机器学习实验报告

句子分类

摘要：

文本分类是自然语言处理(NLP)的重要的分支。在实际生活中，文本分类有着广泛应用于信息检索，推荐服务，邮件分类等问题。

在本次实验中，我们重点关注文本分类下的分支——句子分类。我们参考了论文[1]的方法实现句子分类：首先用 word2vec 算法将每个词都用一个词向量表示；然后把一句话转成一个二维图像，利用卷积神经网络(CNN)学习图像特征，完成分类目标。在此基础上，我们做了一点改动：在用 word2vec 得到每个词的词向量后，再用多个词的词向量去表示一个词的词向量以得到最终的权矩阵。

遗憾的是，实验结果表明，修改的算法和原算法相比，在 flight_val 数据集上准确率、查全率、查准率和 F1 Score 等指标上略有优势，但是在 movie_val，latop_val 上不具有优势。

1．导言

文本分类问题是令计算机可以根据文本内容，参照一定的标准，将不同的文本划分到预先定义好的类别中的问题。更广泛地说，凡是与文本有关，和分类有关的问题，都可以称之为文本分类问题。文本分类在人类社会中有众多应用场景。例如，通过分析用户的搜索历史和浏览历史，购物网站能对用户的购买意图有所了解，为用户提供合适的推荐[3]。垃圾邮件过滤机制通过分析邮件内容，判断邮件是否为垃圾邮件，将垃圾邮件过滤掉可以有效节省用户的时间，提升用户的效率。此外，句子分类也是实现聊天机器人的核心：聊天机器人需要根据人类的上下文判断人类的真实意图，以做出合适的应答。

对于文本分类问题，人们提出了很多解决办法[5]。最早的词匹配法是根据文本中是否出现与类名相同的词来判断文本是否属于某个类别。这种分类方法过于简单，无法在实际中发挥作用。之后人们尝试利用知识工程的思想来判定文本类别，但是这种方法需要人为介入，设计大量推理规则，效率不高。

人们注意到，人类在获取文本中的信息时，常常是获取文章的语义信息。但是，问题在于此类信息非常抽象，并且存在上下文相关性，很难用计算机可以理解的方式表示出来。为了将文本的“意义”表示成计算机能理解的形式，人们提出假设：认为同一类文档总是存在很多相同的词，而不同类的文档所包含的词之间差异很大。基于此，人们可以用文章中所包含的词汇信息来表示文档，分析文档类别。

为了将词汇信息转为计算机能理解的形式，人们提出多种算法，如词袋模型，TF-IDF，one-hot vector 等，这些算法最终的目标都是将词汇信息转换成数学形式。像 one hot vector 这样的表示模型在实际应用中常常会有“维数灾难”这样的问题，并且 one-hot vector 完全不能捕捉两个词之间的关系。例如“话筒”和“麦克风”会被判断为两个完全不相干的词汇。而像 TF-IDF、词袋模型这样的算法，则是单纯考虑词频，忽略了词的上下文信息。

人们认为，一个词的含义和周围的词有关。一个词和某些词语搭配在一起就是合理的，和另外一些词搭配就是无意义的。Word2vec 就是通过学习这种搭配关系，来将 one-hot vector 转为稠密的、维度更低的向量。

在得到词汇的向量表示之后，可以将统计学习算法(如 SVM，朴素贝叶斯等)应用在文本分类问题上，取得了较好得结果。常见的文本分类算法包括朴素贝叶斯、SVM、决策树、神经网络等机器学习算法。

在此次实验中，我们根据 Yoon Kim 等人的工作[1]，先用 word2vec 的 skip-gram 将词汇信息转换为向量，根据每个词的词向量我们将一句话转成一个二维的矩阵。我们可以将这个二维矩阵视作一个图片，仿照 CNN 对图片分类的思想，我们可以用 CNN 实现句子分类的工作。

2. 实验过程

我们的算法流程图如 Fig.1 所示。

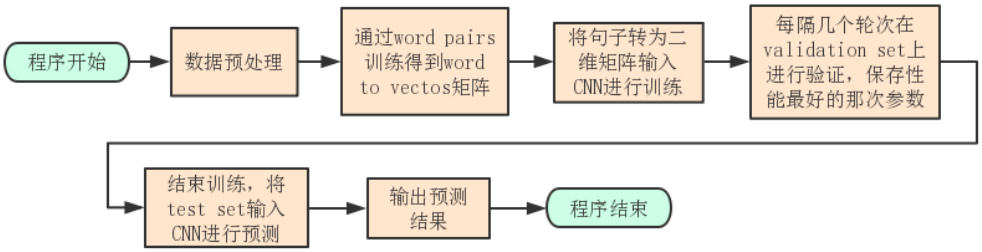

Fig.1 Algorithm flowchart

2.1 数据预处理

在这部分我们读入初始数据，只保留中英文、数字、空格、书名号和制表符。并将处理好的文本内容保存到文件中。

主要代码如下：

```
def read_file(filename):
    rule = re.compile("[^\u4e00-\u9fa5^a-z^A-Z^0-9《》\t ]")  # 只保留中英文，数字，空格和制表符
    contents, labels = [], []
    with open(filename, mode='r', encoding='utf-8') as fr:
        for line in fr:
            line_arr = rule.sub('', line).split()
            labels.append([int(line_arr[0])])
            contents.append(line_arr[1:])
    return contents, labels
def write_contents(filename, contents, split_sign=' '):
    with open(filename, mode='w', encoding='utf-8') as f:
        for words in contents:
            for word in words:
                print(word, end=split_sign, file=f)
```

```
            print(file=f)
```

2.2 Skip-gram

假设词汇表有|V|个词汇，即 one-hot vector 有|V|个维度，目标是将每个词汇用K维的稠密的向量来表示，因此我们需要一个K * |V|的矩阵来实现|V| → K的映射。Skip-gram 使用一个浅层神经网络来达到这个目的。这个浅层神经网络的输入层到隐藏层的权重就是一个K * |V|的矩阵。隐藏层到输出层的权重是一个|V| * K的矩阵，输出层表示词汇表中的各个词出现在输入词周围的概率。

更具体地说，假设有词汇表中有 10000 个词汇，我们希望用 300 维的稠密向量来表示一个词(如 Fig.1 所示)。对输入的词“ants”，用一个 10000 维的 one-hot vector 表示。这个 one-hot 向量与隐藏层矩阵和输出层矩阵相乘输出 10000 的向量。输出的向量经过 softmax 函数和缩放处理后，表示词汇表中的词(如 ability, able 等)出现在“ants”周围的概率。利用实际上出现在“ants”周围的词来计算误差，调整权重，输入层到隐藏层的矩阵为最终所求目标。

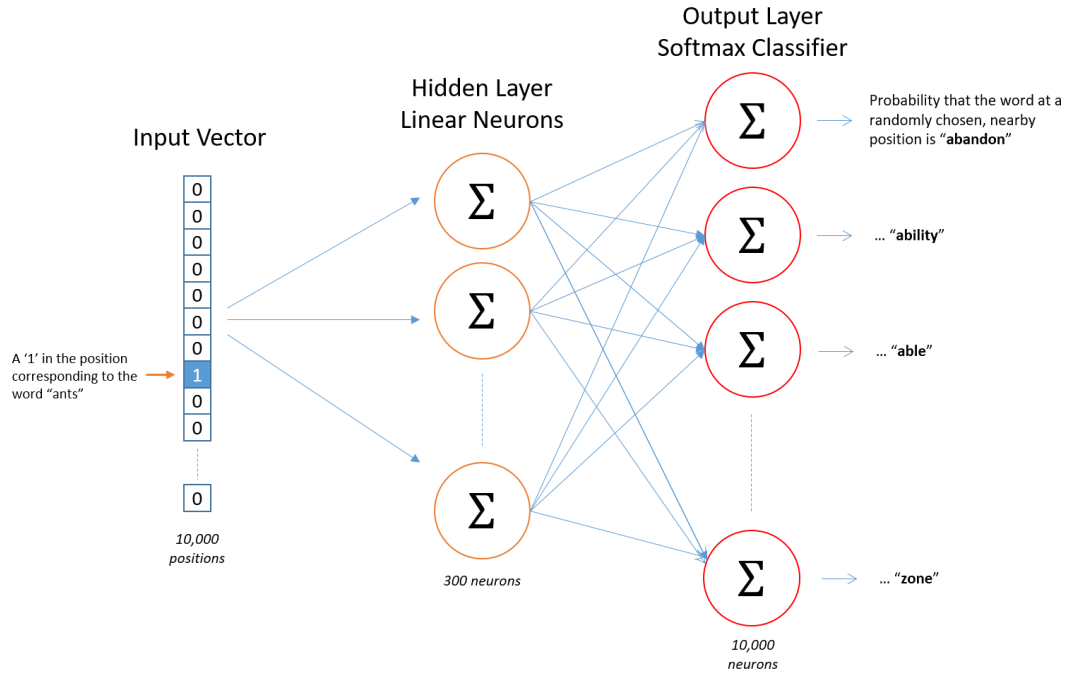

Fig.2 Illustration of Skip-gram(图片来源[2])

对于这部分功能，我们主要使用了 genism 模块来获得每个词的词向量，主要代码如下：

```
from gensim.models import word2vec
def write_word2vec_matrix(contents_file, wv_target_file):
    word2vec_model=word2vec.Word2Vec(
                    word2vec.LineSentence(contents_file),
                    size=cnn.conv1_embedding_dim, window=5,
                    min_count=1)
    word2vec_model.wv.save(wv_target_file)
    del word2vec_model
```

2.3 对 skip-gram 算法进行修改

尽管 word2vec 算法只使用了一个浅层神经网络，但是当隐藏层神经元个数过多时，计算 word2vec 矩阵仍然会带来一定的开销，所以我们要减少隐藏层神经元的数量。但是降低维度会导致词向量不能充分表示词汇所包含的信息的问题。考虑一种极端的情况：将 one-hot vector 映射到一个数字上，本质上这相当于给词汇表中的词编号(赋值)，在某种程度上并不能很好地衡量词与词之间的关联程度。事实上，只要是编号，必然会有大有小。如果我们用向量积衡量两个词之间的关联程度，那么编号大的词会与其他所有词的关联度都很高。我们认为这并不合理。

基于这样的问题，我们提出，减少 word2vec 算法矩阵的行数，即减少浅层神经网络隐藏层的神经元个数。在获得 word2vec 矩阵后，我们不直接用这个矩阵获得某个词的向量表示。对某个词，我们用它的词向量和最接近它的一个或几个词的词向量拼接成的向量来表示。例如，对“水”这个词，它的 word2vec 向量是$[0.3, 0.3]$，与“水”最接近的几个词是“湖”，“海”，对应的 word2vec 向量分别是$[0.1,0.1], [0.2,0.2]$，那么我们用$[0.3, 0.3, 0.1, 0.1, 0.2, 0.2]$来代表“水”这个词。

我们认为两个 word2vec 算法得到的词向量很接近的词，本质上应该蕴含着众多相同或者类似的信息，用两个或多个这样的词向量来表示一个词，一方面加强了这部分相同的信息，另一方面在合理的程度拓宽了词的意义。例如在一个语料中，与“中国”的词向量最接近的词是“基建”。那么用“中国”“基建”的词向量组合表示“中国”，在一定程度上可以认为拓宽了词的联想范围：原本与“中国”的词向量相距较远而与“基建”的词向量相距较近的词，现在与词向量组合有较近的距离，从而与“中国”这个词有了一定的关联度。所以用两个词的词向量来表示一个词有一定的合理性。

主要代码如下。

```
def combine_similar(wv):
    append_wv = {}
    for k in wv.vocab:
        most_similar_tuple =
        (wv.most_similar(positive=[k],topn=1))[0]
        most_similar_key = most_similar_tuple[0]
        append_wv[k] = np.append(wv[k], wv[most_similar_key])
    return append_wv
```

2.4 Convolutional Neural Networks

通过 2.1，2.2 和 2.3，我们得到了将 one-hot vector 转为稠密的、维度较低的$K * |V|$矩阵。通过这个矩阵，可以将一个含有M个词的句子转成一个$M * K$的矩阵。和用 CNN 对图像做分类的思想一样，我们也可以用 CNN 对这个二维矩阵进行分类。值得注意的是，CNN 只能处理固定大小的矩阵。而句子的长度是不确定的，我们需要将这个$M * K$矩阵补零至特定长度才能输入 CNN 进行训练。

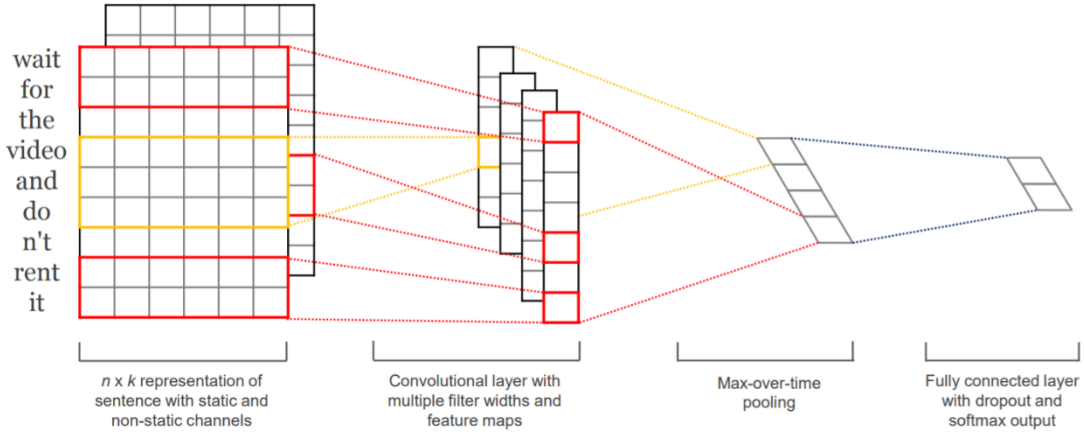

Fig.3 TextCNN(图片来源[1])

这部分我们用设计的 CNN 的形式为：Input Layer➔Convolutional Layer 1➔Pool Layer 1➔Convolutional Layer 2➔Pool Layer 2➔Dense Layer➔Output Layer。

这部分我们主要用 Tensorflow 实现，由于代码较长，因此不在实验报告中展示。

3. 结果分析

3.1 实验环境

硬件：

处理器：Intel® Core™ i507200 CPU @2.50GHz 2.71GHz

内存：8.00GB

软件：Windows10, Python 3.6.8

3.2 算法设计参数

skip-gram 主要参数设置如下

Word Vector Dimension	Window Size	Min Count
[100, 50, 25, 5]	5	1

CNN 的网络结构为：

输入层➔卷积层➔池化层➔卷积层➔池化层➔全连接层➔输出层

主要参数设置如下

Hyperparameters		
Batch Size		[128, 64, 32, 16, 8, 4]
Sequence Length		64
Convolutional Layer #1	Kernel Size	[5, 5]
	Filters	64
Pooling Layer #1	Kernel Size	[2, 2]
	Strides	2
Convolutional Layer #2	Kernel Size	[5, 5]

	Filters	32
Pooling Layer #2	Kernel Size	[2, 2]
	Strides	2
Dense Layer	Units Number	1024
	Dropout Rate	0.4

3.3 实验结果

在这部分，我们主要研究 batch size 和词向量维度对实验结果的影响。相关指标是在验证集上的效果，对每组参数，我们训练 CNN10 次，获得 CNN 在验证集上的预测效果，并取平均值。由于篇幅限制，我们在报告中只展示针对 flight 数据集的实验结果。如下表所示(四舍五入保留四位有效数字)。

令 Word vector dimension = 25

Batch size	Accuracy	Precision	Recall	F1 Score	Time Used (s) CNN
128	0.9315	0.9413	0.9450	0.9430	31.13
64	0.9295	0.9414	0.9443	0.9426	32.14
32	0.9329	0.9479	0.9390	0.9431	32.87
16	0.9293	0.9329	0.9469	0.9393	32.24
8	0.9294	0.9286	0.9570	0.9424	33.26
4	0.9322	0.9467	0.9428	0.9445	33.06

Table 1. Explore the how batch size effects the performance of CNN

综合考虑各个指标，我们认为令 Batch size=32 是比较合适的(准确率、查准率和 F1 Score 较高)。在此基础上，我们分析 word vector dimension 对实验结果的影响。Time Used 是 CNN 训练时间，可以看到，

Word vector dimension	Accuracy	Precision	Recall	F1 Score	Time Used
100	0.9299	0.9306	0.9453	0.9372	179.9
50	0.9428	0.9431	0.9426	0.9427	61.18
25	0.9421	0.9308	0.9555	0.9429	30.13
5	0.9344	0.9459	0.9418	0.9437	7.613

Table 2. Standard skip-gram algorithm

在以上两次实验的基础上，我们进一步讨论我们修改的算法和原来的算法的性能比较。考虑 bath size=32，word2vec 获得的词向量的维度为 25 时的情况。这里考虑了两种时间，Time Used(s) #1 是用 one-hot vector 计算词向量所用的时间，Time Used(s) #2 是用 CNN 学习分类函数的时间。

表格 3 中，100 #1 和 25#1 分别表示在标准 skip-gram 算法下的实验结果，25*4#2 表示在修改后的算法下的结果。Time Used#1 表示训练 skip-gram 算法浅层神经网络所用时间，Time Used#2 表示训练 CNN 学习句子分类所用时间。

Word vector dimension	Accuracy	Precision	Recall	F1 Score	Time Used(s) #1	Time Used(s) #2

100 #1	0.9416	0.9442	0.9391	0.9415	0.8567	171.4
25 #1	0.9415	0.9336	0.9509	0.9421	0.7399	61.67
25*4 #2	0.9490	0.9447	0.9542	0.9492	0.7360	183.3

Table 3. Modified skip-gram algorithm vs standard skip-gram

按照以上步骤，我们可以对每个数据集进行相关测试，找出合适的 batch_size。

在测试 movie 和 laptop 的验证数据集时，我们发现我们修改的算法并没有优势，有时甚至会出现较差的结果。因此，对 flight 和 laptop 数据集，我们只展示修改后的算法的预测结果；对 movie 数据集，我们同时展示了用标准 word2vec+CNN 算法的结果（带“*”号）和用修改后的算法的结果。

预测结果如下：

Test Set	Accuracy	Precision	Recall	F1 Score	Time Used
Flight_test	0.9446	0.9349	0.9563	0.9453	297.7
Laptop_test	0.9081	0.8890	0.9350	0.9103	176.3
Movie_test*	0.7975	0.7712	0.8703	0.8099	110.34
Movie_test	0.7747	0.7239	0.7556	0.8135	120.5

说明：测试 Movie_val 时发现，令 word vector dimension=50 和 100 效果非常接近，此处展示的是 word vector dimension=50 时在 movie_test 数据集上的测试结果。

3.4 算法优缺点分析

分析以上结果，我们可以看到，batch size 对实验结果有一定影响，并且我们修改后的算法确实有一定的效率提升。同时注意到，movie 的测试结果是最差的。我们认为，这是由于电影名本身的含义可以和句子的含义完全不同，造成了 CNN 不能很好地描述句子的“含义”。

还有一点需要指出，我们用修改的算法测试 movie_val 数据集时，准确率等指标很不稳定，好的时候有 90%左右，差的时候只有 60%左右。我们尝试通过修改超参数来找到稳定性能，但是失败了。

用 skip-gram+CNN 的方法来做文本分类，可以在一定程度上学习词与词之间的抽象关系。但是，这种算法的缺点在于，尽管我们可以对文本进行分类，但是无法得到分类的具体逻辑过程，这接近于一个黑箱：一句话中到底是哪些信息导致这句话被分到 A 类而不是 B 类？这个问题是这个算法无法回答的。

同时，我们注意到，用 10 次训练好的网络进行测试，尽管在一定程度上减小了由于偶然事件导致的误差，但是样本量太少无法充分说明我们修改的算法确实有更优秀的性能。因此，如果要做进一步测试，我们应当加大样本量。

此外，我们对超参数的讨论非常有限，只讨论了 batch size 和 word vector dimension 对结果的影响。实际上，skip-gram 的 window size, min count，CNN 的卷积核大小和数量等参数对结果也有重要影响，因此如果要做进一步讨论，必须要考虑更多的超参数。

4．结论

总体分析以上内容，我们可以得出如下结论：

- 在用 CNN 进行训练时，batch size 对训练效果有一定影响。
- 训练效果存在一定偶然性，多次结果取平均值可以降低这种偶然性带来的误

差。

- 在实验过程中，遇到的主要的错误是数据类型不匹配，tensorflow local variable 未初始化。因此，在用 tensorflow 搭建 CNN 时，要非常小心，避免搞错了数据类型。

主要参考文献(三五个即可)

[1] Kim Y. Convolutional neural networks for sentence classification[J]. arXiv preprint arXiv:1408.5882, 2014.

[2] Word2Vec Tutorial - The Skip-Gram Model
http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

[3] Applying word2vec to Recommenders and Advertising
http://mccormickml.com/2018/06/15/applying-word2vec-to-recommenders-and-advertising/

[4] Deep Learning in NLP （一）词向量和语言模型
http://licstar.net/archives/328