ELMo：基于上下文的语言模型，5分钟构建语义搜索引擎代码实战

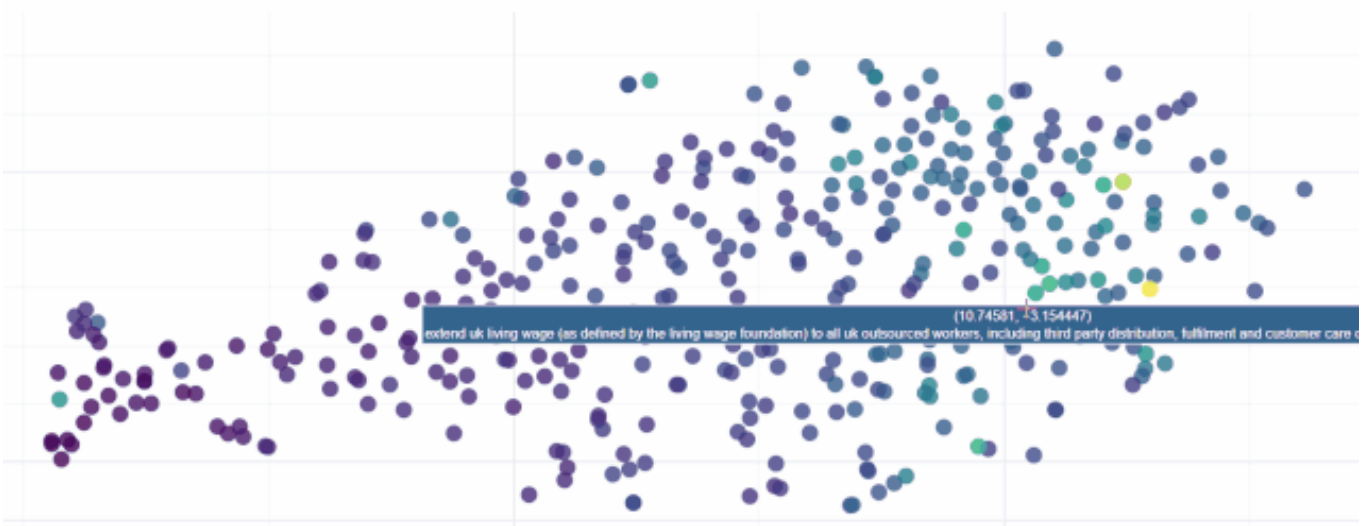
原创 ronghuaiyang AI公园 2019-10-09

作者：Josh Taylor

编译：ronghuaiyang

导读

5分钟内构建一个基于ELMo的语义搜索引擎，在NLP中，上下文就是一切。



使用最先进的ELMo自然语言模型得到的语义句子相似度

本文将探讨自然语言建模的最新进展，深度上下文词嵌入。本文的重点是实践而不是理论，它提供了一个工作示例，说明如何使用最先进的ELMo模型来检查给定文档中的句子相似性，并创建一个简单的语义搜索引擎。完整的代码可以在Colab笔记本中查看：https://colab.research.google.com/drive/13f6dKakC-0yO6_DxqSqo0Kl41KMHT8A1。

上下文在NLP中的重要性

众所周知，语言是复杂的。上下文可以完全改变一个句子中单个单词的意思。例如:

> He kicked the **bucket.**
>
> I have yet to cross-off all the items on my **bucket** list.

> The **bucket** was filled with water.

在这些句子中，虽然“bucket”这个词总是一样的，但它的意思是非常不同的。

根据不同的上下文单词可以有不同的意思

虽然我们可以很容易地理解语言中的这些复杂性，但是创建一个模型来理解周围文本中单词意义的不同细微差别是很困难的。

正是由于这个原因，传统的词嵌入(word2vec、GloVe、fastText)不够完善。它们每个单词只有一个表示，因此它们无法捕获每个单词的含义如何根据周围的上下文变化。

ELMo介绍，深度上下文词表示

ELMo，由AllenNLP于2018年开发，它超越了传统的嵌入技术。它使用深度的、双向的LSTM模型来创建单词表示。

ELMo不是一本单词字典和它们对应的向量，而是在使用它们的上下文中分析单词。它也是基于字符的，允许模型形成词汇表外的单词表示。

因此，这意味着ELMo的使用方式与word2vec或fastText非常不同。ELMo不使用字典“查找”单词及其对应的向量，而是通过深度学习模型传递文本来动态创建向量。

一个工作示例，5分钟内实战ELMo

我们开始！这里我会写出主要的代码，但是如果你想了解全部的代码，请参阅这里：https://colab.research.google.com/drive/13f6dKakC-0yO6_DxqSqo0Kl41KMHT8A1。

根据我最近的几篇文章，我们将使用的数据是基于Modern Slavery returns。这些都是公司的强制性声明，以传达他们如何在内部和供应链内部解决Modern Slavery。在本文中，我们将深入探讨ASOS的return(一家英国在线时尚零售商)。

1. 获取文本数据，清洗并且做成tokens

使用Python字符串函数和spaCy来实现是非常简单的。这里我们做一些基本的文本清理：

a)删除换行符、制表符、多余的空格以及神秘的“xa0”字符;

b)使用spaCy的‘.sents’ 迭代器将文章分成句子。

ELMo可以接收一个句子字符串列表，也可以接收一个列表的列表(句子和单词)。这里我们选择了前者。我们知道ELMo是基于字符的，因此标记单词不应该对性能有任何影响。

```
nlp = spacy.load('en_core_web_md')
#text represents our raw text document
text = text.lower().replace('\n', ' ').replace('\t', ' ').replace('\xa0',' ') #get rid of problem chars
text = ' '.join(text.split()) #a quick way of removing excess whitespace
doc = nlp(text)
sentences = []
for i in doc.sents:
  if len(i)>1:
    sentences.append(i.string.strip()) #tokenize into sentences
```

2. 使用TensorFlow Hub获取ELMo模型：

如果你还没有用过TensorFlow Hub，它提供了大量的预先训练的模型在TensorFlow中使用。幸运的是，ELMo就是其中之一。我们只需两行代码就可以加载一个完全训练好的模型。

```
url = "https://tfhub.dev/google/elmo/2"
embed = hub.Module(url)
```

要使用这个模型，我们只需要多几行代码，将它指向文本文档，并创建句子向量：

```
# This tells the model to run through the 'sentences' list and return the default output (1024 dimension sentence vectors).
embeddings = embed(
    sentences,
    signature="default",
    as_dict=True)["default"]
#Start a session and run ELMo to return the embeddings in variable x
with tf.Session() as sess:
  sess.run(tf.global_variables_initializer())
  sess.run(tf.tables_initializer())
  x = sess.run(embeddings)
```

3. 使用可视化来检查输出

可视化作为一种获得更多数据理解的方式，经常被忽视。图片胜过千言万语，我们将创建一个包含一千个单词的图来证明这一点(实际上是8511个单词)。

在这里，我们使用PCA和t-SNE来把ELMo的输出从1024个维度降到2，以便我们可以查看模型的输出。

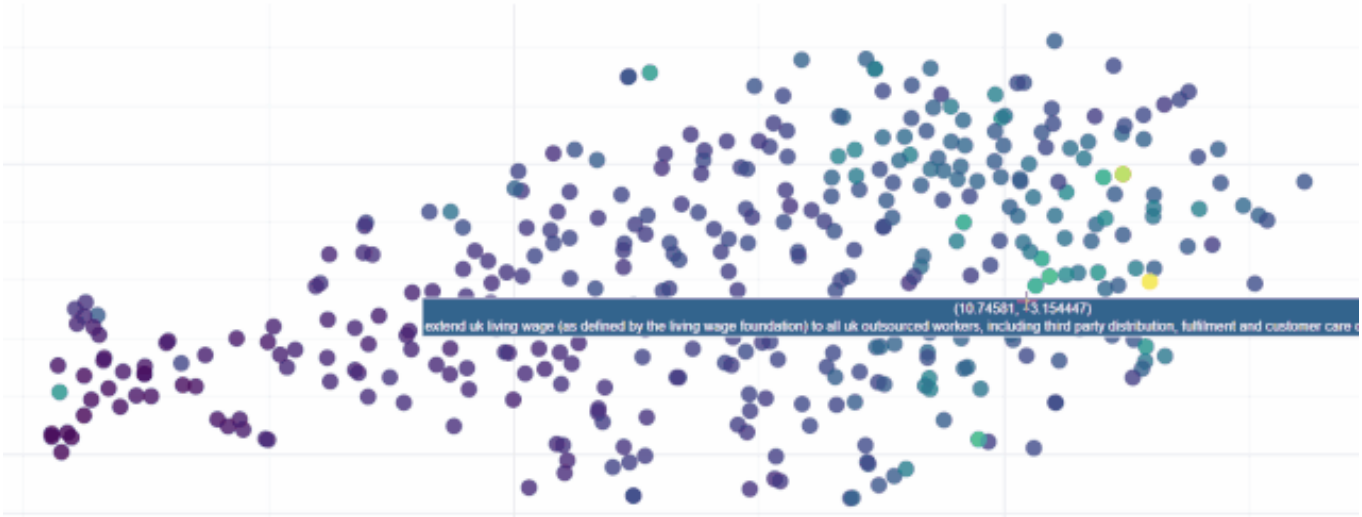
```
from sklearn.decomposition import PCA
pca = PCA(n_components=50) #reduce down to 50 dim
y = pca.fit_transform(x)
from sklearn.manifold import TSNE
y = TSNE(n_components=2).fit_transform(y) # further reduce to 2 dim using t-SNE
```

使用Plotly库，我们可以画出非常漂亮的图。下面的代码展示了如何呈现降维的结果，并将其连接回句子文本。还根据句子的长度添加了颜色。

创建这个的代码如下：

```
import plotly.plotly as py
import plotly.graph_objs as go
from plotly.offline import download_plotlyjs, init_notebook_mode, plot, iplot
init_notebook_mode(connected=True)
data = [
    go.Scatter(
        x=[i[0] for i in y],
        y=[i[1] for i in y],
        mode='markers',
        text=[i for i in sentences],
    marker=dict(
        size=16,
        color = [len(i) for i in sentences], #set color equal to a variable
        opacity= 0.8,
        colorscale='Viridis',
        showscale=False
   )
   )
]
layout = go.Layout()
layout = dict(
            yaxis = dict(zeroline = False),
            xaxis = dict(zeroline = False)
           )
fig = go.Figure(data=data, layout=layout)
file = plot(fig, filename='Sentence encode.html')
from google.colab import files
files.download('Sentence encode.html')
```

在探索这种可视化的过程中，我们可以看到ELMo在根据语义相似性分组句子方面做了出色的工作。事实上，这个模型的效果是相当令人难以置信的：



下载你自己的HTML(上面的链接)，看看ELMo是如何工作的

4. 构造一个语义搜索引擎：

现在我们确信我们的语言模型运行良好，让我们将其应用到语义搜索引擎中。其思想是，我们搜索文本的时候，不是通过关键字，而是通过语义接近我们的搜索查询。

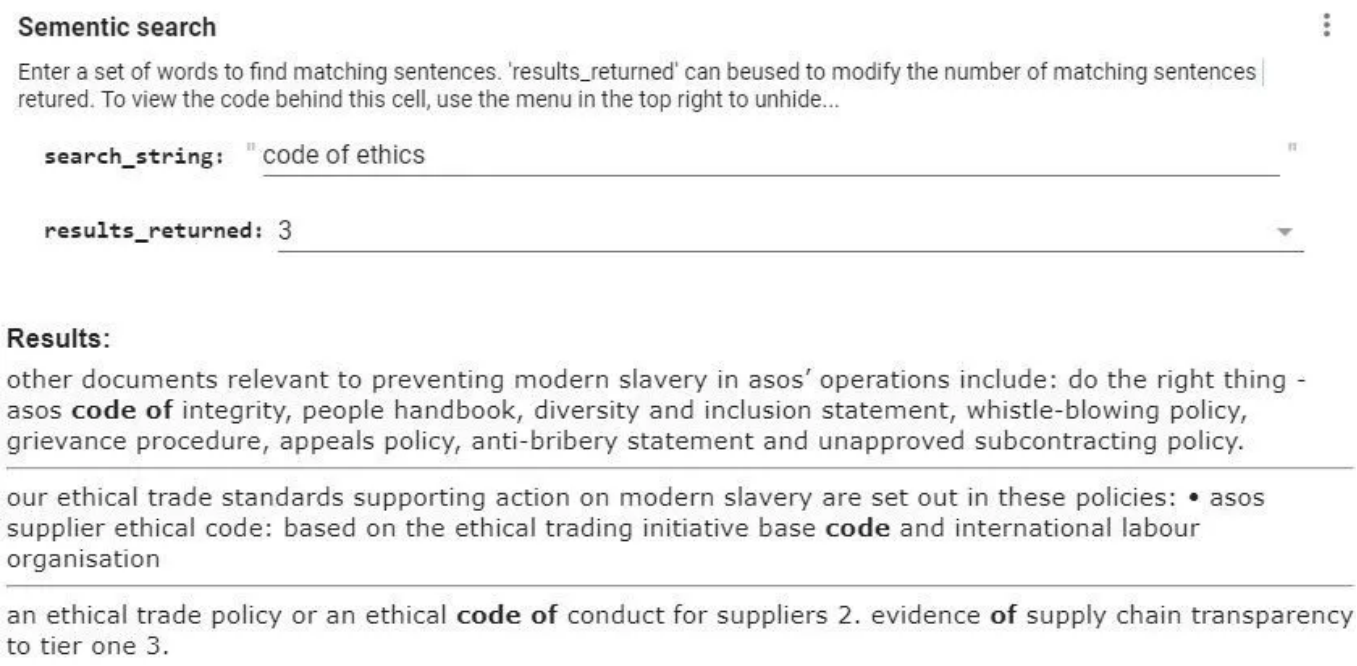
这实际上很容易实现：

- 首先，我们进行一个搜索的query并且在上面运行ELMo；
- 然后在，我们使用余弦相似性来比较这我们的文本文件中的向量；
- 然后从文件中我们可以返回n个最接近的匹配搜索查询。

```
search_string = "example text" #@param {type:"string"}
```

除了使用Colab来输入，我还使用了' IPython.display.HTML '美化输出文本和一些基本的字符串匹配，突出搜索查询和结果之间的常见单词。

让我们来测试一下。让我们看看ASOS在他们的Modern Slavery return中是如何对待道德准则的：



在短短几分钟完成了一个完全互动的语义搜索引擎！

太神奇了！除了关键字之外，谷歌还清楚地知道“ethics”和“ethical”是密切相关的。两者都与我们的搜索查询相关，但不直接基于关键字链接。

— END —