

# AI 实战训练营第一周

## --判断英文句子语义相似度

### 项目介绍

本次项目提供一系列的英文句子对，每个句子对的两个句子，在语义上具有一定的相似性；每个句子对，获得一个在 0-5 之间的分值来衡量两个句子的语义相似性，打分越高说明两者的语义越相近。

### 数据说明

1.Tal\_SenEvl\_train\_136KB.txt:

训练数据文件，共有 1500 个数据样本，共有 4 个字段：第一个字段为样本编号，第二个字段为一个句子，第三个字段为另一个句子，第四个字段为两个句子的语义相似度打分，如下：

10001      two big brown dogs running through the snow.      A brown dog running through the grass.      2.00000

10002      A woman is peeling a potato.      A woman is slicing a tomato.      1.33300

2.Tal\_SenEvl\_test\_62KB.txt:

测试数据文件，共有 750 个数据样本，共有 3 个字段：第一个字段为样本编号，第二个字段为一个句子，第三个字段为另一个句子，举训练样例说明如下：

10001   two big brown dogs running through the snow.      A brown dog running through the grass.

10002   A woman is peeling a potato.      A woman is slicing a tomato.

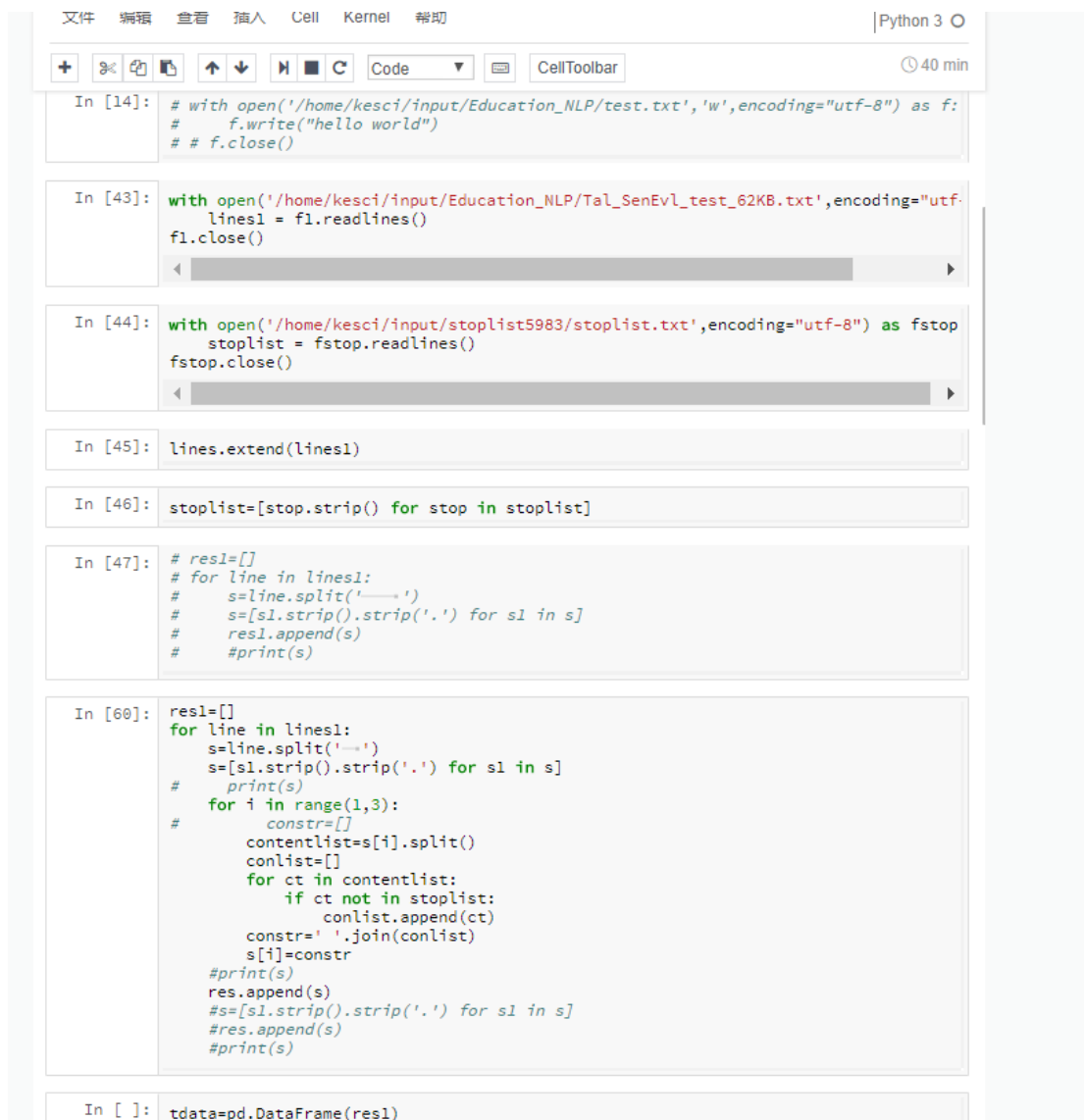
### 解决思路

本次项目实战我打算采取三个阶段进行处理：

- 1、直接套用写过的程序，使用 sklearn 提取 tf-idf 特征进行预测
- 2、采用 gensim 的词袋模型，转化为相应矩阵进行计算
- 3、采用 gensim 的 word2vec，计算出词向量计算

### 采取 tf-idf 特征进行相似度预测

第一步便是进行分词、采用 pandas 中的 DataFrame 进行数据装载



```
In [14]: # with open('/home/kesci/input/Education_NLP/test.txt','w',encoding="utf-8") as f:
#         f.write("hello world")
#         f.close()

In [43]: with open('/home/kesci/input/Education_NLP/Tal_SenEvl_test_62KB.txt',encoding="utf-8") as f1:
lines1 = f1.readlines()
f1.close()

In [44]: with open('/home/kesci/input/stoplist5983/stoplist.txt',encoding="utf-8") as fstop:
stoplist = fstop.readlines()
fstop.close()

In [45]: lines.extend(lines1)

In [46]: stoplist=[stop.strip() for stop in stoplist]

In [47]: # res1=[]
# for line in lines1:
#     s=line.split('→')
#     s=[s1.strip().strip('.') for s1 in s]
#     res1.append(s)
#     #print(s)

In [60]: res1=[]
for line in lines1:
    s=line.split('→')
    s=[s1.strip().strip('.') for s1 in s]
    # print(s)
    for i in range(1,3):
        # constr=[]
        contentlist=s[i].split()
        conlist=[]
        for ct in contentlist:
            if ct not in stoplist:
                conlist.append(ct)
        constr=' '.join(conlist)
        s[i]=constr
    #print(s)
    res.append(s)
    #s=[s1.strip().strip('.') for s1 in s]
    #res.append(s)
    #print(s)

In [ ]: tdata=pd.DataFrame(res1)
```

接下来就是核心步骤

```
from sklearn.linear_model.logistic import LogisticRegression as LR
```

```
clf=LR()
clf.fit(f_tfidf[:1500],df[3][:1500])
tdata[3]=clf.predict(f_tfidf[1500:])
```

利用 **tf-idf** 特征进行处理、预测，获得相似度分析，但这个结果最高为 **0.17**  
后来经过分析，明白了这种提取 **tf-idf** 特征进行相似度分析的，或许更多的是在处理大型文本时才会使用

## 采用词袋模型

第一步的分词步骤与上面大致相同

第二部采用词袋模型时，代码如下：

```
corpus=[dictionary.doc2bow(text) for text in texts]
```

第三步是使用 sklearn 中的第三方库进行处理，这时出现一个**难点**，就是 gensim 时使用的 **sparse 向量如何被 sklearn 识别**。

经过查询，终于找到了办法

#由 corpus 的 sparse 向量转化为密集向量

```
from scipy.sparse import csr_matrix
data = []
rows = []
cols = []
line_count = 0
for line in corpus: # lsi_corpus_total 是之前由 gensim 生成的 lsi 向量
    for elem in line:
        rows.append(line_count)
        cols.append(elem[0])
        data.append(elem[1])
    line_count += 1
sparse_matrix = csr_matrix((data,(rows,cols))) # 稀疏向量
corpus_matrix = sparse_matrix.toarray() # 密集向量
```

再下来的核心步骤：

```
clf=LogisticRegression(C=10)
```

#训练模型

```
clf.fit(traindata[0:1500],skdf.label[0:1500])
```

#输出测试集结果

```
test_yp=clf.predict(traindata[1500:])
```

这种处理方式线下测试结果较好，但由于周六完成，未能进行线上测试

## 采用词向量

预处理同上

模型构建如下：

```
from gensim.models import word2vec
```

```
model = word2vec.Word2Vec(sentences, min_count=1)
```

生成相应 dataframe

```
res=[]
```

```
for sen in sentences:
```

```
    res.append(model[sen].reshape(-1))
```

```
resdf=pd.DataFrame(res)
```

接下便是预测

```
from sklearn.linear_model.logistic import LogisticRegression
```

```
clf=LogisticRegression(C=10)
```

```
#训练模型
```

```
clf.fit(traindf[:1500],res0df[:1500].label)
```

```
#输出测试集结果
```

```
test_yp=clf.predict(traindf[1500:])
```

## 改进之处：

我一直是使用线性模型进行预测分析，明天的话会试着尝试着使用一下 `xgboost`

## 总结感悟

在这一周，真的是收获颇丰，再队友们的激励下，在志愿者的提醒下，在任务的急迫压力之下，自己不断吸收周围的知识，回首这一周真的感谢训练营。

在新的一周，希望自己学到更多的东西，逐渐向优秀的同学靠近