链滴

【NLP】bert 入门之分词源码解析

作者：jditlee

原文链接：https://ld246.com/article/1677661531243

来源网站：链滴

最近做bert模型做分类，涉及到模型上线，需要做文本的编码映射，然后就看了一下bert分词源码，这里做一下记录

bert编码方法总结：

其实就是分词+分词后的切片映射id

1.分词：

通过BasicTokenizer分词后，遍历每一个分词，将每一个词再经过WordpieceTokenizer分成子串

```
  def tokenize(self, text):
    split_tokens = []
    # 使用BasicTokenizer分词
    for token in self.basic_tokenizer.tokenize(text):
      # 使用WordpieceTokenizer将每一个分词切成子串
      for sub_token in self.wordpiece_tokenizer.tokenize(token):
        split_tokens.append(sub_token)
```

2.编码：

编码没什么好说的，就是一个切片映射成id的过程，加载词典，将最终的分词结果映射成词典id

```
def convert_by_vocab(vocab, items):
  """Converts a sequence of [tokens|ids] using the vocab."""
  output = []
  for item in items:
    output.append(vocab[item])
  return output
```

然后就是详细解读一下分词的两个方法

一、BasicTokenizer

大致流程：转成 unicode -> 去除各种奇怪字符 -> 处理中文 -> 空格分词 -> 去除多余字符和标点分 -> 再次空格分词

1.转成unicode:

如果是字符串直接返回字符串，如果是字节数组就转成utf-8的格式

```
def convert_to_unicode(text):
    """Converts `text` to Unicode (if it's not already), assuming utf-8 input."""
    if isinstance(text, str):
      return text
    elif isinstance(text, bytes):
      return text.decode("utf-8", "ignore")
    else:
      raise ValueError("Unsupported string type: %s" % (type(text)))
```

2.去除各种奇怪字符

遍历每一个字符：

1.过滤结束符0，替换符0xfffd，除\t\r\n外的控制字符

2.将所有空白字符转换为空格，包括标准空格、\t、\r、\n 以及 Unicode 类别为 Zs 的字符

```
def _clean_text(self, text):
    """Performs invalid character removal and whitespace cleanup on text."""
    output = []
    for char in text:
    # ord获取字符的码位
      cp = ord(char)
      # 过滤结束符，替换符，除\t\r\n外的控制字符
      if cp == 0 or cp == 0xfffd or _is_control(char):
        continue
      # 将所有空白字符转换为空格，包括标准空格、\t、\r、\n 以及 Unicode 类别为 Zs 的字符
      if _is_whitespace(char):
        output.append(" ")
      else:
        output.append(char)
    return "".join(output)


def _is_control(char):
  """Checks whether `chars` is a control character."""
  # These are technically control characters but we count them as whitespace
  # characters.
  if char == "\t" or char == "\n" or char == "\r":
    return False
  cat = unicodedata.category(char)
  if cat in ("Cc", "Cf"):
    return True
  return False


def _is_whitespace(char):
  """Checks whether `chars` is a whitespace character."""
  # \t, \n, and \r are technically contorl characters but we treat them
  # as whitespace since they are generally considered as such.
  if char == " " or char == "\t" or char == "\n" or char == "\r":
    return True
  cat = unicodedata.category(char)
  if cat == "Zs":
    return True
  return False
```

3.处理中文

遍历每一个字符：

1.获取字符的Unicode码位

2.通过码位判断是否是中文字符，见方法_is_hinese_char

3.如果是中文字符，在前后添加空格，否则原样输出

```
def _tokenize_chinese_chars(self, text):
    """Adds whitespace around any CJK character."""
    output = []
    for char in text:
    # 获取字符的Unicode码位
      cp = ord(char)
      # 如果是中文字符，在前后添加空格，否则原样输出
      if self._is_chinese_char(cp):
        output.append(" ")
        output.append(char)
        output.append(" ")
      else:
        output.append(char)
    return "".join(output)

# 通过码位来判断是否是中文字符
def _is_chinese_char(self, cp):
    """Checks whether CP is the codepoint of a CJK character."""
    # This defines a "chinese character" as anything in the CJK Unicode block:
    #   https://en.wikipedia.org/wiki/CJK_Unified_Ideographs_(Unicode_block)
    #
    # Note that the CJK Unicode block is NOT all Japanese and Korean characters,
    # despite its name. The modern Korean Hangul alphabet is a different block,
    # as is Japanese Hiragana and Katakana. Those alphabets are used to write
    # space-separated words, so they are not treated specially and handled
    # like the all of the other languages.
    if ((cp >= 0x4E00 and cp <= 0x9FFF) or  #
        (cp >= 0x3400 and cp <= 0x4DBF) or  #
        (cp >= 0x20000 and cp <= 0x2A6DF) or  #
        (cp >= 0x2A700 and cp <= 0x2B73F) or  #
        (cp >= 0x2B740 and cp <= 0x2B81F) or  #
        (cp >= 0x2B820 and cp <= 0x2CEAF) or
        (cp >= 0xF900 and cp <= 0xFAFF) or  #
        (cp >= 0x2F800 and cp <= 0x2FA1F)):  #
      return True

    return False
```

4.空格分词

1.去掉两表空格

2.如果空就直接返回空列表，否则就按空格分词，返回分词列表

这一步将字符串变成了字符数组

```
def whitespace_tokenize(text):
  """Runs basic whitespace cleaning and splitting on a piece of text."""
  text = text.strip()
  if not text:
    return []
  tokens = text.split()
  return tokens
```

5.去除多余字符和标点分词

1. token转小写，然后去除变音符号

2. 将带有标点符号的词串再次根据标点符号分词

这里主要说一下变音符号：eg:'ā' 它是由'a'和'-'两个字符组成,代码中unicodedata.normalize("NFD", ext)其实就是把'ā'分解成'a'和'-'，即：把一个码位拆成两个码位

```
split_tokens = []
for token in orig_tokens:
  if self.do_lower_case:
    # 将token转成小写
    token = token.lower()
    # 去除变音符号
    token = self._run_strip_accents(token)
  # 标点分词
  split_tokens.extend(self._run_split_on_punc(token))


def _run_strip_accents(self, text):
    """Strips accents from a piece of text."""
    # 返回字符串的规范分解形式，unicodedata是python内置库：相当于把一个码位拆成两个码位
    text = unicodedata.normalize("NFD", text)
    output = []
    for char in text:
      # 返回字符的Unicode类别
      cat = unicodedata.category(char)
      # 过滤类别为Mn的字符，变音字符就属于这一类
      if cat == "Mn":
        continue
      output.append(char)
    return "".join(output)

# 标点分词，按照标点符号分词
# eg: （start_new）将被分词为['(','start','_','new',')']
def _run_split_on_punc(self, text):
    """Splits punctuation on a piece of text."""
    chars = list(text)
    i = 0
    start_new_word = True
    output = []
    while i < len(chars):
      char = chars[i]
      # 判断是否是标点符号
      if _is_punctuation(char):
        output.append([char])
        start_new_word = True
      else:
        if start_new_word:
          output.append([])
        start_new_word = False
        output[-1].append(char)
      i += 1

    return ["".join(x) for x in output]

# 判断是否是标点符号，通过码位和Unicode类别来判断
def _is_punctuation(char):
  """Checks whether `chars` is a punctuation character."""
  cp = ord(char)
  # We treat all non-letter/number ASCII as punctuation.
  # Characters such as "^", "$", and "`" are not in the Unicode
  # Punctuation class but we treat them as punctuation anyways, for
  # consistency.
  # 码位在[33,47],[58,64],[91,96],[123,126]的都是标点符号，也就是这些字符!"#$%&'()*+,-./:;<=>
@[\]^_`{|}~
  if ((cp >= 33 and cp <= 47) or (cp >= 58 and cp <= 64) or
      (cp >= 91 and cp <= 96) or (cp >= 123 and cp <= 126)):
    return True
  cat = unicodedata.category(char)
  # Unicode类别以P开头的也是标点符号
  if cat.startswith("P"):
    return True
  return False
```

原文链接：【NLP】bert 入门之分词源码解析

6.再次空格分词，同第四步

用标准空格拼接上一步的处理结果，再执行空格分词

```
output_tokens = whitespace_tokenize(" ".join(split_tokens))
```

二、WordpieceTokenizer

WordpieceTokenizer是在BasicTokenizer的基础上再次进行分词，主要是对英文再次分为一个个子token，通过匹配vocab词典，使用greedy longest-match-first algorithm 贪婪最长优先匹配算法，一个词拆分成多个词

当然，对于中文来说，没必要使用WPT来分词了，因为一个字已经没法再子了

大概步骤：转成Unicode->空格分词->异常词处理->加载词典->匹配算法

转成Unicode->空格分词都和BasicTokenizer中的一样，主要说后面三步：

1.异常词处理

1.转成Unicode

2.遍历空格分词后的每一个词

3.判断词是否超过设置的最大字符长度(模型设置为200)

4.超过就标记该词为[UNK]

```
  text = convert_to_unicode(text)

  output_tokens = []
  for token in whitespace_tokenize(text):
    chars = list(token)
    if len(chars) > self.max_input_chars_per_word:
      output_tokens.append(self.unk_token)
      continue
```

2.加载词典

1.读取词典文件：

2.将每一行的值去除两端空格作为词典的key，索引作为词典的value，索引从0开始

```
def load_vocab(vocab_file):
  """Loads a vocabulary file into a dictionary."""
  vocab = collections.OrderedDict()
  index = 0
  with tf.gfile.GFile(vocab_file, "r") as reader:
    while True:
      token = convert_to_unicode(reader.readline())
      if not token:
        break
      token = token.strip()
      vocab[token] = index
      index += 1
  return vocab
```

3.匹配算法（greedy longest-match-first algorithm）

greedy longest-match-first algorithm 主要步骤：

1. 定义start=0,end=len(word)

2. 从最长子串词本身[sart:end
开始判断是否存在词典中

3. 以end为标记从右至左扫描，判断子串[sart:
nd]是否存在词典中

4. 如果在，切分子串，修改start=end，end=len(word)标记，再次执行第三步

5. 不存在，标记为bad,整个词赋值[UNK]

6. 子串start不为0需要添加'##'作为开头

其实就是双指针从后向前扫描，非开头的子串以##作为开头，如果有一个子串不在词表中，就将整个赋值为[UNK]，然后就是将匹配子串作为最终分词结果

```
  is_bad = False
  start = 0
  sub_tokens = []
  while start < len(chars):
    end = len(chars)
    cur_substr = None
    while start < end:
      substr = "".join(chars[start:end])
      if start > 0:
        # 非开头子串以##作为开头
        substr = "##" + substr
      if substr in self.vocab:
        cur_substr = substr
        break
      end -= 1
    # 子串不存在词典中，跳出循环，标记为[UNK]
    if cur_substr is None:
      is_bad = True
      break
    sub_tokens.append(cur_substr)
    start = end

  if is_bad:
    output_tokens.append(self.unk_token)
  else:
    output_tokens.extend(sub_tokens)
```