# 基于深度学习的 IMDB 数据情感分析

## （终期报告）

小组成员

司恩泽 2120171050　　安骄阳 2120170998

李安民 2120171027　　邢博文 2120171082　　董永银 2120171007

## 内容与意义

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。

电影评论，简称影评，是对一部电影的导演、演员、镜头语言、拍摄技术、剧情、线索、环境、色彩、光线等进行分析和批评，又称电影批评。电影评论的目的在于分析、鉴定和评价蕴含在银幕中的审美价值、

认识价值、社会意义、镜头语言等方面，达到拍摄影片的目的，解释影片中所体现出的道理，既能通过分析影片的成败得失，帮助导演开阔视野，提高创作水平，以促进电影艺术的繁荣和发展；还能通过分析和评价，影响观众对影片的理解和鉴赏，提高观众的欣赏水平，从而间接促进电影艺术的发展。

## 分析方法

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：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、LSTM、BiLSTM+Attention。

# 目标与功能

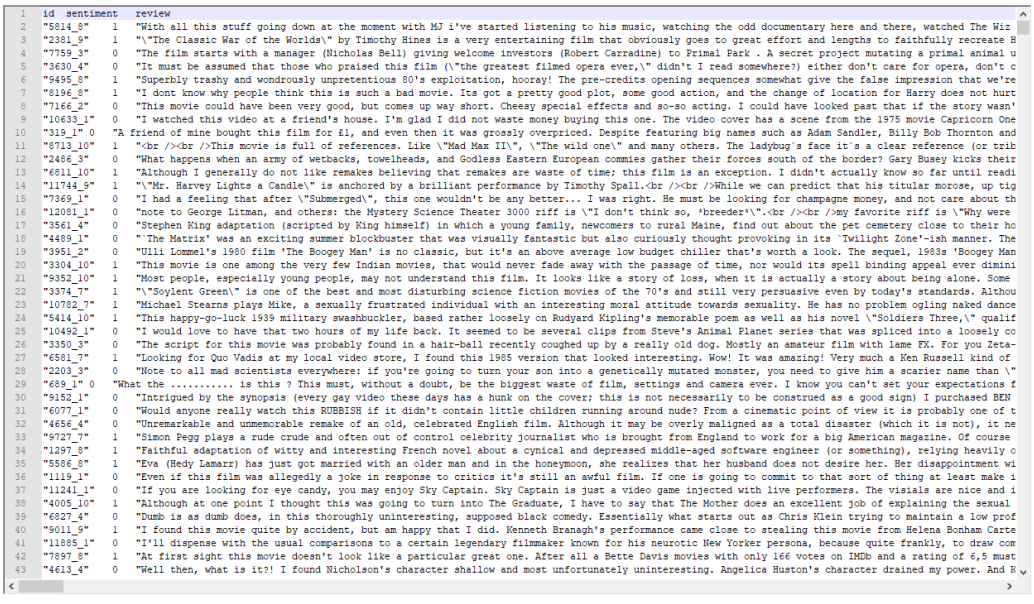
本文主要基于深度学习方法对 IMDB 电影评论进行分析，这其实是一个分类问题，将 IMDB 电影评论分为正面评价（positive）和负面评价（negative）。

训练出的结果作为评论电影好坏的模型。

# 过程与结果

## 1、 数据的准备

该电影评论是来自 IMDb 中的电影评论，数据集一共包含了 25000 条关于电影的正面评论和负面评论，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）：



| id | sentiment | review |
|---|---|---|
| "5814_8" | 1 | "With all this stuff going down at the moment with MJ i've started listening to his music, watching the odd documentary here and there, watched The Wiz |
| "2381_9" | 1 | "\"The Classic War of the Worlds\" by Timothy Hines is a very entertaining film that obviously goes to great effort and lengths to faithfully recreate H |
| "7759_3" | 0 | "The film starts with a manager (Nicholas Bell) giving welcome investors (Robert Carradine) to Primal Park . A secret project mutating a primal animal u |
| "3630_4" | 0 | "It must be assumed that those who praised this film (\"the greatest filmed opera ever,\" didn't I read somewhere?) either don't care for opera, don't c |
| "9495_8" | 1 | "Superbly trashy and wondrously unpretentious 80's exploitation, hooray! The pre-credits opening sequences somewhat give the false impression that we're |
| "8196_8" | 1 | "I dont know why people think this is such a bad movie. Its got a pretty good plot, some good action, and the change of location for Harry does not hurt |
| "7166_2" | 0 | "This movie could have been very good, but comes up way short. Cheesy special effects and so-so acting. I could have looked past that if the story wasn' |
| "10633_1" | 0 | "I watched this video at a friend's house. I'm glad I did not waste money buying this one. The video cover has a scene from the 1975 movie Capricorn One |
| "319_1" | 0 | "A friend of mine bought this film for £1, and even then it was grossly overpriced. Despite featuring big names such as Adam Sandler, Billy Bob Thornton and |
| "8713_10" | 1 | "<br /><br />This movie is full of references. Like \"Mad Max II\", \"The wild one\" and many others. The ladybug´s face it´s a clear reference (or trib |
| "2486_3" | 0 | "What happens when an army of wetbacks, towelheads, and Godless Eastern European commies gather their forces south of the border? Gary Busey kicks their |
| "6811_10" | 1 | "Although I generally do not like remakes believing that remakes are waste of time; this film is an exception. I didn't actually know so far until readi |
| "11744_9" | 1 | "\"Mr. Harvey Lights a Candle\" is anchored by a brilliant performance by Timothy Spall.<br /><br />While we can predict that his titular morose, up tig |
| "7369_1" | 0 | "I had a feeling that after \"Submerged\", this one wouldn't be any better.. I was right. He must be looking for champagne money, and not care about th |
| "12081_1" | 0 | "note to George Litman, and others: the Mystery Science Theater 3000 riff is \"I don't think so, 'breeder*\".<br /><br />my favorite riff is \"Why were |
| "3561_4" | 0 | "Stephen King adaptation (scripted by King himself) in which a young family, newcomers to rural Maine, find out about the pet cemetery close to their ho |
| "4489_1" | 0 | "`The Matrix' was an exciting summer blockbuster that was visually fantastic but also curiously thought provoking in its `Twilight Zone'-ish manner. The |
| "3951_2" | 0 | "Ulli Lommel's 1980 film 'The Boogey Man' is no classic, but it's an above average low budget chiller that's worth a look. The sequel, 1983s 'Boogey Man |
| "3304_10" | 1 | "This movie is one among the very few Indian movies, that would never fade away with the passage of time, nor would its spell binding appeal ever dimini |
| "9352_10" | 1 | "Most people, especially young people, may not understand this film. It looks like a story of loss, when it is actually a story about being alone. Some |
| "3374_7" | 1 | "\"Soylent Green\" is one of the best and most disturbing science fiction movies of the 70's and still very persuasive even by today's standards. Althou |
| "10782_7" | 1 | "Michael Stearns plays Mike, a sexually frustrated individual with an interesting moral attitude towards sexuality. He has no problem ogling naked dance |
| "5414_10" | 1 | "This happy-go-luck 1939 military swashbuckler, based rather loosely on Rudyard Kipling's memorable poem as well as his novel \"Soldiers Three,\" qualif |
| "10492_1" | 0 | "I would love to have that two hours of my life back. It seemed to be several clips from Steve's Animal Planet series that was spliced into a loosely co |
| "3350_3" | 0 | "The script for this movie was probably found in a hair-ball recently coughed up by a really old dog. Mostly an amateur film with lame FX. For you Zeta- |
| "6581_7" | 1 | "Looking for Quo Vadis at my local video store, I found this 1985 version that looked interesting. Wow! It was amazing! Very much a Ken Russell kind of |
| "2203_3" | 0 | "Note to all mad scientists everywhere: if you're going to turn your son into a genetically mutated monster, you need to give him a scarier name than \" |
| "689_1" | 0 | "What the ..... is this ? This must, without a doubt, be the biggest waste of film, settings and camera ever. I know you can't set your expectations f |
| "9152_1" | 0 | "Intrigued by the synopsis (every gay video these days has a hunk on the cover; this is not necessarily to be construed as a good sign) I purchased BEN |
| "6077_1" | 0 | "Would anyone really watch this RUBBISH if it didn't contain little children running around nude? From a cinematic point of view it is probably one of t |
| "4656_4" | 0 | "Unremarkable and unmemorable remake of an old, celebrated English film. Although it may be overly maligned as a total disaster (which it is not), it ne |
| "5727_7" | 1 | "Simon Pegg plays a rude crude and often out of control celebrity journalist who is brought from England to work for a big American magazine. Of course |
| "1297_8" | 1 | "Faithful adaptation of witty and interesting French novel about a cynical and depressed middle-aged software engineer (or something), relying heavily o |
| "5586_8" | 1 | "Eva (Hedy Lamarr) has just got married with an older man and in the honeymoon, she realizes that her husband does not desire her. Her disappointment wi |
| "1119_1" | 0 | "Even if this film was allegedly a joke in response to critics it's still an awful film. If one is going to commit to that sort of thing at least make i |
| "11241_1" | 0 | "If you are looking for eye candy, you may enjoy Sky Captain. Sky Captain is just a video game injected with live performers. The visials are nice and i |
| "4005_10" | 1 | "Although at one point I thought this was going to turn into The Graduate, I have to say that The Mother does an excellent job of explaining the sexual |
| "6827_4" | 0 | "Dumb is as dumb does, in this thoroughly uninteresting, supposed black comedy. Essentially what starts out as Chris Klein trying to maintain a low prof |
| "9011_9" | 1 | "I found this movie quite by accident, but am happy that I did. Kenneth Branagh's performance came close to stealing this movie from Helena Bonham Carte |
| "11885_1" | 0 | "I'll dispense with the usual comparisons to a certain legendary filmmaker known for his neurotic New Yorker persona, because quite frankly, to draw com |
| "7897_8" | 1 | "At first sight this movie doesn't look like a particular great one. After all a Bette Davis movies with only 166 votes on IMDb and a rating of 6,5 must |
| "4613_4" | 0 | "Well then, what is it?! I found Nicholson's character shallow and most unfortunately uninteresting. Angelica Huston's character drained my power. And R |

## 2、 数据预处理

（1）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。

```
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()

PATH = 'E:/dataset/IMDB_movie_reviews/labeledTrainData.tsv'
data_train = pd.read_csv(PATH, 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.append(data_train.sentiment[idx])
```

（2）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）。

```
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:]
```

（3）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_PATH = 'E:/model/nlp/glove.6B.100d.txt'
embeddings_index = {}
f = open(GLOVE_PATH, encoding='utf-8')
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))
```

```
the -0.038194 -0.24487 0.72812 -0.39961 0.083172 0.043953 -0.39141 0.3344 -0.57545 0.087459 0.28787 -0.06731 0.30906 -0.26384 -0.
, -0.10767 0.11053 0.59812 -0.54361 0.67396 0.10663 0.038867 0.35481 0.06351 -0.094189 0.15786 -0.81665 0.14172 0.21939 0.58505 -
. -0.33979 0.20941 0.46348 -0.64792 -0.38377 0.038034 0.17127 0.15978 0.46619 -0.019169 0.41479 -0.34349 0.26872 0.04464 0.42131
of -0.1529 -0.24279 0.89837 0.16996 0.53516 0.48784 -0.58826 -0.17982 -1.3581 0.42541 0.15377 0.24215 0.13474 0.41193 0.67043 -0.
to -0.1897 0.050024 0.19084 -0.049184 -0.089737 0.21006 -0.54952 0.098377 -0.20135 0.34241 -0.092677 0.161 -0.13268 -0.2816 0.187
and -0.071953 0.23127 0.023731 -0.50638 0.33923 0.1959 -0.32943 0.18364 -0.18057 0.28963 0.20448 -0.5496 0.27399 0.58327 0.20468
in 0.085703 -0.22201 0.16569 0.13373 0.38239 0.35401 0.01287 0.22461 -0.43817 0.50164 -0.35874 -0.34983 0.055156 0.69648 -0.17958
a -0.27086 0.044006 -0.02026 -0.17395 0.6444 0.71213 0.3551 0.47138 -0.29637 0.54427 -0.72294 -0.0047612 0.040611 0.043236 0.2972
" -0.30457 -0.23645 0.17576 -0.72854 -0.28343 -0.2564 0.26587 0.025309 -0.074775 -0.3766 -0.057774 0.12159 0.34384 0.41928 -0.232
's 0.58854 -0.2025 0.73479 -0.68338 -0.19675 -0.1802 -0.39177 0.34172 -0.60561 0.63816 -0.26695 0.36486 -0.40379 -0.1134 -0.58718
for -0.14401 0.32554 0.14257 -0.099227 0.72536 0.19321 -0.24188 0.20223 -0.89599 0.15215 0.035963 -0.59513 -0.051635 -0.014428 0.
- -1.2557 0.61036 0.56793 -0.96596 -0.45249 -0.071696 0.57122 -0.31292 -0.43814 0.90622 0.06961 -0.053104 0.25029 0.27841 0.77724
that -0.093337 0.19043 0.68457 -0.41548 -0.22777 -0.11803 -0.095434 0.19613 0.17785 -0.020244 -0.055409 0.33867 0.79396 -0.047126
on -0.21863 -0.42664 0.5196 0.0043103 0.58045 -0.10873 -0.37726 0.4566 -0.60627 -0.075773 0.11306 0.17703 0.1605 0.074514 0.63649
is -0.54264 0.41476 1.0322 -0.40244 0.46691 0.21816 -0.074864 0.47332 0.080996 -0.22079 -0.12808 -0.1144 0.50891 0.11568 0.028211
was 0.13717 -0.54287 0.19419 -0.29953 0.17545 0.084672 0.67752 0.098295 -0.035611 0.21334 0.51663 0.20687 0.44082 -0.33655 0.5602
said -0.13128 -0.452 0.043399 -0.99798 -0.21053 -0.95868 -0.24609 0.48413 0.18178 0.475 -0.22305 0.30064 0.43496 -0.3605 0.20245
with -0.43608 0.39104 0.51657 -0.13861 0.2029 0.50723 -0.012544 0.22948 -0.6316 0.21199 -0.018043 -0.39364 0.74164 0.30221 0.5179
he 0.1225 -0.058833 0.23658 -0.28877 -0.028181 0.31524 0.070229 0.16447 -0.027623 0.25214 0.21174 -0.059674 0.36133 0.13607 0.187
as -0.32721 0.096446 0.34244 -0.44327 0.30535 -0.042016 -0.071235 -0.31036 -0.22557 -0.181 -0.29088 -0.61542 0.29751 0.030491 0.4
it -0.30664 0.16821 0.98511 -0.33606 -0.2416 0.16186 -0.053496 0.4301 0.57342 -0.071569 0.36101 0.26729 0.27789 -0.072268 0.13838
by -0.20875 -0.1174 0.26478 -0.28339 0.19584 0.7446 -0.03887 0.028499 -0.44252 -0.30426 0.27133 -0.51907 0.52183 -0.76648 0.28043
at 0.1766 0.093851 0.24351 0.44313 -0.39037 0.12524 -0.19918 0.59855 -0.82035 0.28006 0.54231 0.023079 0.12837 -0.044489 0.3837 -
( 0.19247 0.36617 0.52301 -0.79857 -0.2592 0.18267 0.19564 0.83148 -0.67636 -0.84648 1.4429 -0.84978 -0.023986 1.328 0.74061 0.03
) -0.13797 0.27084 0.84036 -0.45668 -0.49429 0.35777 0.077772 0.42481 0.0076481 -0.50942 1.4008 -0.79993 0.053011 1.2054 0.3783 0
from 0.30731 0.24737 0.68231 -0.52367 0.44053 0.42044 0.0002514 0.15265 -0.61363 0.22631 0.083071 0.070425 0.017683 0.56807 1.006
```

## 3、 MLP 方法

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。

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:

```
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, epochs=10, batch_size=50, validation_data=(x_val, y_val))
```

我们可以看到结果如下图所示:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   Connected to
=================================================================
input_1 (InputLayer)         (None, 1000)              0
_________________________________________________________________
embedding_1 (Embedding)      (None, 1000, 100)         8054700   input_1[0][0]
_________________________________________________________________
dense_1 (Dense)              (None, 1000, 100)         10100     embedding_1[0][0]
_________________________________________________________________
globalmaxpooling1d_1 (GlobalMaxP (None, 100)           0         dense_1[0][0]
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 202       globalmaxpooling1d_1[0][0]
=================================================================
Total params: 8,065,002
Trainable params: 10,302
Non-trainable params: 8,054,700
```

```
Train on 20000 samples, validate on 5000 samples
Epoch 1/10
20000/20000 [==============================] - 3s - loss: 0.5431 - acc: 0.7467 - val_los
s: 0.4496 - val_acc: 0.8108
Epoch 2/10
20000/20000 [==============================] - 2s - loss: 0.3993 - acc: 0.8329 - val_los
s: 0.3752 - val_acc: 0.8436
Epoch 3/10
20000/20000 [==============================] - 2s - loss: 0.3511 - acc: 0.8541 - val_los
s: 0.3527 - val_acc: 0.8552
Epoch 4/10
20000/20000 [==============================] - 2s - loss: 0.3183 - acc: 0.8707 - val_los
s: 0.3393 - val_acc: 0.8628
Epoch 5/10
20000/20000 [==============================] - 2s - loss: 0.2958 - acc: 0.8801 - val_los
s: 0.3325 - val_acc: 0.8616
Epoch 6/10
20000/20000 [==============================] - 2s - loss: 0.2765 - acc: 0.8901 - val_los
s: 0.3256 - val_acc: 0.8654
Epoch 7/10
20000/20000 [==============================] - 2s - loss: 0.2612 - acc: 0.8973 - val_los
s: 0.3358 - val_acc: 0.8628
Epoch 8/10
20000/20000 [==============================] - 2s - loss: 0.2466 - acc: 0.9034 - val_los
s: 0.3195 - val_acc: 0.8680
Epoch 9/10
20000/20000 [==============================] - 2s - loss: 0.2330 - acc: 0.9110 - val_los
s: 0.3260 - val_acc: 0.8648
Epoch 10/10
20000/20000 [==============================] - 2s - loss: 0.2220 - acc: 0.9161 - val_los
s: 0.3192 - val_acc: 0.8650
```

可以看到，在第 8 次迭代，达到最高的准确率 86.80%（看来最简单的 3 层 MLP 效果还不错）

## 4、 LSTM 方法

LSTM 算法全称为 Long short-term memory，最早由 Sepp

Hochreiter 和 Jürgen Schmidhuber 于 1997 年提出，是一种特定形式的 RNN（Recurrent neural network，循环神经网络），而 RNN 是一系列能够处理序列数据的神经网络的总称。

```
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, epochs=10, batch_size=50, validation_data=(x_val, y_val))
```

我们可以看到结果如下图所示：

```
Layer (type)                 Output Shape              Param #     Connected to
==================================================================================
input_1 (InputLayer)         (None, 1000)              0
_________________________________________________________________
embedding_1 (Embedding)      (None, 1000, 100)         8054700     input_1[0][0]
_________________________________________________________________
bidirectional_1 (Bidirectional) (None, 1000, 200)      160800      embedding_1[0][0]
_________________________________________________________________
attention_layer_1 (Attention_lay (None, 200)           40200       bidirectional_1[0][0]
_________________________________________________________________
dense_1 (Dense)              (None, 100)               20100       attention_layer_1[0][0]
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 202         dense_1[0][0]
==================================================================================
Total params: 8,276,002
Trainable params: 221,302
Non-trainable params: 8,054,700
```

```
Train on 20000 samples, validate on 5000 samples
Epoch 1/10
20000/20000 [==============================] - 1225s - loss: 0.6145 - acc: 0.6607 - val_
loss: 0.4628 - val_acc: 0.7974
Epoch 2/10
20000/20000 [==============================] - 1225s - loss: 0.4354 - acc: 0.8004 - val_
loss: 0.3667 - val_acc: 0.8418
Epoch 3/10
20000/20000 [==============================] - 1228s - loss: 0.3561 - acc: 0.8446 - val_
loss: 0.3283 - val_acc: 0.8566
Epoch 4/10
20000/20000 [==============================] - 1227s - loss: 0.3161 - acc: 0.8683 - val_
loss: 0.3147 - val_acc: 0.8652
Epoch 5/10
20000/20000 [==============================] - 1230s - loss: 0.2863 - acc: 0.8816 - val_
loss: 0.3059 - val_acc: 0.8760
Epoch 6/10
20000/20000 [==============================] - 1234s - loss: 0.2603 - acc: 0.8952 - val_
loss: 0.2988 - val_acc: 0.8756
Epoch 7/10
20000/20000 [==============================] - 1230s - loss: 0.2377 - acc: 0.9042 - val_
loss: 0.2947 - val_acc: 0.8782
Epoch 8/10
20000/20000 [==============================] - 1224s - loss: 0.2143 - acc: 0.9142 - val_
loss: 0.3108 - val_acc: 0.8736
Epoch 9/10
20000/20000 [==============================] - 1231s - loss: 0.1895 - acc: 0.9255 - val_
loss: 0.3183 - val_acc: 0.8748
Epoch 10/10
20000/20000 [==============================] - 1227s - loss: 0.1631 - acc: 0.9367 - val_
loss: 0.3362 - val_acc: 0.8726
```

可以看到，在第 7 次迭代达到 87.82%的准确率，比 MLP 的效果好。

## 5、 BiLSTM+Attention 方法

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。

BiLSTM+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, epochs=10, batch_size=50, validation_data=(x_val, y_val))
```

我们可以看到结果如下图所示:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_1 (InputLayer)         (None, 1000)              0         
_________________________________________________________________
embedding_1 (Embedding)      (None, 1000, 100)         8054700   
_________________________________________________________________
bidirectional_1 (Bidirectional) (None, 1000, 200)      160800    
_________________________________________________________________
attention_1 (Attention)      (None, 200)               40400     
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 402       
=================================================================
Total params: 8,256,302
Trainable params: 8,256,302
Non-trainable params: 0
```

```
Train on 20000 samples, validate on 5000 samples
Epoch 1/10
20000/20000 [==============================] - 1190s - loss: 0.4394 - acc: 0.7988 - val_
loss: 0.3183 - val_acc: 0.8714
Epoch 2/10
20000/20000 [==============================] - 1191s - loss: 0.2919 - acc: 0.8807 - val_
loss: 0.2717 - val_acc: 0.8928
Epoch 3/10
20000/20000 [==============================] - 1182s - loss: 0.2234 - acc: 0.9109 - val_
loss: 0.2462 - val_acc: 0.8984
Epoch 4/10
20000/20000 [==============================] - 1111s - loss: 0.1714 - acc: 0.9359 - val_
loss: 0.2430 - val_acc: 0.9054
Epoch 5/10
20000/20000 [==============================] - 1098s - loss: 0.1304 - acc: 0.9538 - val_
loss: 0.2568 - val_acc: 0.9018
Epoch 6/10
20000/20000 [==============================] - 1101s - loss: 0.0942 - acc: 0.9665 - val_
loss: 0.2876 - val_acc: 0.9030
Epoch 7/10
20000/20000 [==============================] - 1101s - loss: 0.0618 - acc: 0.9801 - val_
loss: 0.3566 - val_acc: 0.8990
Epoch 8/10
20000/20000 [==============================] - 1104s - loss: 0.0441 - acc: 0.9868 - val_
loss: 0.3851 - val_acc: 0.8960
Epoch 9/10
20000/20000 [==============================] - 1099s - loss: 0.0298 - acc: 0.9905 - val_
loss: 0.4063 - val_acc: 0.8972
Epoch 10/10
20000/20000 [==============================] - 1107s - loss: 0.0208 - acc: 0.9934 - val_
loss: 0.5198 - val_acc: 0.8834
```

可以看到，在第 4 次迭代达到 90.54%的准确率，证明加 Attention 层确实有效。

## 分析方法

随着信息时代的快速发展，数据的增量迅速扩大，从数据中挖掘有价值的内容成为了一个重要的研究问题。对于电影影评，充分挖掘用户对于电影的直接情感，，既能通过分析影片的成败得失，帮助导演开阔视野，提高创作水平，以促进电影艺术的繁荣和发展；还能通过分析和评价，影响观众对影片的理解和鉴赏，提高观众的欣赏水平，从而间接促进电影艺术的发展。

本文通过三种方法对电影评论进行了数据挖掘，结果显示不同的方法达到的准确率不同且使用的迭代数不同，综合使用一些方法能达到更好的效果。因此，在数据挖掘中，选用好的模型和方法技巧至关重要。