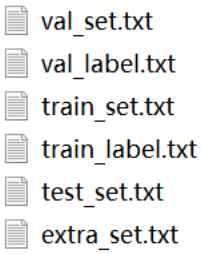
互联网数据挖掘第二次作业

——文本分类

朱政烨 1700017760

数据预处理

- 利用.split("<sep>")将训练集和验证集的文本、标签分开，方便单独处理
- 分词：对于文本，发现有些词中间并没有空格，需要分词，于是采用了 wordninja 库，这个库提供了英文无空格分词的功能。直接调用该库的 split，便能得到分完后的单词的一个列表。
- 大小写转换：所有单词统统转成小写
- 去缩写：将诸如 he’s 的形式改写成 he is 的形式 （这样的情况不多，直接写多个 replace 即可）
- 去标点符号
- 去停用词：nltk 自带的停用词规模太大，去掉后效果不是很好，于是选了一个规模小得多的 Google stopwords（只有四五十个词）
- 词性还原：统统还原成动词
- 存储结果：数据集和标签共 6 个文件，存下来以后随时可以调用



模型搭建

- 首先用传统的机器学习结合 tfidf 简单跑了下，发现效果尚可，其中最好的 LogisticRegression 准确率能达到 86%，但我觉得还不够好。

```
corpus = train_set + val_set + test_set + extra_set
vec = TfidfVectorizer()
tfidf = vec.fit_transform(corpus)
words = vec.get_feature_names()
print ("words: {0}".format(len(words)))

train_set = tfidf[:30000]
val_set = tfidf[30000:37500]
test_set = tfidf[37500:45000]

model = LogisticRegression(solver='sag', multi_class='multinomial')
# model = RandomForestClassifier(n_estimators=10)
# model = AdaBoostClassifier()
# model = MLPClassifier()
# model = SGDClassifier()

model.fit(train_set, train_label)
y_pred = model.predict(val_set)
```

```
Run:  train  code1-sklearn
              precision    recall  f1-score   support

           0       0.93      0.81      0.87      1500
           1       0.83      0.85      0.84      3000
           2       0.85      0.89      0.87      3000

 avg / total       0.86      0.86      0.86      7500
```

- 于是我采用了 Bi-LSTM+attention 的模型，这也是目前准确率比较高的模型。语言采用 Python+Keras。
- 首先，用 Keras 自带的 Tokenizer，将文本序列化，生成文档词典，并将所得的序列填充为统一长度。
- 然后，将原本的数字标签（0、1、2）采取 Onehot 编码为[1,0,0][0,1,0][0,0,1]的形式，更方便神经网络的计算。
- 模型搭建上，采用如图方式：

```
def get_lstm_model():
    model = Sequential()
    model.add(Embedding(len(word_index)+1, embed_size, input_length =maxlen))
    model.add(SpatialDropout1D(0.2))
    model.add(Bidirectional(LSTM(64, return_sequences=True)))
    model.add(Bidirectional(LSTM(64, return_sequences=True)))
    model.add(SeqSelfAttention(attention_type=SeqSelfAttention.ATTENTION_TYPE_MUL,
                               kernel_regularizer=regularizers.l2(1e-3),
                               bias_regularizer=regularizers.l1(1e-3),
                               attention_regularizer_weight=1e-3,
                               name='Attention'))
    model.add(Dropout(0.5))
    model.add(Dense(256, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(256, activation='relu'))
    model.add(Flatten())
    model.add(Dense(3, activation='softmax',kernel_regularizer=regularizers.l2(1e-3),activity_regularizer=regularizers.l1(1e-4)))
    model.summary()
    model.compile(optimizer='Nadam', loss='categorical_crossentropy', metrics=['categorical_accuracy'])

    return model
```

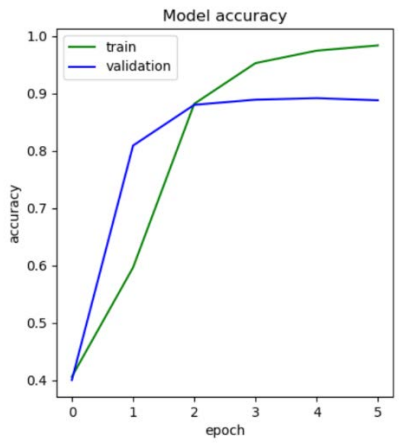
- 首先是 embedding 层，将稀疏的数据降维成密集的矩阵；
- 然后是简单的一个 dropout；
- 接着是两层 Bi-lstm，也就是模型中最重要的部分。原理不必赘述，而且 Keras 已经封装好，我们直接调用就行。这里的 units 设为 64，而且注意到 return_sequences=True，才能连接到下一层；
- LSTM 层之后，我加入了 Self-attention 机制，目前我还不会单独写一个层，还是调用了现有的包 keras_self_attention，并且加入了一些正则化；
- 接着我加了两个全连接层以及 dropout，然后输出，同样加了一些正则化。
- 优化器随便用一个'Nadam'，损失函数使用多分类对应的 'categorical_crossentropy'

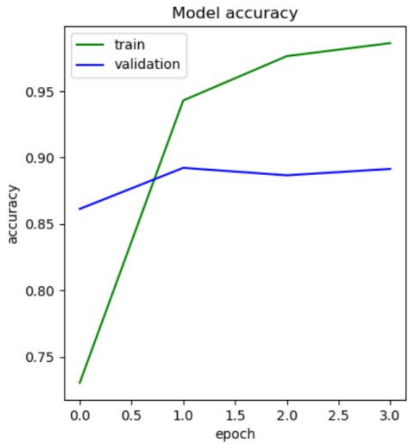
- 模型训练上，直接采用提供的训练集和验证集（没有把 self_test 加入训练集，虽然这样可能得到好结果，但不符合规范），batch-size 设为 500 以达到速度与内存的平衡，epochs 设为 20（事实上发现几个 epoch 便能收敛）。加入了 early-stopping 机制以防过拟合，若 val_loss 连续下降 3 个回合即退出。
- 将训练过程可视化，用 matplotlib 画出训练集和验证集准确度随时间的变化。
- 使用 sklearn 包，在验证集（也就是 self_test.txt ）上计算 micro-{precision，recall，F1} 以及 macro-{precision，recall，F1} 这六项指标。
- 为了更好地利用 extra 数据集，我还采用了标签一致性融合（伪标签）的半监督方法，使用前面的 LR 和 LSTM 模型同时对 extra 进行预测，将得到标签一致的加入训练集，再训练，得到了更好的表现
- 保存模型，并预测测试集的标签，保存到 results 文件中。

- 几点遗憾和思考：
  1. 文本特征化采用的是 embedding，更注重语义方面，而忽视了 tf-idf 所体现的词频信息，如果能结合起来，比如用 tf-idf 给每个 word 的 embedding 向量加权，也许能得到更好的结果
  2. 文本中出现了大量的命名实体，比如人名、组织等，它们在每篇文章中是独一无二的，是否也能合理利用以达到更好的分类
  3. 本来我在 embedding 时嵌入了 word2vec 预训练词向量，但是反而准确率下降，不得不取消了。猜想是没有贴合语料的原因，以后有时间应该可以自训练微调
  4. 词向量模型也可以尝试一下热门的 BERT 模型
  5. 由于未安装 tensorflow-gpu，训练时间较长，两次训练总共跑了四五个小时

结果分析

以下分别是初训练和再训练的准确度变化图，图下第一行是 micro-{precision，recall，F1}，第二行是 macro-{precision，recall，F1}：





0.888 0.888 0.888

0.8935785004118634 0.8901111111111112 0.8916694812835972

0.8914666666666666 0.8914666666666666 0.8914666666666666

0.8958625742177366 0.894 0.8949109297504639

可以发现，初次训练 train_acc 上升得很快，val_acc 也随之上升，3 个 epoch 以后，train_acc 已经超过了 99%，但 val_acc 已经不再上升，趋于稳定，约为 88.8%。之后，经过半监督方法把训练集扩大后再训练，也有相同的趋势，只不过这次 val_acc 更高，约为 89.1%，可见模型有了更好的拟合效果。