(self, text) -> List[str]:
        graphemes = [x for x in self._text_to_jaso(text)]
        graphemes =
self._remove_non_korean_characters(graphemes)
```

```
        if self.no_space:
            graphemes = list(filter(lambda s: s != " ", graphemes))
        else:
            graphemes = [x if x != " " else self.space_symbol for x in
graphemes]
        return graphemes


class Phonemizer:
    """Phonemizer module for various languages.

    This is wrapper module of
https://github.com/bootphon/phonemizer.
    You can define various g2p modules by specifying options for
phonemizer.

    See available options:

https://github.com/bootphon/phonemizer/blob/master/phonemizer/
phonemize.py#L32

    """

    def __init__(
        self,
        backend,
        word_separator: Optional[str] = None,
        syllable_separator: Optional[str] = None,
        phone_separator: Optional[str] = " ",
        strip=False,
        split_by_single_token: bool = False,
        **phonemizer_kwargs,
    ):
        # delayed import
        from phonemizer.backend import BACKENDS
```

```
        from phonemizer.separator import Separator

        self.separator = Separator(
            word=word_separator,
            syllable=syllable_separator,
            phone=phone_separator,
        )

        # define logger to suppress the warning in phonemizer
        logger = logging.getLogger("phonemizer")
        logger.setLevel(logging.ERROR)
        self.phonemizer = BACKENDS[backend](
            **phonemizer_kwargs,
            logger=logger,
        )
        self.strip = strip
        self.split_by_single_token = split_by_single_token

    def __call__(self, text) -> List[str]:
        tokens = self.phonemizer.phonemize(
            [text],
            separator=self.separator,
            strip=self.strip,
            njobs=1,
        )[0]
        if not self.split_by_single_token:
            return tokens.split()
        else:
            # "a: ab" -> ["a", ":", "<space>",  "a", "b"]
            # TODO(kan-bayashi): space replacement should be dealt
in PhonemeTokenizer
            return [c.replace(" ", "<space>") for c in tokens]


class PhonemeTokenizer(AbsTokenizer):
```

```
    def __init__(
        self,
        g2p_type: Union[None, str],
        non_linguistic_symbols: Union[Path, str, Iterable[str]] =
None,
        space_symbol: str = "<space>",
        remove_non_linguistic_symbols: bool = False,
    ):
        assert check_argument_types()
        if g2p_type is None:
            self.g2p = split_by_space
        elif g2p_type == "g2p_en":
            self.g2p = G2p_en(no_space=False)
        elif g2p_type == "g2p_en_no_space":
            self.g2p = G2p_en(no_space=True)
        elif g2p_type == "pyopenjtalk":
            self.g2p = pyopenjtalk_g2p
        elif g2p_type == "pyopenjtalk_kana":
            self.g2p = pyopenjtalk_g2p_kana
        elif g2p_type == "pyopenjtalk_accent":
            self.g2p = pyopenjtalk_g2p_accent
        elif g2p_type == "pyopenjtalk_accent_with_pause":
            self.g2p = pyopenjtalk_g2p_accent_with_pause
        elif g2p_type == "pyopenjtalk_prosody":
            self.g2p = pyopenjtalk_g2p_prosody
        elif g2p_type == "pypinyin_g2p":
            self.g2p = pypinyin_g2p
        elif g2p_type == "pypinyin_g2p_phone":
            self.g2p = pypinyin_g2p_phone
        elif g2p_type == "espeak_ng_arabic":
            self.g2p = Phonemizer(
                language="ar",
                backend="espeak",
                with_stress=True,
                preserve_punctuation=True,
```

```
    )
elif g2p_type == "espeak_ng_german":
    self.g2p = Phonemizer(
        language="de",
        backend="espeak",
        with_stress=True,
        preserve_punctuation=True,
    )
elif g2p_type == "espeak_ng_french":
    self.g2p = Phonemizer(
        language="fr-fr",
        backend="espeak",
        with_stress=True,
        preserve_punctuation=True,
    )
elif g2p_type == "espeak_ng_spanish":
    self.g2p = Phonemizer(
        language="es",
        backend="espeak",
        with_stress=True,
        preserve_punctuation=True,
    )
elif g2p_type == "espeak_ng_russian":
    self.g2p = Phonemizer(
        language="ru",
        backend="espeak",
        with_stress=True,
        preserve_punctuation=True,
    )
elif g2p_type == "espeak_ng_greek":
    self.g2p = Phonemizer(
        language="el",
        backend="espeak",
        with_stress=True,
        preserve_punctuation=True,
```

```
    )
elif g2p_type == "espeak_ng_finnish":
    self.g2p = Phonemizer(
        language="fi",
        backend="espeak",
        with_stress=True,
        preserve_punctuation=True,
    )
elif g2p_type == "espeak_ng_hungarian":
    self.g2p = Phonemizer(
        language="hu",
        backend="espeak",
        with_stress=True,
        preserve_punctuation=True,
    )
elif g2p_type == "espeak_ng_dutch":
    self.g2p = Phonemizer(
        language="nl",
        backend="espeak",
        with_stress=True,
        preserve_punctuation=True,
    )
elif g2p_type == "espeak_ng_hindi":
    self.g2p = Phonemizer(
        language="hi",
        backend="espeak",
        with_stress=True,
        preserve_punctuation=True,
    )
elif g2p_type == "espeak_ng_tamil":
    self.g2p = Phonemizer(
        language="ta",
        backend="espeak",
        with_stress=True,
        preserve_punctuation=True,
```

```
            )
        elif g2p_type == "g2pk":
            self.g2p = G2pk(no_space=False)
        elif g2p_type == "g2pk_no_space":
            self.g2p = G2pk(no_space=True)
        elif g2p_type == "espeak_ng_english_us_vits":
            # VITS official implementation-like processing
            # Reference: https://github.com/jaywalnut310/vits
            self.g2p = Phonemizer(
                language="en-us",
                backend="espeak",
                with_stress=True,
                preserve_punctuation=True,
                strip=True,
                word_separator=" ",
                phone_separator="",
                split_by_single_token=True,
            )
        elif g2p_type == "korean_jaso":
            self.g2p = Jaso(space_symbol=space_symbol,
no_space=False)
        elif g2p_type == "korean_jaso_no_space":
            self.g2p = Jaso(no_space=True)
        else:
            raise NotImplementedError(f"Not supported:
g2p_type={g2p_type}")

        self.g2p_type = g2p_type
        self.space_symbol = space_symbol
        if non_linguistic_symbols is None:
            self.non_linguistic_symbols = set()
        elif isinstance(non_linguistic_symbols, (Path, str)):
            non_linguistic_symbols = Path(non_linguistic_symbols)
            try:
```

```
                with non_linguistic_symbols.open("r", encoding="utf-
8") as f:
                    self.non_linguistic_symbols = set(line.rstrip() for line
in f)
            except FileNotFoundError:
                warnings.warn(f"{non_linguistic_symbols} doesn't
exist.")
                self.non_linguistic_symbols = set()
        else:
            self.non_linguistic_symbols = set(non_linguistic_symbols)
        self.remove_non_linguistic_symbols =
remove_non_linguistic_symbols

    def __repr__(self):
        return (
            f"{self.__class__.__name__}("
            f'g2p_type="{self.g2p_type}", '
            f'space_symbol="{self.space_symbol}", '

f'non_linguistic_symbols="{self.non_linguistic_symbols}"'
            ")"
        )

    def text2tokens(self, line: str) -> List[str]:
        tokens = []
        while len(line) != 0:
            for w in self.non_linguistic_symbols:
                if line.startswith(w):
                    if not self.remove_non_linguistic_symbols:
                        tokens.append(line[: len(w)])
                    line = line[len(w) :]
                    break
            else:
                t = line[0]
                tokens.append(t)
```

```
                line = line[1:]

        line = "".join(tokens)
        tokens = self.g2p(line)
        return tokens

    def tokens2text(self, tokens: Iterable[str]) -> str:
        # phoneme type is not invertible
        return "".join(tokens)
```

abs_tokenizer.py

```
from abc import ABC
from abc import abstractmethod
from typing import Iterable
from typing import List


class AbsTokenizer(ABC):
    @abstractmethod
    def text2tokens(self, line: str) -> List[str]:
        raise NotImplementedError

    @abstractmethod
    def tokens2text(self, tokens: Iterable[str]) -> str:
        raise NotImplementedError
```

cmu_phones.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

from phoneme_tokenizer import PhonemeTokenizer
```

```
def cmu_gen_phones(word):

    g2p="g2p_en_no_space"
    tokenizer = PhonemeTokenizer(g2p_type=g2p)
    tokens = tokenizer.text2tokens(word)
    print(" ".join(tokens) + "\n")
    return(" ".join(tokens) + "\n")
```