# 第3章 文本相似度计算

文本相似度计算是自然语言处理的基本任务。文本相似度在狭义上是指两个字符串的相似情况，广义上是指文本在所含语义上的相似程度，可以广泛应用于自然语言处理任务的多个领域。如在机器翻译领域可以用于评价翻译结果的准确程度，在搜索引擎领域可以用于衡量检索文本与被检索文本之间的相关程度，在文本分类领域可以用于评价文本间内容的相似程度，在问答领域可以用于评定用户输入问题与问答库中问题的相似程度及问题与答案的相关程度等。本章主要介绍常见的文本相似度算法及用于比较文本相似度的模型。

本章主要涉及到的知识点有：

* 文本相似度计算介绍。
* 基于字符的文本相似度算法。
* 基于语义的文本相似度算法。
* 语义匹配模型。

注意：本章代码地址：xxx

## 3.1 文本相似度计算介绍

在信息爆炸时代，如何从海量信息中获取需要的信息成为一种亟待解决的问题，为了解决这个问题而出现的搜索引擎、推荐系统、问答系统、文本分类、信息检索等。这些应用场景的关键技术就是文本相似度计算。

自然语言处理中的许多问题都可以抽象为文本相似度计算的问题，例如网页搜索可以抽象为网页内容与用户搜索Query的文本相似度计算问题；在社区问答网站中，往往会出现越来越多的相似重复问题，文本相似度计算可以帮助检测这些重复问题，一方面降低数据冗余，减少数据存储维护的成本；另一方面减少重复问题可以帮助用户快速检索到想要获取的信息，提升用户体验。同样文本相似度计算也可以应用于网页搜索中，

文本相似度计算方法一般可以分为三类，一种是基于字符的方法，通常是一些较为传统的方法。另一种是基于语义的方法，通常是基于机器学习及深度学习的方法。其余方法是通过对文本进行特征抽取进而比较相似度的方法。

常用的文本相似度计算方法如下图所示：

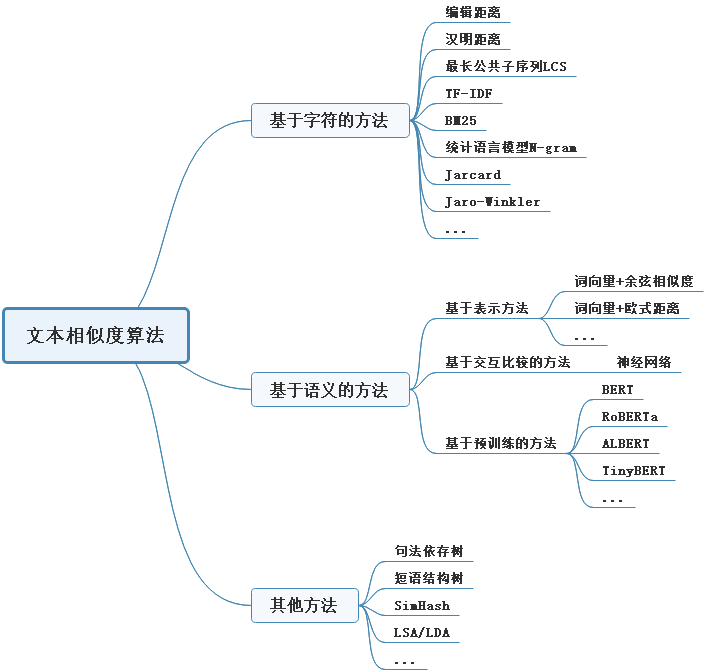


图3.1 文本相似度计算方法分类

## 3.2 基于字符的文本相似度计算

本节主要介绍基于字符的文本相似度计算方法及其优缺点。给定两个句子与，若两个句子中包含的字符种类及字符间的排列顺便均相同，那么可以认为两个句子相同，若两个句子中字符不同或者字符排列顺序不同，那么如何衡量两个句子之间的相似情况？

在机器学习出现之前，传统的文本相似度是基于句子中的字符信息及词频信息进行计算的。例如通过最长公共子序列计算公共字符串占全部文本比例，进而比较文本的相似度以及Jaccard距离方法；基于字符变换的距离来计算相似度的编辑距离、汉明距离；基于信息检索中的TF-IDF、BM25、语言模型等方法，主要解决字面相似度问题。这些方法由于计算简单，适用范围广，到现在依旧是很多场景下的优秀基准模型。

下面介绍几种典型的基于字符的文本相似度计算方法。

### 3.2.1 编辑距离

编辑距离，又称为Levenshtein距离是指由原字符串转换为另外一个字符串所需要的最小编辑次数或最小代价，编辑即是对字符串指定位置的单个字符进行插入、删除和替换的操作。

考虑长度分别为m和n的两个字符串与，构造矩阵，利用动态规划方法获得该矩阵元素即

在该矩阵中，矩阵右下角元素LD(n,m)即为字符串与之间的编辑距离。

如将字符串“abc”转换为“ab”，只需要将字符串“abc”的“c”删除即可，因此这两个字符串之间的编辑距离为1；同理，将字符串“abd”转换为“abc”，只需要将字符“d”替换为“c”，故编辑距离为1；

**代码3.1 编辑距离**

#!/usr/bin/env python  
# \_\*\_ coding:utf-8 \_\*\_  
import numpy as np  
  
  
def edit\_distance(str1, str2):  
 *'''  
 利用动态规划计算编辑距离* ***:param*** *str1:* ***:param*** *str2:* ***:return****:   
 '''* len1 = len(str1)  
 len2 = len(str2)  
 dp = np.zeros((len1 + 1, len2 + 1))  
 for i in range(len1 + 1):  
 dp[i][0] = i  
 for j in range(len2 + 1):  
 dp[0][j] = j  
 # 利用计算编辑距离的公式来计算  
 for i in range(1, len1 + 1):  
 for j in range(1, len2 + 1):  
 if str1[i - 1] == str2[j - 1]:  
 delta = 0  
 else:  
 delta = 1  
 dp[i][j] = min(dp[i - 1][j - 1] + delta, min(dp[i - 1][j] + 1, dp[i][j - 1] + 1))  
 return dp[len1][len2]  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 str1 = 'hello'  
 str2 = 'world'  
 # 计算str1与str2之间的编辑距离  
 ed = edit\_distance(str1, str2)  
 print(ed)

### 3.2.2 汉明距离

与编辑距离算法相近的是汉明距离，汉明距离主要应用于编码通讯领域，在误差检测和校正码处理上有很好的效果。对于两个等长字符串之间的汉明距离是两个字符串对应位置的不同字符的个数。

汉明距离的核心原理就是如何通过字符替换（最初应用在通讯中实际上是二进制的0-1替换），能将一个字符串替换成另外一个字符串。

如“karolin” 和 “kathrin” 的汉明距离为(字符2 3 4替换)

“karolin” 和 “kerstin” 的汉明距离为(字符1 3 4替换)

**代码3.2 汉明距离**

#!/usr/bin/env python

# \_\*\_ coding:utf-8 \_\*\_

def hamming\_distance(str1, str2):

'''

Return the Hamming distance between equal-length sequences

:param str1:

:param str2:

:return:

'''

if len(str1) != len(str2):

raise ValueError("Undefined for sequences of unequal length")

return sum(el1 != el2 for el1, el2 in zip(str1, str2))

if \_\_name\_\_ == "\_\_main\_\_":

str1 = 'abc'

str2 = 'abd'

hd = hamming\_distance(str1, str2)

print(hd)

### 3.2.3 TF-IDF

TF-IDF（term frequency–inverse document frequency ）是一种用于资讯检索与文本挖掘的常用加权技术。TF-IDF是一种统计方法，用以评估一个字词对于一个文档集或一个语料库中的其中一份文件的重要程度。字词的重要性随着它在文件中出现的次数成正比增加，但同时会随着它在语料库中出现的频率成反比下降。

TF：在一份给定的文件里，词频（term frequency，TF）指的是某一个给定的词语在该文件中出现的次数。对于在某一特定文件里的词语ti来说，它的TF可表示为：

TF = 某个词在文档中的出现次数/文档中的总词数

​ IDF ：逆向文件频率（inverse document frequency，IDF）是一个词语普遍重要性的度量。某一特定词语的IDF，可以由总文件数目除以包含该词语之文件的数目，再将得到的商取对数得到：

IDF = log(语料库中的总文档数/语料库中出现该词的文档数)

最终，TF-IDF=TF \* IDF

**代码3.3 TF-IDF**

#!/usr/bin/env python

# \_\*\_ coding:utf-8 \_\*\_

import numpy as np

from collections import defaultdict

class TFIDF(object):

'''

用于计算文档TF-IDF的类

'''

def \_\_init\_\_(self, corpus, word\_sep=' ', smooth\_value=0.01, scale=False):

assert isinstance(corpus, list), 'Not support this type corpus.'

self.corpus = corpus

self.vob = defaultdict(int)

self.word\_sep = word\_sep

self.smooth\_value = smooth\_value

self.doc\_cnt = defaultdict(set)

self.scale = scale

def get\_tf\_idf(self):

# 获取词表

for i, line in enumerate(self.corpus):

if isinstance(line, str):

line = line.split(self.word\_sep)

for w in line:

self.vob[f'{i}\_{w}'] += 1

self.doc\_cnt[w].add(i)

# 计算TF-IDF

output = np.zeros((len(self.corpus), len(self.vob)))

for i, line in enumerate(self.corpus):

if isinstance(line, str):

line = line.split(self.word\_sep)

tmp\_size = len(line)

for j, w in enumerate(self.vob.keys()):

w\_ = w.split('\_')[1]

if w\_ in line:

output[i, j] = self.vob[w] / tmp\_size \* np.log(

(self.smooth\_value + len(self.corpus)) / (self.smooth\_value + len(self.doc\_cnt[w\_])) + 1)

if self.scale:

output = (output - output.mean(axis=1).reshape(len(self.corpus), -1)) / output.std(axis=1).reshape(

len(self.corpus), -1)

return output

if \_\_name\_\_ == '\_\_main\_\_':

# 每个列表代表一个文档

corpus = [['this', 'is', 'a', 'simple', 'tfidf', 'code', 'but', 'code', 'might', 'has', 'bugs'],

['python', 'is', 'a', 'code', 'language', 'not', 'human', 'language'],

['learning', 'python', 'make', 'things', 'simple', 'but', 'not', 'simple', 'enough']]

result = TFIDF(corpus)

print(result.get\_tf\_idf())

### 3.2.4 BM25

BM25算法是一种应用广泛的对TF-IDF的改进算法，解决了TF-IDF偏向于长文档的问题。传统的TF值理论上是可以无限大的。而BM25与之不同，它在TF计算方法中增加了一个常量k，用来限制TF值的增长极限。

BM25还引入了平均文档长度的概念，单个文档长度对相关性的影响力与它和平均长度的比值有关系引入另外两个参数：L和b。L是文档长度与平均长度的比值。如果文档长度是平均长度的2倍，则L＝2。b是一个常数，它的作用是规定L对评分的影响有多大。加了L和b的TF计算公式变为:

TF = ((k + 1) \* tf) / (k \* (1.0 - b + b \* L) + tf)

IDF部分计算方法与TF-IDF中相同。 最终，BM25=TF \* IDF

**代码3.4 BM25**

#!/usr/bin/env python

# \_\*\_ coding:utf-8 \_\*\_

import re

import codecs

import math

import jieba

# 测试文本

text = '''

自然语言处理是计算机科学领域与人工智能领域中的一个重要方向。

它研究能实现人与计算机之间用自然语言进行有效通信的各种理论和方法。

自然语言处理是一门融语言学、计算机科学、数学于一体的科学。

因此，这一领域的研究将涉及自然语言，即人们日常使用的语言，

所以它与语言学的研究有着密切的联系，但又有重要的区别。

自然语言处理并不是一般地研究自然语言，

而在于研制能有效地实现自然语言通信的计算机系统，

特别是其中的软件系统。因而它是计算机科学的一部分。

'''

class BM25(object):

'''

计算BM25的类

'''

def \_\_init\_\_(self, docs):

self.D = len(docs)

self.avgdl = sum([len(doc) + 0.0 for doc in docs]) / self.D

self.docs = docs

# 列表的每一个元素是一个dict，dict存储着一个文档中每个词的出现次数

self.f = []

# 存储每个词及出现了该词的文档数量

self.df = {}

# 存储每个词的idf值

self.idf = {}

self.k1 = 1.5

self.b = 0.75

self.init()

def init(self):

for doc in self.docs:

tmp = {}

for word in doc:

# 存储每个文档中每个词的出现次数

tmp[word] = tmp.get(word, 0) + 1

self.f.append(tmp)

for k in tmp.keys():

self.df[k] = self.df.get(k, 0) + 1

for k, v in self.df.items():

self.idf[k] = math.log(self.D - v + 0.5) - math.log(v + 0.5)

def sim(self, doc, index):

'''

根据算法计算BM25

:param doc:

:param index:

:return:

'''

score = 0

for word in doc:

if word not in self.f[index]:

continue

d = len(self.docs[index])

# TF = ((k + 1) \* tf) / (k \* (1.0 - b + b \* L) + tf)

score += (self.idf[word] \* self.f[index][word] \* (self.k1 + 1)

/ (self.f[index][word] + self.k1 \* (1 - self.b + self.b \* d

/ self.avgdl)))

return score

def simall(self, doc):

scores = []

for index in range(self.D):

score = self.sim(doc, index)

scores.append(score)

return scores

def filter\_stop(words):

'''

过滤停用词表中的词

:param words:

:return:

'''

stop = set()

fr = codecs.open('stopwords.txt', 'r', 'utf-8')

for word in fr:

stop.add(word.strip())

fr.close()

re\_zh = re.compile('([\u4E00-\u9FA5]+)')

return list(filter(lambda x: x not in stop, words))

def get\_sentences(doc):

'''

对输入文本进行预处理，将其拆分为多个doc格式

:param doc:

:return:

'''

line\_break = re.compile('[\r\n]')

delimiter = re.compile('[，。？！；]')

sentences = []

for line in line\_break.split(doc):

line = line.strip()

if not line:

continue

for sent in delimiter.split(line):

sent = sent.strip()

if not sent:

continue

sentences.append(sent)

return sentences

if \_\_name\_\_ == '\_\_main\_\_':

sents = get\_sentences(text)

print(sents)

doc = []

for sent in sents:

words = list(jieba.cut(sent))

words = filter\_stop(words)

doc.append(words)

print(doc)

s = BM25(doc)

print(s.f)

print(s.idf)

print(s.simall(['自然语言', '计算机科学', '领域', '人工智能', '领域']))

### 3.2.5 统计语言模型

统计语言模型用于计算给定一个问题，另一个问题由其生成的概率。通过引入马尔可夫假设，我们可以认为一句话中每个单词出现的概率只与它前面n个词有关，整句的概率就是各个词出现概率的乘积。该模型被称为N-gram语言模型。

统计语言模型通常对语料库的大小有着较强的要求，通常来说，随着N-gram模型中n的增加，模型对于概率的估计会更加准确，但是需要的数据量也会急剧增加，所以，常用的统计语言模型通常为2-gram模型或者one-gram模型。

**代码3.5 NGram**

#!/usr/bin/env python

# \_\*\_ coding:utf-8 \_\*\_

class NGram():

'''

计算NGram的类

'''

# 设置参数n，默认为2

def \_\_init\_\_(self, n=2):

self.n = n

def distance(self, s0, s1):

'''

计算输入文本的NGram距离

:param s0:

:param s1:

:return:

'''

if s0 is None:

raise TypeError("Argument s0 is NoneType.")

if s1 is None:

raise TypeError("Argument s1 is NoneType.")

if s0 == s1:

return 0.0

special = '\n'

sl = len(s0)

tl = len(s1)

if sl == 0 or tl == 0:

return 1.0

cost = 0

if sl < self.n or tl < self.n:

for i in range(min(sl, tl)):

if s0[i] == s1[i]:

cost += 1

return 1.0 \* cost / max(sl, tl)

sa = [''] \* (sl + self.n - 1)

for i in range(len(sa)):

if i < self.n - 1:

sa[i] = special

else:

sa[i] = s0[i - self.n + 1]

p = [0.0] \* (sl + 1)

d = [0.0] \* (sl + 1)

t\_j = [''] \* self.n

for i in range(sl + 1):

p[i] = 1.0 \* i

for j in range(1, tl + 1):

if j < self.n:

for ti in range(self.n - j):

t\_j[ti] = special

for ti in range(self.n - j, self.n):

t\_j[ti] = s1[ti - (self.n - j)]

else:

t\_j = s1[j - self.n:j]

d[0] = 1.0 \* j

for i in range(sl + 1):

cost = 0

tn = self.n

for ni in range(self.n):

if sa[i - 1 + ni] != t\_j[ni]:

cost += 1

elif sa[i - 1 + ni] == special:

tn -= 1

ec = cost / tn

d[i] = min(d[i - 1] + 1, p[i] + 1, p[i - 1] + ec)

p, d = d, p

return p[sl] / max(tl, sl)

if \_\_name\_\_ == '\_\_main\_\_':

twogram = NGram(2)

print(twogram.distance('ABCD', 'ABTUIO'))

s1 = 'Adobe CreativeSuite 5 Master Collection from cheap 4zp'

s2 = 'Adobe CreativeSuite 5 Master Collection from cheap d1x'

fourgram = NGram(4)

print(fourgram.distance(s1, s2))

### 3.2.6 Jaccard距离

两个集合A和B的交集元素在A，B的并集中所占的比例，称为两个集合的Jaccard相似系数，用符号J(A,B)表示。Jaccard相似系数是衡量两个集合的相似度一种指标。

Jaccard距离是基于Jaccard系数来计算的。

**代码3.6 Jaccard**

#!/usr/bin/env python

# \_\*\_ coding:utf-8 \_\*\_

import jieba

def Jaccrad(model, reference):

'''

计算jaccard系数

:param model: 候选句子

:param reference: 源句子

:return:

'''

# 对句子分词 默认精准模式

terms\_reference = jieba.cut(reference)

terms\_model = jieba.cut(model)

# 去重，如果不需要就改为list

grams\_reference = set(terms\_reference)

grams\_model = set(terms\_model)

# 两个句子的并集

temp = 0

for i in grams\_reference:

if i in grams\_model:

temp = temp + 1

# 两个句子的交集

fenmu = len(grams\_model) + len(grams\_reference) - temp

# 两个句子Jaccard距离

jaccard\_coefficient = float(temp / fenmu)

return jaccard\_coefficient

if \_\_name\_\_ =='\_\_main\_\_':

a = "香农在信息论中提出的信息熵定义为自信息的期望"

b = "信息熵作为自信息的期望"

jaccard\_coefficient = Jaccrad(a, b)

print(jaccard\_coefficient)

## 3.3 基于语义的文本相似度计算

基于字符的文本相似度计算方法尽管计算方法简单，但是在实际应用场景中依然有很大的局限性。主要表现在以下两个方面。

1. 语言的一词多义及多词同义问题，同一个词在不同的语境场景中可以表达不同的语义信息，例如“苹果”既可以指一种叫“苹果”的水果也可以表示“苹果”科技公司及其产品；同理，不同的表达也可以表示相同的信息，例如“的士”、“计程车”、“滴滴”都可以表示出租车。
2. 语言的组合结构问题，词是自然语言中的最小语义单位，通过词来组成句子和篇章，不同的词序及语法结构可以表达不同的语义。例如“奶油蛋糕”和“蛋糕奶油”由于词序不同，表达了不同的语义信息。“从北京到上海的高铁”和“从上海到北京的高铁”，虽然所包含的词相同，但语义信息并不相同。

随着机器学习的发展，尤其是深度学习模型在自然语言处理领域的广泛应用，相较于传统的方法，可以更精确的计算语义层级的相似度，解决基于字符的文本相似度计算方法存在的语义局限和结构局限的问题。

基于语义文本相似度计算的模型主要包括传统的语义匹配模型和深度语义匹配模型。其中传统的语义匹配模型主要指隐式模型，典型的如LSA、LDA主题模型等。

深度语义匹配模型主要有三种类型，分别是基于表示的方法，基于交互比较的方法和基于预训练的方法。

### 3.3.1 基于表示的方法

基于表示的方法是指将字符输入的句子通过深度学习模型表示为向量空间中的向量，进而在向量空间中比较向量的相似度进而比较句子的语义相似度。基于表示的方法其比较结构一般为三层结构，分别是嵌入层、编码层与比较层，一般架构如下图所示：

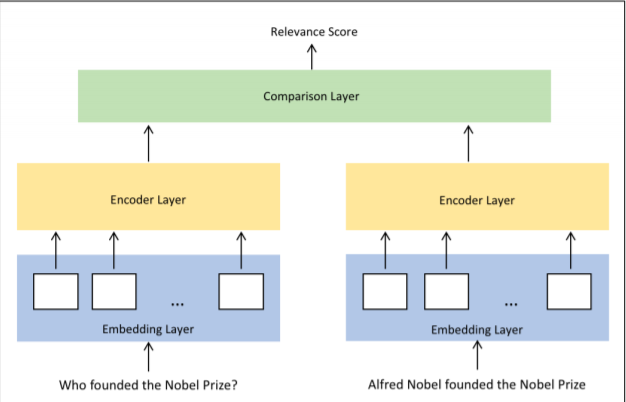


图3.2 基于表示的方法

嵌入层（Embedding Layer）将输入句子的每个词的one-hot编码转换为d维的词向量编码，通常使用word2vec或者glove词向量来实现。

编码层（Encoder Layer）将句子整体转换为向量表示，通常会使用BOW、CNN、RNN、LSTM及BiLSTM网络和池化层来构建句子的向量表示。

比较层（Comparison Layer）对两个句子转换后的向量进行比较，通常使用余弦相似度或者向量距离如曼哈顿距离来衡量两个句子的相似度。

基于表示的方法的典型模型即为微软提出的DSSM模型，这种方法的优点在于简单方便，而且可以将Embedding的操作转为线下，提高线上的效率，而且将句子向量表示后便于可视化，后续还可以应用于句子聚类等任务中。

基于表示的方法其缺点在于句子编码表示的过程分别由独立的嵌入层与编码层来实现，两个输入句子彼此没有交互，不利于结合上下文信息。

### 3.3.2 基于交互比较的方法

基于交互比较的方法通常比较输入句子的较小单位（例如词），然后将比较结果汇总（例如，通过CNN或RNN），以做出最终决定。与基于表示的方法相比，基于交互比较的方法可以捕获输入句子之间的更多交互功能，因此在对TrecQA等公共数据集进行评估时，通常具有更好的性能。一个典型的基于交互比较的方法的模型如下图所示：

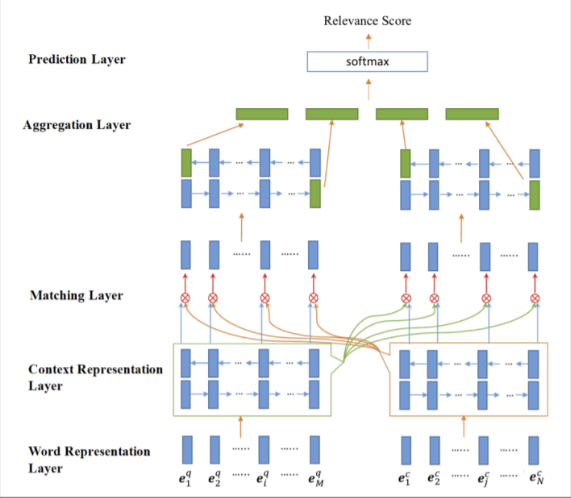


图3.3 基于交互比较的方法

该模型为五层结构，相比较于基于表示的方法网络，增加了上下文表示层与聚合层。

单词表示层（Word Representation Layer）与嵌入层的作用一样，将句子输入词的one-hot编码转换为d维向量。通常也是使用词向量技术来完成，如GloVe或word2vec。

上下文表示层（Contex Representation Layer）是为输入句子中的每个位置获取一个新的表示形式，该表示形式除了捕获该位置的单词以外，还捕获一些上下文信息。这样不仅考虑句子中词的信息，也考虑了句子中词序的信息，可以采用BiLSTM来完成。

匹配层（Matching Layer）将一个句子的每个上下文表示与另一句子的所有上下文表示进行比较。该层的输出是两个匹配向量序列，其中每个匹配向量对应于一个句子的一个位置与另一个句子的所有位置的比较结果。

聚合层（Aggregation Layer） 汇总来自上一层的比较结果，将匹配向量的两个序列聚合为固定长度向量。

预测层（Prediction Layer）做出最终预测。最终应用softmax函数获得最终的句子相似度比较结果。

这种基于交互比较的方法在构建神经网络比较的时候，将输入的句子对进行交互，可以更好的获取文本表示。

### 3.3.3 基于预训练的方法

随着BERT及其改进模型XLNET、RoBERTa、ALBERT、ERNIE、TinyBERT等预训练模型的出现，由于其在大规模的语料库上进行过训练，所以能捕捉到更多的语义信息。而且在各种自然语言处理任务中均取得了很好的效果，因此基于预训练的方法已经成为文本语义相似度计算的最优解决策略。这种方法将语义相似度计算任务作为一个二分类任务（相似/不相似）来解决，通过[CLS]标记将两个句子拼接作为模型的输入，通过[SEP]标记两个句子的连接位置，输出为两者之间相似的概率。 如下图所示：

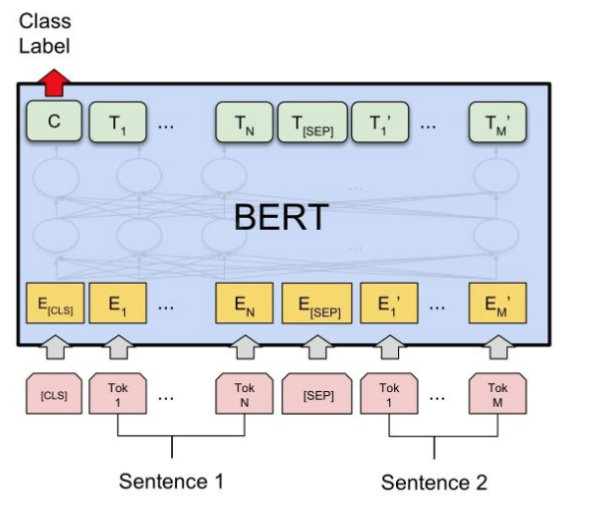


图3.4 基于预训练的方法

输入层：BERT模型的输入为两个句子的拼接，在句子开始位置加入[CLS]标记，在两个句子之间加入[SEP]标记。利用BPE算法等进行分词，得到BERT模型的输入特征向量。

相似度计算：将特征向量输入BERT后，经计算将得到BERT的输出（句子中每个词的向量表示），取[CLS]标记的向量表示作为输出，通过一个单层或多层全连接神经网络，得到两个文档的相似度得分（相似的概率)。

## 3.4 其他文本相似度计算

除了上面介绍的文本相似度计算方法，其余还有基于句法结构的方式，通过分析一个句子中的各个成分的依存关系得到的有效配对，再计算有效配对之间的相似度得到句子的相似度。这种方法将句法特征信息引入，可以直观展现句子内部的结构，但是没有考虑语义在句子中的作用，而且现有的依存句法分析准确率不高，在实际应用中的效果并不是很突出。

还有一种方法是对文本降维，如给文本生成一个SimHash值，即指纹(fingure print)，通过SimHash的值来评估两个文本的相似情况。

此外还有通过《同义词词林》、《知网》、WordNet等语义资源对通用词汇进行扩展，识别问句中的同义词和近义词信息，从而可以更精确的计算句子间的相似度。但该方法受限于知识源，不适用于特定领域，需要特定的专家基于语言学知识来处理，需要耗费较高的人力成本。

## 3.5 语义匹配模型训练

基于表示的语义匹配模型训练中，常用的训练方法为point-wise与 pair-wise，采用监督学习的方法来训练模型。下面主要介绍一下point-wise与pair-wise的训练模式。

### 3.5.1point-wise

Point-wise方法的主要思路是将文本相似度计算转换为二分类问题，其中训练过程中的输入为句子对，句子对之间的真实相似度作为标签信息，输出为二者之间相关度的预测得分信息。

其中用于训练point-wise的数据由三列组成，以制表符（'\t'）分隔，第一列和第二列是需要比对的句对，第三列是标签（0表示不相似，1表示相似），文件为 utf8 编码。字段信息如表3.1所示：

表3.1 point-wise训练数据格式字段介绍

|  |  |  |
| --- | --- | --- |
| 字段 | 说明 | 备注 |
| Sentence1 | 句子1 |  |
| Sentence2 | 句子2 |  |
| Label | 句子语义相似标签 | 1表示两个句子相似，反之为0 |

部分数据如下：

Sentence1,sentence2,label

宝马启动空调就开了 宝马启动空调 1

宝马启动空调就开了 宝马汽车一启动空调也会开 1

宝马启动空调就开了 宝马一系空调启动关闭 0

宝马启动空调就开了 宝马一系启动 0

宝马启动空调就开了 宝马325空调不启动 0

宝马启动空调就开了 宝马3系空调保险丝 0

宝马启动空调就开了 宝马3系如何播放歌词 0

宝马启动空调就开了 宝马三系什么情况下自动启动空调 0

宝马启动空调 宝马汽车一启动空调也会开 1

宝马启动空调 宝马3系自动启停 0

一个树各个位上的数字加起来和是九倍数那么这个数几也 一个数各个位上的数字加起来和是九倍数那么这个就什么也 1

一个树各个位上的数字加起来和是九倍数那么这个数几也 一个数各个位上的数字加起来和是九倍数那么这个就什么倍 1

一个树各个位上的数字加起来和是九倍数那么这个数几也 11一个数各位上的加起来和是9倍数那么这个(2)也 0

一个数各个位上的数字加起来和是九倍数那么这个就什么也 一个数各个位上的数字加起来和是九倍数那么这个就什么倍 1

一个数各个位上的数字加起来和是九倍数那么这个就什么也 一个数各位上的加起来和是九倍数那么这个 1

一个数各个位上的数字加起来和是九倍数那么这个就什么也 11一个数各位上的加起来和是9倍数那么这个(2)也 0

一个数各个位上的数字加起来和是九倍数那么这个就什么倍 一个数各位上的加起来和是九倍数那么这个 1

一个数各个位上的数字加起来和是九倍数那么这个就什么倍 11一个数各位上的加起来和是9倍数那么这个(2)也 0

训练模型过程中，损失函数L的评估为句子对的预测得分f(x)与真实相似度y之间的差异。通过最小化损失函数来优化模型。

L

模型结构如下所示

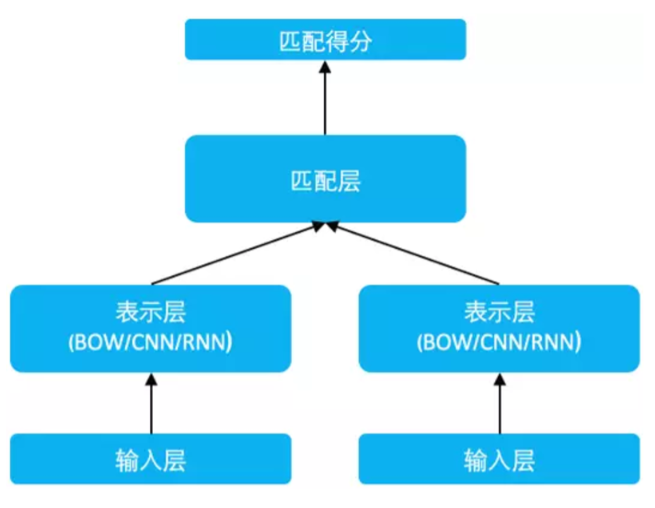


图3.5 point-wise模型

其中输入层通过 look up table 将文本词序列转换为 word embedding 序列。

表示层主要功能是由词到句的表示构建，或者说将序列的孤立的词语的 embedding 表示，转换为具有全局信息的一个或多个低维稠密的语义向量。最简单的是 Bag of Words（BOW）的累加方法，除此之外，也可以使用卷积网络（CNN）、循环神经网络（RNN）等多种表示技术。

匹配层利用文本的表示向量进行交互计算，根据应用的场景不同，一般会使用不同的匹配算法。一种是通过固定的度量函数计算，实际中最常用的就是 cosine 函数，这种方式简单高效，并且得分区间可控意义明确，相似结果的输出区间为[0,1],其中越接近1表示输入的句子在语义上越相似，越接近于0，表示输入的句子在语义上越不相似。

另一种方法就是将两个向量再过一个多层感知器网络（MLP），通过数据训练拟合出一个匹配度得分，这种方式更加灵活拟合能力更强，但对训练的要求也更高。

### 3.5.2pair-wise

Pair-wise方法相较于Pointwise的方法，通过构造偏序关系，让语义相似的句子对的得分明显高于语义不相似的得分来训练语义匹配模型。

其中用于训练pair-wise的数据由三列组成，以制表符（'\t'）分隔，第一列为句子，第二列是与第一列语义相似度句子，第三列是与第一列语义不相似的句子，文件为 utf8 编码。字段信息如表3.2所示：

表3.2 pair-wise训练数据格式字段介绍

|  |  |  |
| --- | --- | --- |
| 字段 | 说明 | 备注 |
| Sentence1 | 句子1 |  |
| Sentence2 | 句子2 | 与sentence1语义相似 |
| Sentence3 | 句子3 | 与sentence1语义不相似 |

部分数据如下：

Sentence1,sentence2,sentence3

用电视机当笔记本电脑显示器好吗 电视机可以当笔记本电脑的显示器吗 笔记本电脑屏幕可以当电视机

用电视机当笔记本电脑显示器好吗 电视机可以当笔记本电脑的显示器吗 笔记本能配显示屏吗

用电视机当笔记本电脑显示器好吗 电视机可以当笔记本电脑的显示器吗 笔记本可以用电视机屏幕

用电视机当笔记本电脑显示器好吗 电视机可以当笔记本电脑的显示器吗 笔记本连了电视机怎么不能用显示器

用电视机当笔记本电脑显示器好吗 电视机可以当笔记本电脑的显示器吗 笔记本电脑屏幕能当电视用吗

用电视机当笔记本电脑显示器好吗 电视机可以当笔记本电脑的显示器吗 笔记本电脑显示屏能当电视吗

笔记本电脑屏幕可以当电视机 笔记本可以用电视机屏幕 笔记本能配显示屏吗

笔记本电脑屏幕可以当电视机 笔记本可以用电视机屏幕 笔记本连了电视机怎么不能用显示器

笔记本电脑屏幕可以当电视机 笔记本可以用电视机屏幕 液晶电视能不接笔记本电脑

笔记本电脑屏幕可以当电视机 笔记本可以用电视机屏幕 如何在用笔记本使用电视机屏幕

笔记本电脑屏幕可以当电视机 笔记本可以用电视机屏幕 笔记本电脑可以用html在电视上显示吗

笔记本电脑屏幕可以当电视机 笔记本可以用电视机屏幕 手提电脑跟电视屏幕不

笔记本电脑屏幕可以当电视机 笔记本电脑屏幕能当电视用吗 笔记本能配显示屏吗

笔记本电脑屏幕可以当电视机 笔记本电脑屏幕能当电视用吗 笔记本连了电视机怎么不能用显示器

笔记本电脑屏幕可以当电视机 笔记本电脑屏幕能当电视用吗 液晶电视能不接笔记本电脑

笔记本电脑屏幕可以当电视机 笔记本电脑屏幕能当电视用吗 如何在用笔记本使用电视机屏幕

笔记本电脑屏幕可以当电视机 笔记本电脑屏幕能当电视用吗 笔记本电脑可以用html在电视上显示吗

训练模型过程中，待比较的句子为S，与S语义相似的句子为S+，与S语义不相似的句子为S-，计算句子语义相似度的匹配函数分别为Q(S,S+）和Q(S,S-)。故损失函数的优化目标如下：

L

模型结构如下所示：

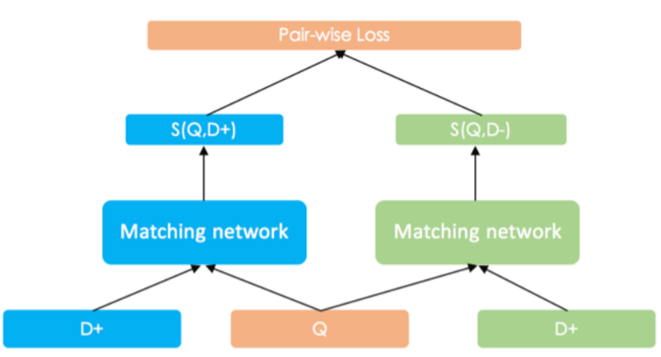


图3.6 pair-wise模型

**代码3.7 训练pointwise网络**

#!/usr/bin/env python

# \_\*\_ coding:utf-8 \_\*\_

import argparse

import logging

import json

import sys

import os

import tensorflow as tf

from tensorflow.python.framework import graph\_util

from utils import datafeeds

from utils import controler

from utils import utility

from utils import converter

\_WORK\_DIR = os.path.split(os.path.realpath(\_\_file\_\_))[0]

sys.path.append(os.path.join(\_WORK\_DIR, '../../../common'))

def load\_config(config\_file):

"""

导入配置文件数据

"""

# 读取配置文件

with open(config\_file, "r") as f:

try:

conf = json.load(f)

except Exception:

logging.error("load json file %s error" % config\_file)

# 保存配置文件数据

conf\_dict = {}

unused = [conf\_dict.update(conf[k]) for k in conf]

logging.debug("\n".join(

["%s=%s" % (u, conf\_dict[u]) for u in conf\_dict]))

return conf\_dict

def train(conf\_dict):

"""

训练网络

"""

training\_mode = conf\_dict["training\_mode"]

net = utility.import\_object(

conf\_dict["net\_py"], conf\_dict["net\_class"])(conf\_dict)

# 采用pointwise模式

if training\_mode == "pointwise":

# 获取pointwise训练数据

datafeed = datafeeds.TFPointwisePaddingData(conf\_dict)

input\_l, input\_r, label\_y = datafeed.ops()

# 输出网络的预测值

pred = net.predict(input\_l, input\_r)

# 将softmax的概率值转换为0，1

output\_prob = tf.nn.softmax(pred, -1, name="output\_prob")

# 设置loss函数

loss\_layer = utility.import\_object(

conf\_dict["loss\_py"], conf\_dict["loss\_class"])()

loss = loss\_layer.ops(pred, label\_y)

# 采用pairwise模式

elif training\_mode == "pairwise":

# 获取pairwise训练数据

datafeed = datafeeds.TFPairwisePaddingData(conf\_dict)

input\_l, input\_r, neg\_input = datafeed.ops()

# 输出正向比较的预测结果

pos\_score = net.predict(input\_l, input\_r)

output\_prob = tf.identity(pos\_score, name="output\_prob")

# 输出负向比较的预测结果

neg\_score = net.predict(input\_l, neg\_input)

# 设置loss函数，使得正向结果大于负向结果

loss\_layer = utility.import\_object(

conf\_dict["loss\_py"], conf\_dict["loss\_class"])(conf\_dict)

loss = loss\_layer.ops(pos\_score, neg\_score)

else:

print(sys.stderr, "training mode not supported")

sys.exit(1)

# 定义优化器

lr = float(conf\_dict["learning\_rate"])

optimizer = tf.train.AdamOptimizer(learning\_rate=lr).minimize(loss)

# 开始训练

controler.run\_trainer(loss, optimizer, conf\_dict)

def predict(conf\_dict):

"""

predict

"""

# 导入网络

net = utility.import\_object(

conf\_dict["net\_py"], conf\_dict["net\_class"])(conf\_dict)

# 导入测试数据

conf\_dict.update({"num\_epochs": "1", "batch\_size": "1",

"shuffle": "0", "train\_file": conf\_dict["test\_file"]})

test\_datafeed = datafeeds.TFPointwisePaddingData(conf\_dict)

test\_l, test\_r, test\_y = test\_datafeed.ops()

# 预测结果

pred = net.predict(test\_l, test\_r)

controler.run\_predict(pred, test\_y, conf\_dict)

def freeze(conf\_dict):

"""

freeze net for c api predict

"""

model\_path = conf\_dict["save\_path"]

freeze\_path = conf\_dict["freeze\_path"]

training\_mode = conf\_dict["training\_mode"]

graph = tf.Graph()

with graph.as\_default():

net = utility.import\_object(

conf\_dict["net\_py"], conf\_dict["net\_class"])(conf\_dict)

test\_l = dict([(u, tf.placeholder(tf.int32, [None, v], name=u))

for (u, v) in dict(conf\_dict["left\_slots"]).iteritems()])

test\_r = dict([(u, tf.placeholder(tf.int32, [None, v], name=u))

for (u, v) in dict(conf\_dict["right\_slots"]).iteritems()])

pred = net.predict(test\_l, test\_r)

if training\_mode == "pointwise":

output\_prob = tf.nn.softmax(pred, -1, name="output\_prob")

elif training\_mode == "pairwise":

output\_prob = tf.identity(pred, name="output\_prob")

restore\_saver = tf.train.Saver()

with tf.Session(graph=graph) as sess:

sess.run(tf.global\_variables\_initializer())

restore\_saver.restore(sess, model\_path)

output\_graph\_def = tf.graph\_util. \

convert\_variables\_to\_constants(sess, sess.graph\_def, ["output\_prob"])

tf.train.write\_graph(output\_graph\_def, '.', freeze\_path, as\_text=False)

def convert(conf\_dict):

"""

convert

"""

converter.run\_convert(conf\_dict)

if \_\_name\_\_ == "\_\_main\_\_":

parser = argparse.ArgumentParser()

parser.add\_argument('--task', default='train',

help='task: train/predict/freeze/convert, the default value is train.')

parser.add\_argument('--task\_conf', default='./examples/cnn-pointwise.json',

help='task\_conf: config file for this task')

args = parser.parse\_args()

task\_conf = args.task\_conf

config = load\_config(task\_conf)

task = args.task

if args.task == 'train':

train(config)

elif args.task == 'predict':

predict(config)

elif args.task == 'freeze':

freeze(config)

elif args.task == 'convert':

convert(config)

else:

print(sys.stderr, 'task type error.')

**代码3.8 设置网络结构**

#!/usr/bin/env python

# \_\*\_ coding:utf-8 \_\*\_

import logging

import layers.tf\_layers as layers

class MLPCnn(object):

"""

设置网络结构

"""

def \_\_init\_\_(self, config):

self.vocab\_size = int(config['vocabulary\_size'])

self.emb\_size = int(config['embedding\_dim'])

self.kernel\_size = int(config['num\_filters'])

self.win\_size = int(config['window\_size'])

self.hidden\_size = int(config['hidden\_size'])

self.left\_name, self.seq\_len = config['left\_slots'][0]

self.right\_name, self.seq\_len = config['right\_slots'][0]

self.task\_mode = config['training\_mode']

# 设置网络嵌入层

self.emb\_layer = layers.EmbeddingLayer(self.vocab\_size, self.emb\_size)

# 设置网络编码层为CNN

self.cnn\_layer = layers.CNNLayer(self.seq\_len, self.emb\_size,

self.win\_size, self.kernel\_size)

# 设置rule层

self.relu\_layer = layers.ReluLayer()

self.concat\_layer = layers.ConcatLayer()

# 设置pointwise网络结构

if self.task\_mode == "pointwise":

self.n\_class = int(config['n\_class'])

# 连接两个全连接层

self.fc1\_layer = layers.FCLayer(2 \* self.kernel\_size, self.hidden\_size)

self.fc2\_layer = layers.FCLayer(self.hidden\_size, self.n\_class)

# 设置pairwise网络结构

elif self.task\_mode == "pairwise":

# 设置一个全连接层与cosina层

self.fc1\_layer = layers.FCLayer(self.kernel\_size, self.hidden\_size)

self.cos\_layer = layers.CosineLayer()

else:

logging.error("training mode not supported")

# 网络的预测函数

def predict(self, left\_slots, right\_slots):

"""

predict graph of this net

"""

left = left\_slots[self.left\_name]

right = right\_slots[self.right\_name]

left\_emb = self.emb\_layer.ops(left)

right\_emb = self.emb\_layer.ops(right)

left\_cnn = self.cnn\_layer.ops(left\_emb)

right\_cnn = self.cnn\_layer.ops(right\_emb)

left\_relu = self.relu\_layer.ops(left\_cnn)

right\_relu = self.relu\_layer.ops(right\_cnn)

if self.task\_mode == "pointwise":

concat = self.concat\_layer.ops([left\_relu, right\_relu], self.kernel\_size \* 2)

concat\_fc = self.fc1\_layer.ops(concat)

concat\_relu = self.relu\_layer.ops(concat\_fc)

pred = self.fc2\_layer.ops(concat\_relu)

else:

hid1\_left = self.fc1\_layer.ops(left\_relu)

hid1\_right = self.fc1\_layer.ops(right\_relu)

left\_relu2 = self.relu\_layer.ops(hid1\_left)

right\_relu2 = self.relu\_layer.ops(hid1\_right)

pred = self.cos\_layer.ops(left\_relu2, right\_relu2)

return pred

**代码3.9 设置损失函数**

#!/usr/bin/env python

# \_\*\_ coding:utf-8 \_\*\_

import tensorflow as tf

class PairwiseHingeLoss(object):

"""

pairwise 采用hinge loss

"""

def \_\_init\_\_(self, config):

"""

init function

"""

self.margin = float(config["margin"])

def ops(self, score\_pos, score\_neg):

"""

operation

"""

return tf.reduce\_mean(tf.maximum(0., score\_neg +

self.margin - score\_pos))

class PairwiseLogLoss(object):

"""

pairwise 采用log loss

"""

def \_\_init\_\_(self, config=None):

"""

init function

"""

pass

def ops(self, score\_pos, score\_neg):

"""

operation

"""

return tf.reduce\_mean(tf.nn.sigmoid(score\_neg - score\_pos))

class SoftmaxWithLoss(object):

"""

softmax loss

"""

def \_\_init\_\_(self):

"""

init function

"""

pass

def ops(self, pred, label):

"""

operation

"""

return tf.reduce\_mean(tf.nn.softmax\_cross\_entropy\_with\_logits(logits=pred,

labels=label))

**代码3.10 数据处理**

#!/usr/bin/env python

# \_\*\_ coding:utf-8 \_\*\_

from collections import Counter

import time

import sys

import os

import tensorflow as tf

from tensorflow.contrib import learn

from utils.utility import get\_all\_files

def load\_batch\_ops(example, batch\_size, shuffle):

"""

load batch ops

"""

if not shuffle:

return tf.train.batch([example],

batch\_size = batch\_size,

num\_threads = 1,

capacity = 10000 + 2 \* batch\_size)

else:

return tf.train.shuffle\_batch([example],

batch\_size = batch\_size,

num\_threads = 1,

capacity = 10000 + 2 \* batch\_size,

min\_after\_dequeue = 10000)

class TFPairwisePaddingData(object):

"""

for pairwise padding data

"""

def \_\_init\_\_(self, config):

self.filelist = get\_all\_files(config["train\_file"])

self.batch\_size = int(config["batch\_size"])

self.epochs = int(config["num\_epochs"])

if int(config["shuffle"]) == 0:

shuffle = False

else:

shuffle = True

self.shuffle = shuffle

self.reader = None

self.file\_queue = None

self.left\_slots = dict(config["left\_slots"])

self.right\_slots = dict(config["right\_slots"])

def ops(self):

"""

produce data

"""

self.file\_queue = tf.train.string\_input\_producer(self.filelist,

num\_epochs=self.epochs)

self.reader = tf.TFRecordReader()

\_, example = self.reader.read(self.file\_queue)

batch\_examples = load\_batch\_ops(example, self.batch\_size, self.shuffle)

features\_types = {}

[features\_types.update({u: tf.FixedLenFeature([v], tf.int64)})

for (u, v) in self.left\_slots.items()]

[features\_types.update({"pos\_" + u: tf.FixedLenFeature([v], tf.int64)})

for (u, v) in self.right\_slots.items()]

[features\_types.update({"neg\_" + u: tf.FixedLenFeature([v], tf.int64)})

for (u, v) in self.right\_slots.items()]

features = tf.parse\_example(batch\_examples, features = features\_types)

return dict([(k, features[k]) for k in self.left\_slots.keys()]),\

dict([(k, features["pos\_" + k]) for k in self.right\_slots.keys()]),\

dict([(k, features["neg\_" + k]) for k in self.right\_slots.keys()])

class TFPointwisePaddingData(object):

"""

for pointwise padding data

"""

def \_\_init\_\_(self, config):

self.filelist = get\_all\_files(config["train\_file"])

self.batch\_size = int(config["batch\_size"])

self.epochs = int(config["num\_epochs"])

if int(config["shuffle"]) == 0:

shuffle = False

else:

shuffle = True

self.shuffle = shuffle

self.reader = None

self.file\_queue = None

self.left\_slots = dict(config["left\_slots"])

self.right\_slots = dict(config["right\_slots"])

def ops(self):

"""

gen data

"""

self.file\_queue = tf.train.string\_input\_producer(self.filelist,

num\_epochs=self.epochs)

self.reader = tf.TFRecordReader()

\_, example = self.reader.read(self.file\_queue)

batch\_examples = load\_batch\_ops(example, self.batch\_size, self.shuffle)

features\_types = {"label": tf.FixedLenFeature([2], tf.int64)}

[features\_types.update({u: tf.FixedLenFeature([v], tf.int64)})

for (u, v) in self.left\_slots.items()]

[features\_types.update({u: tf.FixedLenFeature([v], tf.int64)})

for (u, v) in self.right\_slots.items()]

features = tf.parse\_example(batch\_examples, features = features\_types)

return dict([(k, features[k]) for k in self.left\_slots.keys()]),\

dict([(k, features[k]) for k in self.right\_slots.keys()]),\

features["label"]