人工智能导论第一次作业

数据集划分

```
    for k in label0:
        k_df = source_df[source_df['label'] == k]
        k_df.drop_duplicates(inplace=True)
        query_list = k_df['text'].values
        val_idx = int(0.8 * len(query_list))
        test_idx = int(0.9 * len(query_list))
        train_q, val_q, test_q = query_list[:val_idx], query_list[val_idx:
test_idx], query_list[test_idx:]
        train_df = train_df.append(pd.DataFrame([[q, k] for q in train_q]))
        val_df = val_df.append(pd.DataFrame([[q, k] for q in val_q]))
        test_df = test_df.append(pd.DataFrame([[q, k] for q in test_q]))
```

划分方式如上所示，按照训练集：验证集：测试集=8:1:1的方式划分

实验过程

数据读取和数据预处理

从json文件中读取数据

```
with open('20news.json','r',encoding='utf-8') as f:
        try:
            while True:
                line = f.readline()
                if line:
                    r = json.loads(line)
                    # print(r['text'])
                    data_list.append(r)
                else:
                    break
        except:
            f.close()
```

停用词处理

```
for row in datas['text']:
    after_text=[]

    for i in row:
        if i not in stopword:
            after_text.append(i)
    data_after.append(after_text)
```

数据切分

以0-4、5-9、10-14...如此切分,此处以取0-4切片为例，截取后存入csv文件中

```
data_cut1=datas[datas['label'].isin(['0','1','2','3','4'])]
data_cut1.to_csv("raw0.csv")
```

word2vec词向量化

设置单个词向量的维数为300维，利用word2vec训练模型并保存训练好的模型。这里采用讲所有语料中的词全部事先向量化获得对应的向量表示，方便后续操作。

word2vec模型其实就是简单化的神经网络。输入是One-Hot向量，Hidden Layer没有激活函数。Output Layer维度跟Input Layer的维度一样，用的是Softmax回归。

```
num_featrues = 300
# 设置并行化训练使用CPU计算核心数量
num_workers = 2
# 设置词语上下午窗口大小
context = 5
downsampling = 1e-3
print(sentences)
model = word2vec.Word2Vec(sentences, vector_size=300,workers=num_workers,
window=context)
model.init_sims(replace=True)
# 输入一个路径，保存训练好的模型，其中./data/model目录事先要存在
model.wv.save_word2vec_format("./model/word2vec")
model.save("./model/word2vec_gensim")
```

textCNN训练

基于训练好的word2vec模型进行文本的词向量转化，以及label的匹配

```
vectors=Vectors(name="./model/word2vec")
text_field.build_vocab(train_dataset, val_dataset, min_freq=1,
vectors=vectors)
label_field.build_vocab(train_dataset, val_dataset)
```

data.Iterator 迭代器，用来生成 batch,这里的batch_size设为64

kernel_sizes 卷积核的大小，设为[3,4,5]

embed_dim设为300，kernel_num设为100

```
    train_iter, val_iter = data.Iterator.splits((train_dataset, val_dataset),
                                                batch_sizes=
(config.batch_size, config.batch_size),
                                                sort_key=lambda x:
len(x.text))
    embed_num = len(text_field.vocab)
    class_num = len(label_field.vocab) - 1
    kernel_sizes = [int(k) for k in config.kernel_sizes.split(',')]
    cnn = tm.TextCnn(embed_num, config.embed_dim, class_num,
config.kernel_num, kernel_sizes, config.dropout)
```

textCNN深入架构

优化器使用Adam，它的主要优点有1）结合了Adagrad善于处理稀疏梯度和RMSprop善于处理非平稳目标的优点; 2）对内存需求较小; 3）为不同的参数计算不同的自适应学习率;

损失函数采用交叉熵计算。这里是多分类问题，采用交叉熵损失求导更简单。

```
logit = model(feature)
loss = F.cross_entropy(logit, target)
```

打印每一轮的训练结果如下

```
corrects = (torch.max(logit, 1)[1].view(
                    target.size()).data == target.data).sum()
                accuracy = 100.0 * float(corrects) / batch.batch_size
                sys.stdout.write('\rBatch[{}] - loss: {:.6f}  acc: {:.3f}%
({}/{})'.format(steps,
                  loss.data,
                             accuracy,
                                          corrects,
                                                      batch.batch_size))
```

这里设置为每100轮训练后保存最好的一次训练效果的snapshot,包括每一轮的损失、准确率等

```
if steps % args.test_interval == 0:
                dev_acc = val(dev_iter, model, args)
                if dev_acc > best_acc:
                    best_acc = dev_acc
                    last_step = steps
                    save(model, args.save_dir, 'best', steps)
```

验证

在验证集上计算平均损失，以及正确率

```
logit = model(feature)
loss = F.cross_entropy(logit, target, size_average=False)
avg_loss += loss.data
corrects += (torch.max(logit, 1)[1].view(target.size()).data ==
target.data).sum()
```

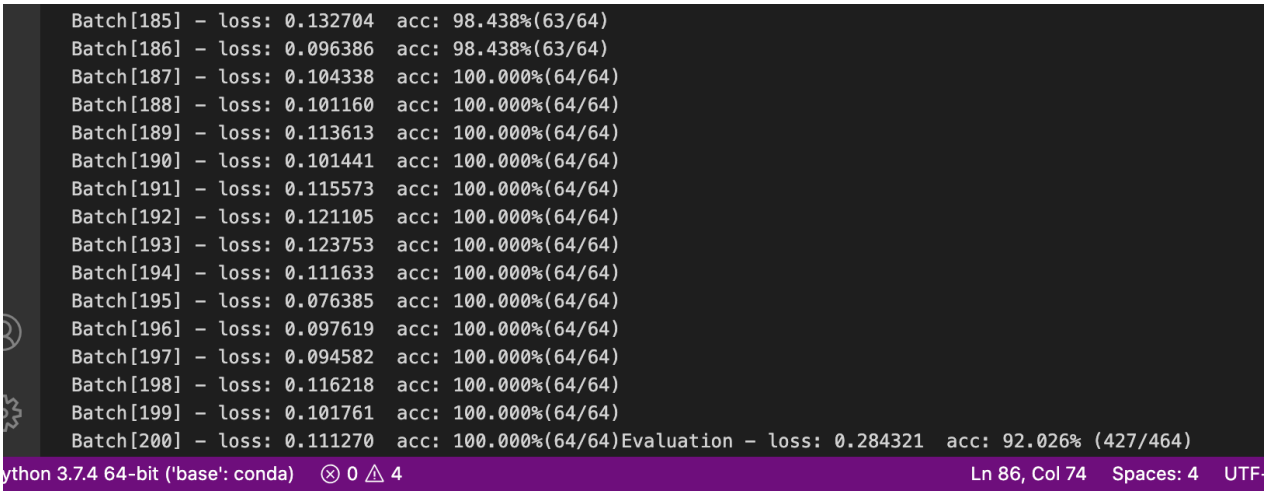
测试

在测试集上对每次是否正确分类进行计数，得到最后的测试准确率

```
k_count = k_df.shape[0]
err_count = 0
query_list = k_df['text'].values
for q in query_list:
    label = tm.predict(q, cnn, text_field, label_field)
    if int(label) != k:
        err_count += 1
print(err_count)
print('acc: %.4f' % ((k_count - err_count) / k_count))
```

实验结果

划分后的第一组训练结果如下



可以看到训练187轮后，准确率保持在100%

测试结果如下