

**《模式信息分析》**

**课程论文**

题目 句子相似度分析

自动化学院

学院

学号

姓名

日期

1217053433

周俊

2017/2018学年第二学期

目录

[**一、** **背景** 3](#_Toc516402248)

[二、 **数据** 4](#_Toc516402249)

[三、 **相似度检测方法**： 5](#_Toc516402250)

[**3.1 基准方法** 5](#_Toc516402251)

[3.2 **词移距离** 5](#_Toc516402252)

[3.3 **平滑反向频率** 5](#_Toc516402253)

[3.4 **预训练编码** 5](#_Toc516402254)

[**四、** **实验分析** 7](#_Toc516402255)

[4.1 **基准方法** 7](#_Toc516402256)

[4.2 **词移距离** 7](#_Toc516402257)

[4.3 **平滑反向频率** 8](#_Toc516402258)

[4.4 **预训练编码器** 8](#_Toc516402259)

[五、 **结论** 10](#_Toc516402260)

句子相似度的比较

1. **背景**

在自然语言处理中，词嵌入已经变得很普遍。它们允许我们轻松计算两个单词之间的语义相似度，或查找与目标单词最相似的单词。但是，我们通常更关心两句话或短文本之间的相似性。在这篇文章中，我们将比较计算句子相似度的最流行方式并调查它们的表现。

许多NLP应用需要计算两段短文之间的相似性。例如，搜索引擎需要建模，估计一份文本与提问问题之间的关联度，其中涉及到的并不只是看文字是否有重叠。与之相似的，类似Quora之类的问答网站也有这项需求，他们需要判断某一问题是否之前已出现过。要判断这类的文本相似性，首先要对两个短文本进行embedding，然后计算二者之间的余弦相似度（cosine similarity）。尽管word2vec和GloVe等词嵌入已经成为寻找单词间语义相似度的标准方法，但是对于句子嵌入应如何倍计算仍存在不同的声音。接下来，我们将回顾一下几种最常用的方法，并比较它们之间的性能。

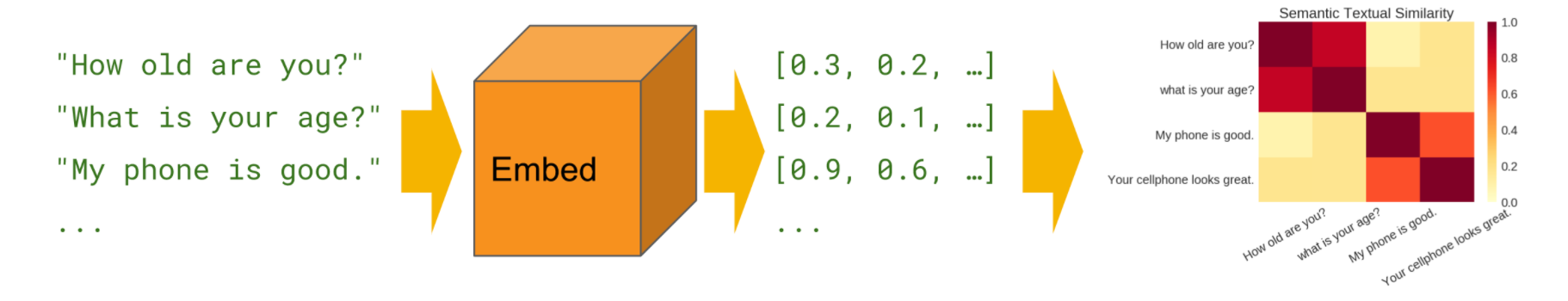


图1典型的句子相似度计算方法

1. **数据**

我们将在两个被广泛使用的数据集上测试所有相似度计算方法，同时还与人类的判断作对比。两个数据集分别是：

* 1. STS基准收集了2012年至2017年国际语义评测SemEval中所有的英语数据
  2. SICK数据库包含了10000对英语句子，其中的标签说明了它们之间的语义关联和逻辑关系

下面的表格是STS数据集中的几个例子。可以看到，两句话之间的语义关系通常非常微小。例如第四个例子：

Table 1：STS数据集中的例子

|  |  |  |  |
| --- | --- | --- | --- |
|  | Sent\_1 | Sent\_2 | sim |
| 0 | A gril is styling her hair. | A gril is brushing her hair. | 2.5 |
| 1 | A group of men play soccer on the beach. | A group of boys are playing soccer on the beach. | 3.6 |
| 2 | One woman is measuring another woman’s ankle. | A woman measure another woman’s ankle | 5.0 |
| 3 | A man is cutting up a cucumber. | A man is slicing a cucumber. | 4.2 |
| 4 | A man is playing a harp. | A man is playing a keyboard. | 1.5 |

通过判断，两句话之间“非常不相似”，尽管它们的句法结构相同，并且其中的词嵌入也类似。

1. **相似度检测方法**：

用于计算两句子间语义相似度的方法非常广泛，下面是常见的几种方法。

**3.1 基准方法**

估计两句子间语义相似度最简单的方法就是求句子中所有单词词嵌入的平均值，然后计算两句子词嵌入之间的余弦相似性。很显然，这种简单的基准方法会带来很多变数。我们将研究，如果忽略终止词并用TF-IDF计算平均权重会带来怎样的影响。

3.2 **词移距离**

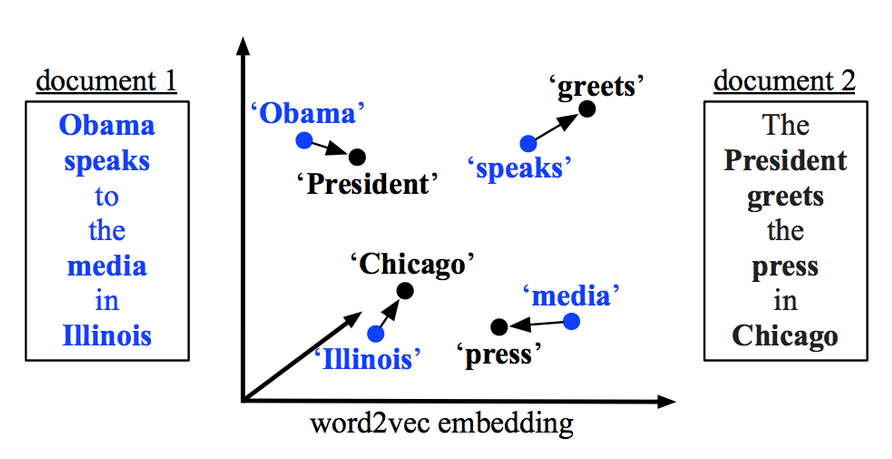


图2 两个文本中的词移距离

两文本之间的词移距离指的是文本一种中所有单词与文本二中的单词之间最小累计距离

替代上述基准方法的其中一种有趣方法就是词移距离（Word Mover’s Distance）。词移距离使用两文本间的词嵌入，测量其中一文本中的单词在语义空间中移动到另一文本单词所需要的最短距离。

3.3 **平滑反向频率**

从语义上来讲，求一句话中词嵌入的平均值似乎给与不相关的单词太多权重了。而Smooth Inverse Frequency试着用两种方法解决这一问题：

* 1. 加权：就像上文用的TF-IDF，SIF取句中词嵌入的平均权重。每个词嵌入都由a/(a + p(w))进行加权，其中a的值经常被设置为0.01，而p(w)是词语在语料中预计出现的频率。
  2. 常见元素删除：接下来，SIF计算了句子的嵌入中最重要的元素。然后它减去这些句子嵌入中的主要成分。这就可以删除与频率和句法有关的变量，他们和语义的联系不大。

最后，SIF使一些不重要的词语的权重下降，例如but、just等，同时保留对语义贡献较大的信息。

3.4 **预训练编码**

上述两种方法都有两个重要的特征。首先，作为简单的词袋方法，它们并不考虑单词的顺序。其次，它们使用的词嵌入是在一种无监督方法中学习到的。这两种特点都有潜在的威胁。由于不同的词语顺序会有不同的意思（例如“the dog bites the man”和“the man bites the dog”），我们想让句子的嵌入对这一变化有所反馈。另外，监督训练可以更直接地帮助句子嵌入学习到句意。

于是就出现了预训练编码器。预训练的句子编码器的目的是充当word2vec和GloVe的作用，但是对于句子嵌入来说：它们生成的嵌入可以用在多种应用中，例如文本分类、近似文本检测等等。一般来说，编码器在许多监督和非监督的任务中训练，目的就是能尽量多地获取通用语义信息。目前已经有好几款这样的编码器了，我们以InferSent和谷歌语句编码器为例。

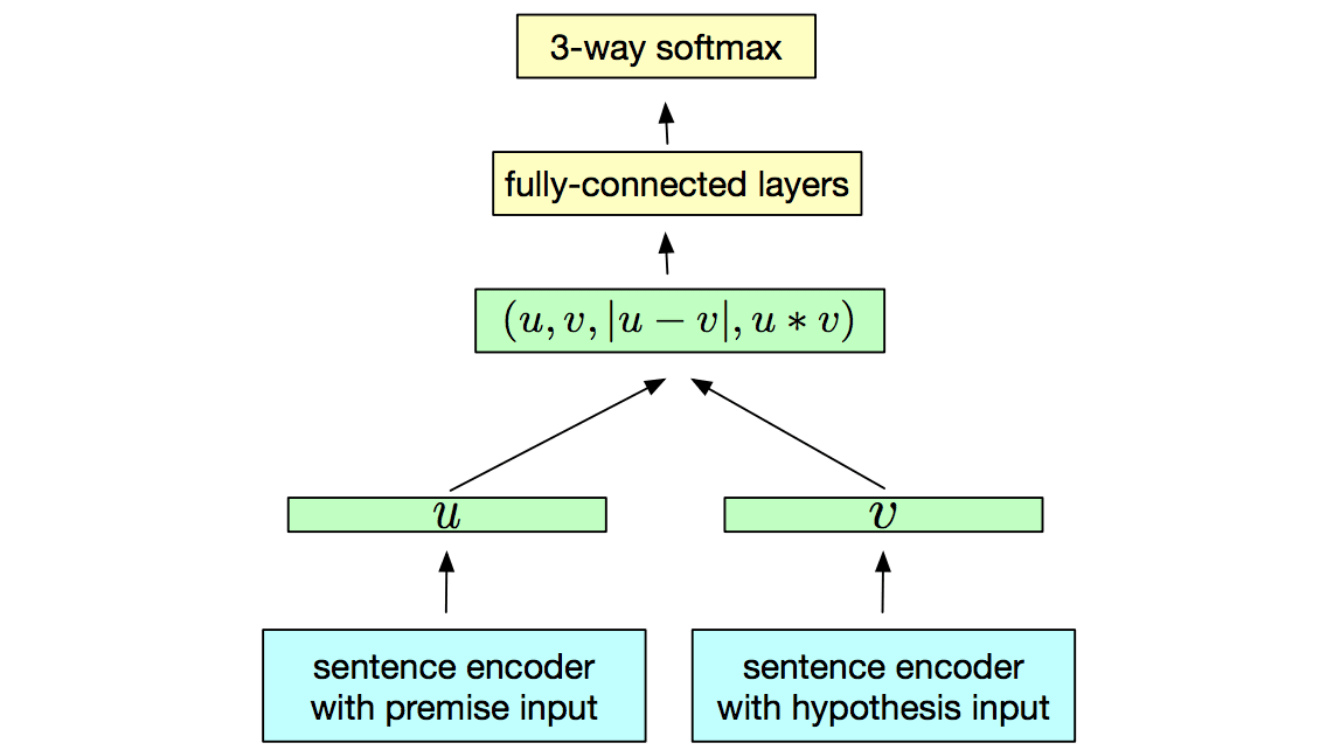


图3 预训练句子编码

预训练句子编码器以来自然语言推理等任务，来学习句子嵌入，以便用于今后的迁移任务中

InferSent是由Facebook研发的预训练编码器，它是一个拥有最大池化的BiLSTM，在SNLI数据集上训练，该数据集含有57万英语句子对，所有句子都属于三个类别的其中一种：推导关系、矛盾关系、中立关系。

为了与Facebook竞争，谷歌也推出了自己的语句编码器，它有两种形式：

* 1. 其中一种高级模型，采用的是变换过的模型编码子图生成的语境感知词所表示的元素总和。
  2. 另一种是简单一些的深度平均网络（DAN），其中输入的单词和双字符的嵌入相加求平均数，并经过一个前馈深层神经网络。

基于变换的模型的结果更好，但是在书写的时候，只有基于DAN的编码器可用。与InferSent不同，谷歌的橘子编码器是在监督数据和非监督数据上共同训练的。

1. **实验分析**

在SICK和STS数据集上测试了上述所有方法，得出句子对之间的相似度，并与人类判断相比较。

4.1 **基准方法**

尽管他们很简洁，在平均词嵌入之间求余弦相似性的基准方法表现得非常好。但是，前提仍要满足一些条件：

* 1. 简单word2vec嵌入比GloVe嵌入表现的好
  2. 在用word2vec时，尚不清楚使用停用词表或TF-IDF加权是否更有帮助。在STS上，有时有用；在SICK上没用。仅计算未加权的所有word2vec嵌入平均值表现得很好。
  3. 在使用GloVe时，停用词列表对于达到好的效果非常重要。利用TF-IDF加权没有帮助。

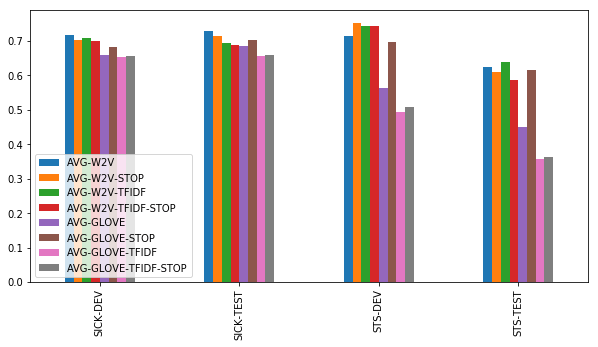


图4 基准方法表现

4.2 **词移距离**

基于我们的结果，好像没有什么使用词移距离的必要了，因为上述方法表现得已经很好了。只有在STS-TEST上，而且只有在有停止词列表的情况下，词移距离才能和简单基准方法一较高下。

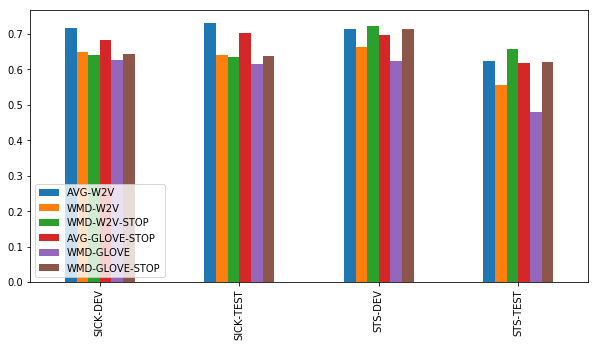


图5 词移距离表现

4.3 **平滑反向频率**

SIF是在测试中表现最稳定的方法。在SICK数据集上，它的表现和简单基准方法差不多，但是在STS数据集上明显超过了它们。注意，在带有word2vec词嵌入的SIF和带有GloVe词嵌入的SIF之间有一些差别，这种差别很显著，它显示了SIF的加权和去除常用元素后减少的非信息噪音。

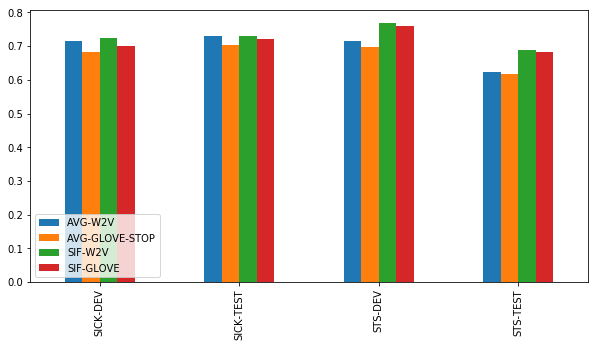


图6 SIF的表现

4.4 **预训练编码器**

预训练编码器的情况比较复杂。但是我们的结果显示编码器还不能完全利用训练的成果。谷歌的句子编码器看起来要比InferSent好一些，但是皮尔森相关系数的结果与SIF的差别不大。

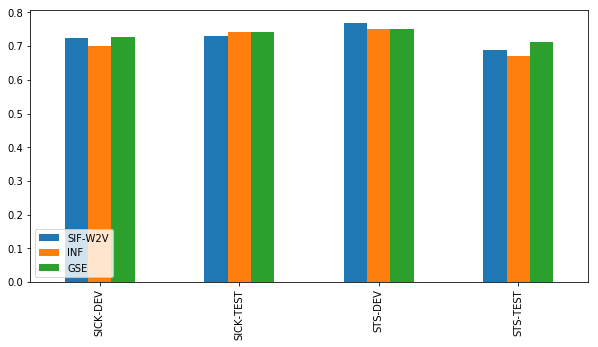


图7 预训练编码器表现

而斯皮尔曼相关系数的效果更直接。这也许表明，谷歌的句子编码器更能了解到句子的正确顺序，但是无法分辨其中的差别。

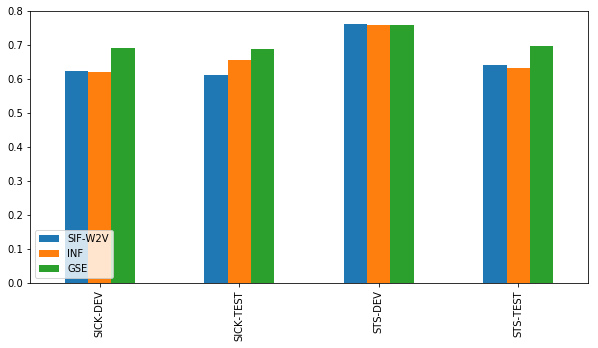


图8 斯皮尔曼相关系数表现

1. **结论**

语句相似度是一个复杂现象，一句话的含义并不仅仅取决于当中的词语，而且还依赖于它们的组合方式。正如开头我们举的那个例子（harp和keyboard），语义相似度有好几种维度，句子可能在其中一种维度上相似，而在其他维度上有差异。目前的句子嵌入方法也只做到了表面。通常我们在皮尔森相关系数（Pearson correlation）上进行测试，除了有些情况下斯皮尔曼相关系数（Spearman correlation）会有不一样的结果。

附录：代码

**Data :**

**STS Benchmark**

import pandas as pd

import numpy as np

import scipy

import math

import os

import tensorflow as tf

import matplotlib.pyplot as plt

import seaborn as sns

def load\_sts\_dataset(filename):

# Loads a subset of the STS dataset into a DataFrame. In particular both

# sentences and their human rated similarity score.

sent\_pairs = []

with tf.gfile.GFile(filename, "r") as f:

for line in f:

ts = line.strip().split("\t")

sent\_pairs.append((ts[5], ts[6], float(ts[4])))

return pd.DataFrame(sent\_pairs, columns=["sent\_1", "sent\_2", "sim"])

def download\_and\_load\_sts\_data():

sts\_dataset = tf.keras.utils.get\_file(

fname="Stsbenchmark.tar.gz",

origin="http://ixa2.si.ehu.es/stswiki/images/4/48/Stsbenchmark.tar.gz",

extract=True)

sts\_dev = load\_sts\_dataset(os.path.join(os.path.dirname(sts\_dataset), "stsbenchmark", "sts-dev.csv"))

sts\_test = load\_sts\_dataset(os.path.join(os.path.dirname(sts\_dataset), "stsbenchmark", "sts-test.csv"))

return sts\_dev, sts\_test

sts\_dev, sts\_test = download\_and\_load\_sts\_data()

**SICK data:**

import requests

def download\_sick(f):

response = requests.get(f).text

lines = response.split("\n")[1:]

lines = [l.split("\t") for l in lines if len(l) > 0]

lines = [l for l in lines if len(l) == 5]

df = pd.DataFrame(lines, columns=["idx", "sent\_1", "sent\_2", "sim", "label"])

df['sim'] = pd.to\_numeric(df['sim'])

return df

sick\_train = download\_sick("https://raw.githubusercontent.com/alvations/stasis/master/SICK-data/SICK\_train.txt")

sick\_dev = download\_sick("https://raw.githubusercontent.com/alvations/stasis/master/SICK-data/SICK\_trial.txt")

sick\_test = download\_sick("https://raw.githubusercontent.com/alvations/stasis/master/SICK-data/SICK\_test\_annotated.txt")

sick\_all = sick\_train.append(sick\_test).append(sick\_dev)

**preparation**

import nltk

STOP = set(nltk.corpus.stopwords.words("english"))

class Sentence:

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

self.raw = sentence

normalized\_sentence = sentence.replace("‘", "'").replace("’", "'")

self.tokens = [t.lower() for t in nltk.word\_tokenize(normalized\_sentence)]

self.tokens\_without\_stop = [t for t in self.tokens if t not in STOP]

import gensim

from gensim.models import Word2Vec

from gensim.scripts.glove2word2vec import glove2word2vec

PATH\_TO\_WORD2VEC = os.path.expanduser("~/data/word2vec/GoogleNews-vectors-negative300.bin")

PATH\_TO\_GLOVE = os.path.expanduser("~/data/glove/glove.840B.300d.txt")

word2vec = gensim.models.KeyedVectors.load\_word2vec\_format(PATH\_TO\_WORD2VEC, binary=True)

tmp\_file = "/tmp/glove.840B.300d.w2v.txt"

glove2word2vec(PATH\_TO\_GLOVE, tmp\_file)

glove = gensim.models.KeyedVectors.load\_word2vec\_format(tmp\_file)

import csv

PATH\_TO\_FREQUENCIES\_FILE = os.path.expanduser("~/data/freqs/frequencies.tsv")

PATH\_TO\_DOC\_FREQUENCIES\_FILE = os.path.expanduser("~/data/freqs/doc\_frequencies.tsv")

def read\_tsv(f):

frequencies = {}

with open(f) as tsv:

tsv\_reader = csv.reader(tsv, delimiter="\t")

for row in tsv\_reader:

frequencies[row[0]] = int(row[1])

return frequencies

frequencies = read\_tsv(PATH\_TO\_FREQUENCIES\_FILE)

doc\_frequencies = read\_tsv(PATH\_TO\_DOC\_FREQUENCIES\_FILE)

doc\_frequencies["NUM\_DOCS"] = 1288431