什么是情感分析?

情感分析（Sentiment analysis）是自然语言处理（NLP）领域的一个任务，又称倾向性分析，意见抽取（Opinion extraction），意见挖掘（Opinion mining），情感挖掘（Sentiment mining），主观分析（Subjectivity analysis）等，它是对带有情感色彩的主观性文本进行分析、处理、归纳和推理的过程，如从电影评论中分析用户对电影的评价（positive、negative），从商品评论文本中分析用户对商品的“价格、大小、重 量、易用性”等属性的情感倾向。

情感分析的主要方法

现阶段主要的情感分析方法主要有两类：

基于词典的方法：该方法主要通过制定一系列的情感词典和规则，对文本进行段落拆解、句法分析，计算情感值，最后通过情感值来作为文本的情感倾向依据。 基于机器学习的方法：这种方法又分为（1）基于传统机器学习的方法；（2）基于深度学习的方法。

该方法大多将问题转化为一个分类问题来看待，对于情感极性的判断，将目标情感分类2类：正、负，或者根据不同程度分为1-5类。对训练文本进行人工标标注，然后进行有监督的机器学习过程。

本文主要基于深度学习方法对IMDB电影评论进行分析，这其实是一个分类问题，将IMDB电影评论分为正面评价（positive）和负面评价（negative）。

本文将用三种方法循序渐进地讲述使用深度学习对IMDB评论进行情感分析。这三种方法为：MLP、BiRNN（LSTM、GRU）、BiGRU+Attention，IMDB的数据集可以从这里（点击打开链接）下载。使用的深度学习框架是Keras，后端是TensorFlow，在GPU服务器上运行，GPU服务器型号是TITAN X。

基于深度学习方法的IMDB情感分析——数据预处理
IMDB电影评论数据总共有25000条，如果你是在上面的链接中下载的数据，那么数据的组织格式就是下图所示（review是评论文本，sentiment是情感分类标注，1代表positive，0代表negative）：

在读出数据之后，需要对数据进行一些处理，例如过滤掉一些非ASCII字符，清洗掉一些换行符，将大写字母转换为小写等：

In [ ]:

```
def clean_str(string):
    """
    Tokenization/string cleaning for dataset
    Every dataset is lower cased except
    """
    string = re.sub(r"\\", "", string)
    string = re.sub(r"\'", "", string)
    string = re.sub(r"\"", "", string)
    return string.strip().lower()

data_train = pd.read_csv('/data/mpk/IMDB/labeledTrainData.tsv', sep='\t')
print data_train.shape

texts = []
labels = []

for idx in range(data_train.review.shape[0]):
    text = BeautifulSoup(data_train.review[idx], "lxml")
    texts.append(clean_str(text.get_text().encode('ascii','ignore')))


labels = to_categorical(np.asarray(labels))
```

In [ ]:

```
#将数据序列化，并统一长度（这里统一句子长度为1000，多的截断，少的补0）：
tokenizer = Tokenizer(nb_words=MAX_NB_WORDS)
tokenizer.fit_on_texts(texts)
sequences = tokenizer.texts_to_sequences(texts)
word_index = tokenizer.word_index
data = pad_sequences(sequences, maxlen=MAX_SEQUENCE_LENGTH)
```

In [ ]:

```
# 随机打乱数据，并将数据切分为训练集和验证集（切分比例8:2）：
indices = np.arange(data.shape[0])
np.random.shuffle(indices)
data = data[indices]
labels = labels[indices]

nb_validation_samples = int(VALIDATION_SPLIT * data.shape[0])
x_train = data[:-nb_validation_samples]
y_train = labels[:-nb_validation_samples]
x_val = data[-nb_validation_samples:]
y_val = labels[-nb_validation_samples:]
```

In [ ]:

```
# 将数据序列化之后，每一句话就变成了固定长度（1000）的index序列，每一个index对应一个词语。
# 接下来我们将index对应到词语的word Embedding（词向量），这里使用的是glove.6B.100d，即每个词用100维向量表示，
# glove词向量可以在这里（点击打开链接）下载。未登录词（OOV问题）采取的是随机初始化向量，词向量不可训练。
```

In [ ]:

```
GLOVE_DIR = "/data/mpk"
embeddings_index = {}
f = open(os.path.join(GLOVE_DIR, 'glove.6B.100d.txt'))
for line in f:
    values = line.split()
    word = values[0]
    coefs = np.asarray(values[1:], dtype='float32')
    embeddings_index[word] = coefs
f.close()

print('Total %s word vectors.' % len(embeddings_index))
```

In [ ]:

```
embedding_matrix = np.random.random((len(word_index) + 1, EMBEDDING_DIM))
for word, i in word_index.items():
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        # words not found in embedding index will be all-zeros.
        embedding_matrix[i] = embedding_vector
print ('Length of embedding_matrix:', embedding_matrix.shape[0])
embedding_layer = Embedding(len(word_index) + 1,
                            EMBEDDING_DIM,
                            weights=[embedding_matrix],
                            mask_zero=False,
                            input_length=MAX_SEQUENCE_LENGTH,
                            trainable=False)
```

In [ ]:

```
# 基于深度学习方法的IMDB情感分析—MLP
# 基于多层感知器（MLP）对IMDB进行分类是非常简单的一种神经网络应用，关于MLP的原理及Keras实现参见我的这篇文章（点击打开链接）。

# 在得到文本向量表示之后，可以直接将向量输入MLP网络，经过多层MLP训练之后，进行softmax分类。代码如下：
```

In [ ]:

```
sequence_input = Input(shape=(MAX_SEQUENCE_LENGTH,), dtype='int32')
embedded_sequences = embedding_layer(sequence_input)
dense_1 = Dense(100,activation='tanh')(embedded_sequences)
max_pooling = GlobalMaxPooling1D()(dense_1)
dense_2 = Dense(2, activation='softmax')(max_pooling)

model = Model(sequence_input, dense_2)

model.compile(loss='categorical_crossentropy',
              optimizer='rmsprop',
              metrics=['acc'])

model.summary()
model.fit(x_train, y_train, validation_data=(x_val, y_val),
          nb_epoch=10, batch_size=50)
```

In [ ]:

```
# 基于深度学习方法的IMDB情感分析—RNN（LSTM、GRU）
# 由于RNN在文本处理方面的优势，所以这里采用RNN的变种LSTM和GRU分别进行IMDB情感分析，同时为了克服RNN的方向性，采用双向RNN，即BiLSTM和BiGRU，Keras里的包装器Bidirectional可以将LSTM和GRU包装成双向的，十分方便。此外，在BiLSTM和BiGRU上面添加了2层Dense层。

#  （1）BiLSTM

# 代码如下：
```

In [ ]:

```
sequence_input = Input(shape=(MAX_SEQUENCE_LENGTH,), dtype='int32')
embedded_sequences = embedding_layer(sequence_input)
l_gru = Bidirectional(LSTM(100, return_sequences=False))(embedded_sequences)
dense_1 = Dense(100,activation='tanh')(l_gru)
dense_2 = Dense(2, activation='softmax')(dense_1)

model = Model(sequence_input, dense_2)

model.compile(loss='categorical_crossentropy',
              optimizer='rmsprop',
              metrics=['acc'])

model.summary()
model.fit(x_train, y_train, validation_data=(x_val, y_val),
          nb_epoch=10, batch_size=50)
```

In [ ]:

```
#biGRU
sequence_input = Input(shape=(MAX_SEQUENCE_LENGTH,), dtype='int32')
embedded_sequences = embedding_layer(sequence_input)
l_gru = Bidirectional(GRU(100, return_sequences=False))(embedded_sequences)
dense_1 = Dense(100,activation='tanh')(l_gru)
dense_2 = Dense(2, activation='softmax')(dense_1)

model = Model(sequence_input, dense_2)

model.compile(loss='categorical_crossentropy',
              optimizer='rmsprop',
              metrics=['acc'])

model.summary()
model.fit(x_train, y_train, validation_data=(x_val, y_val),
          nb_epoch=10, batch_size=50)
```

In [ ]:

```
# 可以看到，在第6次迭代达到最高的准确率88.84%，比MLP提升了2个百分点。

# 对比BiLSTM和BiGRU的结果，可以发现BiGRU比BiLSTM要稍微好一些，这跟大家的经验一致，当数据少时，GRU的效果要比LSTM好一些。所以接下来使用BiGRU结合Attention。



# 基于深度学习方法的IMDB情感分析—BiGRU+Attention
# Attention模型最早提出是用在图像识别上的，模仿人类的注意力机制，给图像不同的局部赋予不同的权重。在自然语言中使用最早是在机器翻译领域，这里我们在BiLSTM的基础上添加一个Attention Model，即对BiLSTM的隐层每一个时间步的向量学习一个权重，也就是在得到句子的向量表示时对评论文本中不同的词赋予不同的权值，然后由这些不同权值的词向量加权得到句子的向量表示。

# 这里的Attention层采用的是论文FEED-FORWARD NETWORKS WITH ATTENTION CANSOLVE SOME LONG-TERM MEMORY PROBLEMS（点击打开链接）中的强制前向Attention模型，如下图所示：



# 具体实现代码如下：
```

In [ ]:

```
from keras import backend as K
from keras.engine.topology import Layer
from keras import initializations, regularizers, constraints

class Attention_layer(Layer):
    """
        Attention operation, with a context/query vector, for temporal data.
        Supports Masking.
        Follows the work of Yang et al. [https://www.cs.cmu.edu/~diyiy/docs/naacl16.pdf]
        "Hierarchical Attention Networks for Document Classification"
        by using a context vector to assist the attention
        # Input shape
            3D tensor with shape: `(samples, steps, features)`.
        # Output shape
            2D tensor with shape: `(samples, features)`.
        :param kwargs:
        Just put it on top of an RNN Layer (GRU/LSTM/SimpleRNN) with return_sequences=True.
        The dimensions are inferred based on the output shape of the RNN.
        Example:
            model.add(LSTM(64, return_sequences=True))
            model.add(AttentionWithContext())
        """

    def __init__(self,
                 W_regularizer=None, b_regularizer=None,
                 W_constraint=None, b_constraint=None,
                 bias=True, **kwargs):

        self.supports_masking = True
        self.init = initializations.get('glorot_uniform')

        self.W_regularizer = regularizers.get(W_regularizer)
        self.b_regularizer = regularizers.get(b_regularizer)

        self.W_constraint = constraints.get(W_constraint)
        self.b_constraint = constraints.get(b_constraint)

        self.bias = bias
        super(Attention_layer, self).__init__(**kwargs)

    def build(self, input_shape):
        assert len(input_shape) == 3

        self.W = self.add_weight((input_shape[-1], input_shape[-1],),
                                 initializer=self.init,
                                 name='{}_W'.format(self.name),
                                 regularizer=self.W_regularizer,
                                 constraint=self.W_constraint)
        if self.bias:
            self.b = self.add_weight((input_shape[-1],),
                                     initializer='zero',
                                     name='{}_b'.format(self.name),
                                     regularizer=self.b_regularizer,
                                     constraint=self.b_constraint)

        super(Attention_layer, self).build(input_shape)
```

```
    def compute_mask(self, input, input_mask=None):
        # do not pass the mask to the next layers
        return None

    def call(self, x, mask=None):
        uit = K.dot(x, self.W)

        if self.bias:
            uit += self.b

        uit = K.tanh(uit)

        a = K.exp(uit)

        # apply mask after the exp. will be re-normalized next
        if mask is not None:
            # Cast the mask to floatX to avoid float64 upcasting in theano
            a *= K.cast(mask, K.floatx())

        # in some cases especially in the early stages of training the sum may be almos
t zero
        # and this results in NaN's. A workaround is to add a very small positive numbe
r to the sum.
        # a /= K.cast(K.sum(a, axis=1, keepdims=True), K.floatx())
        a /= K.cast(K.sum(a, axis=1, keepdims=True) + K.epsilon(), K.floatx())
        weighted_input = x * a
        return K.sum(weighted_input, axis=1)

    def get_output_shape_for(self, input_shape):
        return input_shape[0], input_shape[-1]
```

In [ ]:

```
#BiGRRU+Attention代码如下:
sequence_input = Input(shape=(MAX_SEQUENCE_LENGTH,), dtype='int32')
embedded_sequences = embedding_layer(sequence_input)
l_gru = Bidirectional(LSTM(100, return_sequences=True))(embedded_sequences)
l_att = Attention_layer()(l_gru)
dense_1 = Dense(100,activation='tanh')(l_att)
dense_2 = Dense(2, activation='softmax')(dense_1)

model = Model(sequence_input, dense_2)

model.compile(loss='categorical_crossentropy',
              optimizer='rmsprop',
              metrics=['acc'])

model.summary()
model.fit(x_train, y_train, validation_data=(x_val, y_val),
          nb_epoch=10, batch_size=50)
```

In [ ]:

```
#可以看到，在第4次迭代达到90.54%的准确率，比BiGRU提高了差不多2个百分点。证明加Attention层确实有效。
```