In [1]:
```
import sklearn
import numpy as np
```

In [41]:
```
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.wrappers.scikit_learn import KerasClassifier
```

In [63]:
```
from sklearn.linear_model import LogisticRegression
```

In [2]:
```
from keras.datasets import imdb
(training_data, training_targets), (testing_data, testing_targets) = imdb.load_da
```

```
Using TensorFlow backend.
c:\program files\python37\lib\site-packages\tensorflow\python\framework\dtypes.
py:526: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is dep
recated; in a future version of numpy, it will be understood as (type, (1,)) /
'(1,)type'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
c:\program files\python37\lib\site-packages\tensorflow\python\framework\dtypes.
py:527: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is dep
recated; in a future version of numpy, it will be understood as (type, (1,)) /
'(1,)type'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
c:\program files\python37\lib\site-packages\tensorflow\python\framework\dtypes.
py:528: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is dep
recated; in a future version of numpy, it will be understood as (type, (1,)) /
'(1,)type'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
c:\program files\python37\lib\site-packages\tensorflow\python\framework\dtypes.
py:529: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is dep
recated; in a future version of numpy, it will be understood as (type, (1,)) /
'(1,)type'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
c:\program files\python37\lib\site-packages\tensorflow\python\framework\dtypes.
py:530: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is dep
recated; in a future version of numpy, it will be understood as (type, (1,)) /
'(1,)type'.
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
c:\program files\python37\lib\site-packages\tensorflow\python\framework\dtypes.
py:535: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is dep
recated; in a future version of numpy, it will be understood as (type, (1,)) /
'(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
```

In [3]:
```
training_data.shape
```

Out[3]:
```
(25000,)
```

In [5]:
```
training_targets
```

Out[5]:
```
array([1, 0, 0, ..., 0, 1, 0], dtype=int64)
```

In [6]:
```
np.count_nonzero(training_targets)
```

Out[6]:
```
12500
```

In [7]:
```
print("Categories:", np.unique(training_targets))
```

```
Categories: [0 1]
```

In [8]:
```
print("Number of unique words:", len(np.unique(np.hstack(training_data))))
```

```
Number of unique words: 9998
```

In [9]:
```
index = imdb.get_word_index()
reverse_index = dict([(value, key) for (key, value) in index.items()])
```

In [10]:
```
decoded = " ".join( [reverse_index.get(i - 3, "#") for i in training_data[0]] )
print(decoded)
```

```
# this film was just brilliant casting location scenery story direction everyon
e's really suited the part they played and you could just imagine being there r
obert # is an amazing actor and now the same being director # father came from
the same scottish island as myself so i loved the fact there was a real connect
ion with this film the witty remarks throughout the film were great it was just
brilliant so much that i bought the film as soon as it was released for # and w
ould recommend it to everyone to watch and the fly fishing was amazing really c
ried at the end it was so sad and you know what they say if you cry at a film i
t must have been good and this definitely was also # to the two little boy's th
at played the # of norman and paul they were just brilliant children are often
left out of the # list i think because the stars that play them all grown up ar
e such a big profile for the whole film but these children are amazing and shou
ld be praised for what they have done don't you think the whole story was so lo
vely because it was true and was someone's life after all that was shared with
us all
```

In [11]:
```
from keras.preprocessing import sequence
```

In [12]:
```
X_tr = sequence.pad_sequences(training_data, maxlen=100)
#X_ts = sequence.pad_sequences(testing_data, maxlen=100)
```

In [179]:
```
X_train_1 = []

for i in range(0,len(X_tr)):

    decoded = " ".join( [reverse_index.get(i - 3, "#") for i in X_tr[i]] )
    X_train_1.append(decoded)
```

In [14]:
```
#X_test = []

#for i in range(0,len(X_ts)):

#    decoded = " ".join( [reverse_index.get(i - 3, "#") for i in X_ts[i]] )
#    X_test.append(decoded)
```

In [180]:
```
X_train_1
```

Out[180]:
```
["cry at a film it must have been good and this definitely was also # to the
two little boy's that played the # of norman and paul they were just brillian
t children are often left out of the # list i think because the stars that pl
ay them all grown up are such a big profile for the whole film but these chil
dren are amazing and should be praised for what they have done don't you thin
k the whole story was so lovely because it was true and was someone's life af
ter all that was shared with us all",
 "funny in equal # the hair is big lots of boobs # men wear those cut # shirt
s that show off their # sickening that men actually wore them and the music i
s just # trash that plays over and over again in almost every scene there is
trashy music boobs and # taking away bodies and the gym still doesn't close f
or # all joking aside this is a truly bad film whose only charm is to look ba
ck on the disaster that was the 80's and have a good old laugh at how bad eve
rything was back then",
 "touching the floor at how bad it really was the rest of the time everyone e
lse in the theatre just started talking to each other leaving or generally cr
ying into their popcorn that they actually paid money they had # working to w
atch this feeble excuse for a film it must have looked like a great idea on p
aper but on film it looks like no one in the film has a clue what is going on
```

CountVectorizer and Tfidf

In [16]:
```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
import nltk
```

In [17]:
```
vec1 = CountVectorizer(min_df=2, tokenizer=nltk.word_tokenize)
```

In [18]:
```
train_cv = vec1.fit_transform(X_train_1)
```

In [19]:
```
train_cv
```

Out[19]:
```
<25000x9815 sparse matrix of type '<class 'numpy.int64'>'
	with 1706648 stored elements in Compressed Sparse Row format>
```

In [20]:
```
vec1.vocabulary_.get('good')
```

Out[20]:
```
3889
```

In [21]:
```
vec1.vocabulary_.get('hot')
```

Out[21]:
```
4329
```

In [22]:
```
train_cv.shape
```
Out[22]:
```
(25000, 9815)
```

In [61]:
```
train_cv.shape
```
Out[61]:
```
(25000, 9815)
```

In [64]:
```
X_train_cv = train_cv[:20000]
y_train_cv = training_targets[:20000]
X_test_cv = train_cv[20000::]
y_test_cv = training_targets[20000::]
```

In [65]:
```
print("X_train_cv shape: ", train_cv[:20000].shape)
print("y_train_cv shape: ", training_targets[:20000].shape)
print("X_test_cv shape: ", train_cv[20000::].shape)
print("y_test_cv shape: ", training_targets[20000::].shape)
```
```
X_train_cv shape:  (20000, 9815)
y_train_cv shape:  (20000,)
X_test_cv shape:  (5000, 9815)
y_test_cv shape:  (5000,)
```

Logistic Regression with CountVectorizer

In [66]:
```
classifier = LogisticRegression()
classifier.fit(X_train_cv, y_train_cv)
```
```
c:\program files\python37\lib\site-packages\sklearn\linear_model\logistic.py:43
2: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a
solver to silence this warning.
  FutureWarning)
```
Out[66]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=None, solver='warn', tol=0.0001, verbose=0,
                   warm_start=False)
```

In [68]:
```
print("Logistic Regression on Count Vectorizer Accuracy = ", classifier.score(X_t
```
```
Logistic Regression on Count Vectorizer Accuracy =  82.74000000000001
```

Neural Networks with Count Vectorizer

In [78]:
```
def build_model():
    model = Sequential()
    model.add(Dense(512, input_dim=9815, activation='elu'))
    model.add(Dropout(0.3))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(loss='binary_crossentropy', optimizer='sgd', metrics=['accuracy
    model.summary()
    return model
```

In [80]:
```
cv_estimator = KerasClassifier(build_fn=build_model, epochs=25, batch_size=64)
cv_estimator.fit(x = X_train_cv, y = y_train_cv,validation_split=0.25, workers=2)
```

```
Model: "sequential_16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_41 (Dense)             (None, 512)               5025792   
_________________________________________________________________
dropout_26 (Dropout)         (None, 512)               0         
_________________________________________________________________
dense_42 (Dense)             (None, 1)                 513       
=================================================================
Total params: 5,026,305
Trainable params: 5,026,305
Non-trainable params: 0
_________________________________________________________________
Train on 15000 samples, validate on 5000 samples
Epoch 1/25
15000/15000 [==============================] - 5s 364us/step - loss: 0.6416 - a
ccuracy: 0.6337 - val_loss: 0.5871 - val_accuracy: 0.7052
Epoch 2/25
15000/15000 [==============================] - 5s 349us/step - loss: 0.5688 - a
ccuracy: 0.7151 - val_loss: 0.5457 - val_accuracy: 0.7376
Epoch 3/25
15000/15000 [==============================] - 5s 349us/step - loss: 0.5303 - a
ccuracy: 0.7475 - val_loss: 0.5150 - val_accuracy: 0.7446
Epoch 4/25
15000/15000 [==============================] - 5s 349us/step - loss: 0.5028 - a
ccuracy: 0.7634 - val_loss: 0.4907 - val_accuracy: 0.7666
Epoch 5/25
15000/15000 [==============================] - 5s 350us/step - loss: 0.4759 - a
ccuracy: 0.7861 - val_loss: 0.4778 - val_accuracy: 0.7712
Epoch 6/25
15000/15000 [==============================] - 5s 349us/step - loss: 0.4626 - a
ccuracy: 0.7887 - val_loss: 0.4726 - val_accuracy: 0.7708
Epoch 7/25
15000/15000 [==============================] - 5s 349us/step - loss: 0.4503 - a
ccuracy: 0.7968 - val_loss: 0.4641 - val_accuracy: 0.7712
Epoch 8/25
15000/15000 [==============================] - 5s 348us/step - loss: 0.4339 - a
ccuracy: 0.8071 - val_loss: 0.4755 - val_accuracy: 0.7632
Epoch 9/25
15000/15000 [==============================] - 5s 350us/step - loss: 0.4207 - a
ccuracy: 0.8126 - val_loss: 0.4364 - val_accuracy: 0.7912
Epoch 10/25
15000/15000 [==============================] - 5s 351us/step - loss: 0.4128 - a
ccuracy: 0.8141 - val_loss: 0.4387 - val_accuracy: 0.7896
Epoch 11/25
15000/15000 [==============================] - 5s 350us/step - loss: 0.4048 - a
ccuracy: 0.8201 - val_loss: 0.4125 - val_accuracy: 0.8038
Epoch 12/25
15000/15000 [==============================] - 5s 348us/step - loss: 0.3991 - a
ccuracy: 0.8229 - val_loss: 0.4038 - val_accuracy: 0.8104
Epoch 13/25
15000/15000 [==============================] - 5s 349us/step - loss: 0.3896 - a
ccuracy: 0.8301 - val_loss: 0.4036 - val_accuracy: 0.8104
```

```
Epoch 14/25
15000/15000 [==============================] - 5s 349us/step - loss: 0.3793 - a
ccuracy: 0.8366 - val_loss: 0.3941 - val_accuracy: 0.8144
Epoch 15/25
15000/15000 [==============================] - 5s 352us/step - loss: 0.3752 - a
ccuracy: 0.8344 - val_loss: 0.3863 - val_accuracy: 0.8222
Epoch 16/25
15000/15000 [==============================] - 5s 349us/step - loss: 0.3697 - a
ccuracy: 0.8412 - val_loss: 0.5632 - val_accuracy: 0.7426
Epoch 17/25
15000/15000 [==============================] - 5s 349us/step - loss: 0.3605 - a
ccuracy: 0.8455 - val_loss: 0.3858 - val_accuracy: 0.8198
Epoch 18/25
15000/15000 [==============================] - 5s 348us/step - loss: 0.3672 - a
ccuracy: 0.8383 - val_loss: 0.4109 - val_accuracy: 0.8114
Epoch 19/25
15000/15000 [==============================] - 5s 349us/step - loss: 0.3556 - a
ccuracy: 0.8475 - val_loss: 0.3837 - val_accuracy: 0.8230
Epoch 20/25
15000/15000 [==============================] - 5s 350us/step - loss: 0.3428 - a
ccuracy: 0.8564 - val_loss: 0.4431 - val_accuracy: 0.7838
Epoch 21/25
15000/15000 [==============================] - 5s 349us/step - loss: 0.3446 - a
ccuracy: 0.8517 - val_loss: 0.3664 - val_accuracy: 0.8336
Epoch 22/25
15000/15000 [==============================] - 5s 350us/step - loss: 0.3420 - a
ccuracy: 0.8538 - val_loss: 0.3983 - val_accuracy: 0.8138
Epoch 23/25
15000/15000 [==============================] - 5s 350us/step - loss: 0.3388 - a
ccuracy: 0.8586 - val_loss: 0.3746 - val_accuracy: 0.8244
Epoch 24/25
15000/15000 [==============================] - 5s 350us/step - loss: 0.3360 - a
ccuracy: 0.8580 - val_loss: 0.3766 - val_accuracy: 0.8224
Epoch 25/25
15000/15000 [==============================] - 5s 350us/step - loss: 0.3292 - a
ccuracy: 0.8610 - val_loss: 0.3915 - val_accuracy: 0.8146
```

Out[80]:
```
<keras.callbacks.callbacks.History at 0x169eb55d608>
```

Comments

As we can see, a logstic regression model performed slightly better than my neural network. I think the reason is my neural network model is overfitting and the accuracy should be very similar to te logstic regression model. I tried a few different neural networks with different attributes but they all performed similarly and there was no improvements compared to the logistic regression.

In [ ]:
```
#vec2 = CountVectorizer(min_df=2, tokenizer=nltk.word_tokenize)
```

In [ ]:
```
#test_cv = vec2.fit_transform(X_test)
```

In [ ]:
```
#test_cv
```

In [ ]:
```
#vec2.vocabulary_.get('good')
```

In [ ]:
```
#vec2.vocabulary_.get('hot')
```

In [ ]:
```
#test_cv.shape
```

Tfidf

- [https://towardsdatascience.com/machine-learning-nlp-text-classification-using-scikit-learn-python-and-nltk-c52b92a7c73a (https://towardsdatascience.com/machine-learning-nlp-text-classification-using-scikit-learn-python-and-nltk-c52b92a7c73a)](https://towardsdatascience.com/machine-learning-nlp-text-classification-using-scikit-learn-python-and-nltk-c52b92a7c73a)

In [24]:
```
tfidf_transformer = TfidfTransformer()
```

In [25]:
```
train_tfidf = tfidf_transformer.fit_transform(train_cv)
#test_tfidf = tfidf_transformer.fit_transform(test_cv)
```

In [26]:
```
train_tfidf
```

Out[26]:
```
<25000x9815 sparse matrix of type '<class 'numpy.float64'>'
	with 1706648 stored elements in Compressed Sparse Row format>
```

In [ ]:
```
#print(test_tfidf)
```

In [27]:
```
train_tfidf.shape
```

Out[27]:
```
(25000, 9815)
```

In [33]:
```
X_train_tfidf = train_tfidf[:20000]
y_train = training_targets[:20000]
X_test_tfidf = train_tfidf[20000::]
y_test = training_targets[20000::]
```

In [34]:
```
print("X_train_tfidf shape: ", train_tfidf[:20000].shape)
print("y_train shape: ", training_targets[:20000].shape)
print("X_test_tfidf shape: ", train_tfidf[20000::].shape)
print("y_test shape: ", training_targets[20000::].shape)
```

```
X_train_tfidf shape:  (20000, 9815)
y_train shape:  (20000,)
X_test_tfidf shape:  (5000, 9815)
y_test shape:  (5000,)
```

In [ ]:
```
#test_tfidf.shape
```

A Naive Bayes model with Tfidf

In [35]:
```
from sklearn.naive_bayes import MultinomialNB
clf = MultinomialNB().fit(X_train_tfidf, y_train)
```

In [39]:
```
predicted = clf.predict(X_test_tfidf)
print( "Accuracy of Multinomial Nasive Bayes = ",(np.mean(predicted == y_test)*10
```

```
Accuracy of Multinomial Nasive Bayes =  83.0
```

Neural Networks for TF-IDF

In [56]:
```
def build_model():
    model = Sequential()
    model.add(Dense(512, input_dim=9815, activation='elu'))
    model.add(Dropout(0.4))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(loss='binary_crossentropy', optimizer='adadelta', metrics=['acc
    model.summary()
    return model
```

In [59]:
```
estimator = KerasClassifier(build_fn=build_model, epochs=10, batch_size=128)
estimator.fit(x = X_train_tfidf, y = y_train,validation_split=0.2, workers=2)
```

```
Model: "sequential_10"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_26 (Dense)             (None, 512)               5025792
_________________________________________________________________
dropout_17 (Dropout)         (None, 512)               0
_________________________________________________________________
dense_27 (Dense)             (None, 1)                 513
=================================================================
Total params: 5,026,305
Trainable params: 5,026,305
Non-trainable params: 0
_________________________________________________________________
Train on 16000 samples, validate on 4000 samples
Epoch 1/10
16000/16000 [==============================] - 8s 515us/step - loss: 0.6452 - a
ccuracy: 0.6856 - val_loss: 0.5721 - val_accuracy: 0.7605
Epoch 2/10
16000/16000 [==============================] - 8s 503us/step - loss: 0.4850 - a
ccuracy: 0.8068 - val_loss: 0.4316 - val_accuracy: 0.8058
Epoch 3/10
16000/16000 [==============================] - 8s 504us/step - loss: 0.3801 - a
ccuracy: 0.8461 - val_loss: 0.3867 - val_accuracy: 0.8305
Epoch 4/10
16000/16000 [==============================] - 8s 505us/step - loss: 0.3245 - a
ccuracy: 0.8714 - val_loss: 0.3479 - val_accuracy: 0.8415
Epoch 5/10
16000/16000 [==============================] - 8s 504us/step - loss: 0.2923 - a
ccuracy: 0.8845 - val_loss: 0.3355 - val_accuracy: 0.8518
Epoch 6/10
16000/16000 [==============================] - 8s 503us/step - loss: 0.2634 - a
ccuracy: 0.8995 - val_loss: 0.3353 - val_accuracy: 0.8503
Epoch 7/10
16000/16000 [==============================] - 8s 503us/step - loss: 0.2404 - a
ccuracy: 0.9099 - val_loss: 0.3303 - val_accuracy: 0.8587
Epoch 8/10
16000/16000 [==============================] - 8s 505us/step - loss: 0.2222 - a
ccuracy: 0.9174 - val_loss: 0.3350 - val_accuracy: 0.8593
Epoch 9/10
16000/16000 [==============================] - 8s 504us/step - loss: 0.2061 - a
ccuracy: 0.9247 - val_loss: 0.3425 - val_accuracy: 0.8535
Epoch 10/10
16000/16000 [==============================] - 8s 504us/step - loss: 0.1929 - a
ccuracy: 0.9325 - val_loss: 0.3470 - val_accuracy: 0.8558
```

Out[59]: `<keras.callbacks.callbacks.History at 0x169846ea648>`

Comments

The results of Tfidf make more sense compared to the CountVectorizer. The Naive Bayes model performed about 83% after applying Tfdif on dataset. It's slightly better than the performance of CountVectorizer with both neural networks or logistic regression. Also, we can see that the neural network performed better than the Naive Bayes model in Tfdif. Even though we can see that the neural network is againt overfitting. I tried different neural networks but they all performed as good as 86%.

In [ ]:

```

```

GENSIM

In [81]:

```
import gensim
from gensim import utils
from gensim.models.doc2vec import LabeledSentence
from gensim.models import Doc2Vec
```

Doc2Vec

In [137]:

```
from gensim.test.utils import common_texts
from gensim.models.doc2vec import Doc2Vec, TaggedDocument
from sklearn.metrics import accuracy_score, f1_score
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn import utils
import csv
from tqdm import tqdm
import multiprocessing
import nltk
from nltk.corpus import stopwords
```

In [192]:

```
def read_corpus(doc):
       for i, line in enumerate(doc):

            tokens = gensim.utils.simple_preprocess(line)

            yield gensim.models.doc2vec.TaggedDocument(tokens, str(training_targe
```

In [193]:

```
corp = list(read_corpus(X_train_1))
```

In [195]:
```
corp
```

Out[195]:
```
[TaggedDocument(words=['cry', 'at', 'film', 'it', 'must', 'have', 'been', 'go
od', 'and', 'this', 'definitely', 'was', 'also', 'to', 'the', 'two', 'littl
e', 'boy', 'that', 'played', 'the', 'of', 'norman', 'and', 'paul', 'they', 'w
ere', 'just', 'brilliant', 'children', 'are', 'often', 'left', 'out', 'of',
'the', 'list', 'think', 'because', 'the', 'stars', 'that', 'play', 'them', 'a
ll', 'grown', 'up', 'are', 'such', 'big', 'profile', 'for', 'the', 'whole',
'film', 'but', 'these', 'children', 'are', 'amazing', 'and', 'should', 'be',
'praised', 'for', 'what', 'they', 'have', 'done', 'don', 'you', 'think', 'th
e', 'whole', 'story', 'was', 'so', 'lovely', 'because', 'it', 'was', 'true',
'and', 'was', 'someone', 'life', 'after', 'all', 'that', 'was', 'shared', 'wi
th', 'us', 'all'], tags='1'),
 TaggedDocument(words=['funny', 'in', 'equal', 'the', 'hair', 'is', 'big', 'l
ots', 'of', 'boobs', 'men', 'wear', 'those', 'cut', 'shirts', 'that', 'show',
'off', 'their', 'sickening', 'that', 'men', 'actually', 'wore', 'them', 'an
d', 'the', 'music', 'is', 'just', 'trash', 'that', 'plays', 'over', 'and', 'o
ver', 'again', 'in', 'almost', 'every', 'scene', 'there', 'is', 'trashy', 'mu
sic', 'boobs', 'and', 'taking', 'away', 'bodies', 'and', 'the', 'gym', 'stil
l', 'doesn', 'close', 'for', 'all', 'joking', 'aside', 'this', 'is', 'truly',
'bad', 'film', 'whose', 'only', 'charm', 'is', 'to', 'look', 'back', 'on', 't
```

In [196]:
```
np.array(corp).shape
```

Out[196]:
```
(25000, 2)
```

In [197]:
```
train_doc = corp[:20000]
test_doc = corp[20000::]
```

In [ ]:
```

```

In [198]:
```
cores = multiprocessing.cpu_count()

model_dbow = Doc2Vec(vector_size=100, negative=5, hs=0, min_count=2, workers=core
model_dbow.build_vocab([x for x in tqdm(train_doc)])
train_documents  = utils.shuffle(train_doc)
model_dbow.train(train_documents,total_examples=len(train_doc), epochs=10)
def vector_for_learning(model, input_docs):
    sents = input_docs
    targets, feature_vectors = zip(*[(doc.tags[0], model.infer_vector(doc.words,
    return targets, feature_vectors
model_dbow.save('./movieModel.d2v')
```

```
100%|████████████████████████████████████████████████████████████| 2
0000/20000 [00:00<00:00, 4978402.37it/s]
```

In [199]:
```
gen_y_train, gen_X_train = vector_for_learning(model_dbow, train_documents)
gen_y_test, gen_X_test = vector_for_learning(model_dbow, test_doc)
```

In [201]:
```
gen_y_train
```

Out[201]:
```
('1',
 '0',
 '1',
 '1',
 '0',
 '1',
 '1',
 '1',
 '1',
 '0',
 '1',
 '1',
 '1',
 '0',
 '0',
 '0',
 '0',
 '0',
 '0',
```

In [202]:
```
logreg = LogisticRegression(n_jobs=1, C=1e5)
logreg.fit(gen_X_train, gen_y_train)
y_pred = logreg.predict(gen_X_test)
print('Testing accuracy', accuracy_score(gen_y_test, y_pred)*100)
print('Testing F1 score : {}'.format(f1_score(gen_y_test, y_pred, average='weight
```

```
c:\program files\python37\lib\site-packages\sklearn\linear_model\logistic.py:43
2: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a
solver to silence this warning.
  FutureWarning)

Testing accuracy 81.92
Testing F1 score : 81.91817747884573
```

Neural Networks with Gensim

In [224]:
```
def build_model():
    model = Sequential()
    model.add(Dense(1024, input_dim=100, activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(1024, activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(256, activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(256, activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(128, activation='relu'))
    model.add(Dropout(0.3))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(loss='binary_crossentropy', optimizer='adadelta', metrics=['acc
    model.summary()
    return model
```

In [225]:
```
gen_estimator = KerasClassifier(build_fn=build_model, epochs=5, batch_size=32)
gen_estimator.fit(x = np.asarray(gen_X_train), y = np.asarray(gen_y_train),valida
```

```
Model: "sequential_22"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_65 (Dense)             (None, 1024)              103424    
_________________________________________________________________
dropout_44 (Dropout)         (None, 1024)              0         
_________________________________________________________________
dense_66 (Dense)             (None, 1024)              1049600   
_________________________________________________________________
dropout_45 (Dropout)         (None, 1024)              0         
_________________________________________________________________
dense_67 (Dense)             (None, 256)               262400    
_________________________________________________________________
dropout_46 (Dropout)         (None, 256)               0         
_________________________________________________________________
dense_68 (Dense)             (None, 256)               65792     
_________________________________________________________________
dropout_47 (Dropout)         (None, 256)               0         
_________________________________________________________________
dense_69 (Dense)             (None, 128)               32896     
_________________________________________________________________
dropout_48 (Dropout)         (None, 128)               0         
_________________________________________________________________
dense_70 (Dense)             (None, 1)                 129       
=================================================================
Total params: 1,514,241
Trainable params: 1,514,241
Non-trainable params: 0
_________________________________________________________________
Train on 15000 samples, validate on 5000 samples
Epoch 1/5
15000/15000 [==============================] - 7s 489us/step - loss: 0.4024 - a
ccuracy: 0.8179 - val_loss: 0.3011 - val_accuracy: 0.8786
Epoch 2/5
15000/15000 [==============================] - 7s 457us/step - loss: 0.3192 - a
ccuracy: 0.8651 - val_loss: 0.3984 - val_accuracy: 0.8402
Epoch 3/5
15000/15000 [==============================] - 7s 457us/step - loss: 0.3112 - a
ccuracy: 0.8729 - val_loss: 0.3242 - val_accuracy: 0.8604
Epoch 4/5
15000/15000 [==============================] - 7s 456us/step - loss: 0.3021 - a
ccuracy: 0.8751 - val_loss: 0.3116 - val_accuracy: 0.8606
Epoch 5/5
15000/15000 [==============================] - 7s 457us/step - loss: 0.2926 - a
ccuracy: 0.8786 - val_loss: 0.2980 - val_accuracy: 0.8730
```

Out[225]: <keras.callbacks.callbacks.History at 0x16a70f714c8>

Comments

We can see that the logistic regression with Gensim did not performed very good. Maybe some attributes could be tuned or other machine learning models like Naive Bayes or SVM could be used to improve it. But the neural network model performed better than other vectorizers and models about 87%. I tried other neural networks too but the best performance so far was 87%. Based on my experience from the last two models with Tfidf and CountVectorizer I only trained my model for 5 epochs to prevent overfitting and it seems that the model isnt overfitting by looking at the val_loss.

Conclusion

As a conclusion, Neural network performed better than the machine learning classifiers except for CountVectorizer. Among the three vectorizers, Gensim performed the best with 87% and CountVectorizer had the poorest performance about 81%.

My failed attempts:

In [85]:
```
import nltk as nl
```

In [189]:
```
def read_corpus(doc, tokens_only=False):
       for i, line in enumerate(doc):
            tokens = gensim.utils.simple_preprocess(line)

            if tokens_only:
                yield tokens
            else:
                # For training data, add tags
                yield gensim.models.doc2vec.TaggedDocument(tokens, str(training_t
```

In [89]:
```
np.array(X_train_1).shape
```

Out[89]:
```
(25000,)
```

In [90]:
```
X_train_gen = X_train_1[:20000]
y_train_gen = training_targets[:20000]
X_test_gen = X_train_1[20000::]
y_test_gen = training_targets[20000::]
```

In [109]:
```
print(np.array(X_train_gen).shape)
print(np.array(X_test_gen).shape)
```

```
(20000,)
(5000,)
```

In [190]:
```
train_corpus = list(read_corpus(X_train_gen))
test_corpus = list(read_corpus(X_test_gen, tokens_only=True))
```

In [191]:
```
train_corpus
```

Out[191]:
```
[TaggedDocument(words=['cry', 'at', 'film', 'it', 'must', 'have', 'been', 'go
od', 'and', 'this', 'definitely', 'was', 'also', 'to', 'the', 'two', 'littl
e', 'boy', 'that', 'played', 'the', 'of', 'norman', 'and', 'paul', 'they', 'w
ere', 'just', 'brilliant', 'children', 'are', 'often', 'left', 'out', 'of',
'the', 'list', 'think', 'because', 'the', 'stars', 'that', 'play', 'them', 'a
ll', 'grown', 'up', 'are', 'such', 'big', 'profile', 'for', 'the', 'whole',
'film', 'but', 'these', 'children', 'are', 'amazing', 'and', 'should', 'be',
'praised', 'for', 'what', 'they', 'have', 'done', 'don', 'you', 'think', 'th
e', 'whole', 'story', 'was', 'so', 'lovely', 'because', 'it', 'was', 'true',
'and', 'was', 'someone', 'life', 'after', 'all', 'that', 'was', 'shared', 'wi
th', 'us', 'all'], tags='1'),
 TaggedDocument(words=['funny', 'in', 'equal', 'the', 'hair', 'is', 'big', 'l
ots', 'of', 'boobs', 'men', 'wear', 'those', 'cut', 'shirts', 'that', 'show',
'off', 'their', 'sickening', 'that', 'men', 'actually', 'wore', 'them', 'an
d', 'the', 'music', 'is', 'just', 'trash', 'that', 'plays', 'over', 'and', 'o
ver', 'again', 'in', 'almost', 'every', 'scene', 'there', 'is', 'trashy', 'mu
sic', 'boobs', 'and', 'taking', 'away', 'bodies', 'and', 'the', 'gym', 'stil
l', 'doesn', 'close', 'for', 'all', 'joking', 'aside', 'this', 'is', 'truly',
'bad', 'film', 'whose', 'only', 'charm', 'is', 'to', 'look', 'back', 'on', 't
```

In [141]:
```
np.array(train_corpus).shape
```

Out[141]:
```
(20000, 2)
```

In [93]:
```
test_corpus
```

Out[93]:
```
[['was',
  'probably',
  'creature',
  'ms',
  'is',
  'unfortunately',
  'not',
  'werewolf',
  'she',
  'is',
  'merely',
  'very',
  'strong',
  'lunatic',
  'br',
  'br',
  'as',
  'film',
  'werewolf',
```

In [96]:
```
import random
```

In [94]:
```
model = Doc2Vec()

model.build_vocab(train_corpus)
```

In [97]:
```
for epoch in range(10):
            model.train(
                train_corpus, total_examples= model.corpus_count,
                epochs=model.epochs)
            # shuffle the corpus
            random.shuffle(train_corpus)
            # decrease the learning rate
            model.alpha -= 0.0002
            # fix the learning rate, no decay
            model.min_alpha = model.alpha
```

In [ ]:
```
#model.train(train_corpus, total_examples=model.corpus_count, epochs=10)
```

In [134]:
```
model.wv.vocab.keys()
```

Out[134]:
```
dict_keys(['cry', 'at', 'film', 'it', 'must', 'have', 'been', 'good', 'and',
'this', 'definitely', 'was', 'also', 'to', 'the', 'two', 'little', 'boy', 'th
at', 'played', 'of', 'norman', 'paul', 'they', 'were', 'just', 'brilliant',
'children', 'are', 'often', 'left', 'out', 'list', 'think', 'because', 'star
s', 'play', 'them', 'all', 'grown', 'up', 'such', 'big', 'profile', 'for', 'w
hole', 'but', 'these', 'amazing', 'should', 'be', 'praised', 'what', 'done',
'don', 'you', 'story', 'so', 'lovely', 'true', 'someone', 'life', 'after', 's
hared', 'with', 'us', 'funny', 'in', 'equal', 'hair', 'is', 'lots', 'boobs',
'men', 'wear', 'those', 'cut', 'shirts', 'show', 'off', 'their', 'sickening',
'actually', 'wore', 'music', 'trash', 'plays', 'over', 'again', 'almost', 'ev
ery', 'scene', 'there', 'trashy', 'taking', 'away', 'bodies', 'gym', 'still',
'doesn', 'close', 'joking', 'aside', 'truly', 'bad', 'whose', 'only', 'char
m', 'look', 'back', 'on', 'disaster', 'old', 'laugh', 'how', 'everything', 't
hen', 'touching', 'floor', 'really', 'rest', 'time', 'everyone', 'else', 'the
atre', 'started', 'talking', 'each', 'other', 'leaving', 'or', 'generally',
'crying', 'into', 'popcorn', 'paid', 'money', 'had', 'working', 'watch', 'fee
ble', 'excuse', 'looked', 'like', 'great', 'idea', 'paper', 'looks', 'no', 'o
ne', 'has', 'clue', 'going', 'crap', 'acting', 'costumes', 'can', 'get', 'acr
oss', 'save', 'yourself', 'an', 'hour', 'bit', 'your', 'got', 'slightly', 'an
```

In [ ]:
```
train_arrays = numpy.zeros((25000, 100))

train_labels = numpy.zeros(25000)

for i in range(20000):



    train_arrays[i] = model
    train_arrays[12500 + i] = model[prefix_train_neg]
    train_labels[i] = 1
    train_labels[12500 + i] = 0
```

In [ ]:

In [ ]:

In [ ]:
```
#model.get_latest_training_loss()
```

In [98]:
```
model.wv.similarity('good','bad')
```

Out[98]:
```
0.5569832
```

In [99]:
```
words = list(model.wv.vocab)
```

In [100]:
```
len(words)
```

Out[100]:
```
9546
```

In [101]:
```
vectors = np.array(model.wv.vectors)
```

In [102]:
```
vectors
```

Out[102]:
```
array([[ 0.17890231,  0.33068714, -0.8207312 , ..., -0.01275392,
         0.3270439 ,  0.5880559 ],
       [ 0.04594428,  0.42859146, -0.08810625, ...,  0.77135634,
         1.0101948 ,  0.20413461],
       [-0.5275586 ,  0.69348824,  0.42044514, ...,  0.24094042,
         1.200559  ,  0.4792053 ],
       ...,
       [-1.7779503 , -1.2304244 ,  1.0064094 , ...,  0.7981529 ,
         0.8590613 ,  1.5844318 ],
       [ 1.0759941 ,  0.2052762 ,  1.2978704 , ...,  0.10349308,
         1.6053356 , -0.07444835],
       [-0.33308494,  0.4798437 ,  0.6466291 , ..., -0.5489158 ,
         0.46562156,  1.5704718 ]], dtype=float32)
```

In [103]:
```
vectors.shape
```

Out[103]:
```
(9546, 100)
```

In [105]:
```
model.wv.syn0
```

```
c:\program files\python37\lib\site-packages\ipykernel_launcher.py:1: Deprecatio
nWarning: Call to deprecated `syn0` (Attribute will be removed in 4.0.0, use se
lf.vectors instead).
  """Entry point for launching an IPython kernel.
```

Out[105]:
```
array([[ 0.17890231,  0.33068714, -0.8207312 , ..., -0.01275392,
         0.3270439 ,  0.5880559 ],
       [ 0.04594428,  0.42859146, -0.08810625, ...,  0.77135634,
         1.0101948 ,  0.20413461],
       [-0.5275586 ,  0.69348824,  0.42044514, ...,  0.24094042,
         1.200559  ,  0.4792053 ],
       ...,
       [-1.7779503 , -1.2304244 ,  1.0064094 , ...,  0.7981529 ,
         0.8590613 ,  1.5844318 ],
       [ 1.0759941 ,  0.2052762 ,  1.2978704 , ...,  0.10349308,
         1.6053356 , -0.07444835],
       [-0.33308494,  0.4798437 ,  0.6466291 , ..., -0.5489158 ,
         0.46562156,  1.5704718 ]], dtype=float32)
```

In [106]:
```
model.wv.syn0.shape
```

```
c:\program files\python37\lib\site-packages\ipykernel_launcher.py:1: Deprecatio
nWarning: Call to deprecated `syn0` (Attribute will be removed in 4.0.0, use se
lf.vectors instead).
  """Entry point for launching an IPython kernel.
```

Out[106]:
```
(9546, 100)
```

In [104]:
```
training_targets
```

Out[104]:
```
array([1, 0, 0, ..., 0, 1, 0], dtype=int64)
```

In [ ]:
```
training_targets.shape
```

In [ ]:
```
model.save('./imdb.d2v')
```

In [ ]:
```
model = Doc2Vec.load('./imdb.d2v')
```

In [ ]:
```
model['']
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:
```
clf = LogisticRegression().fit(model.wv.syn0, training_targets[:max_dataset_size]
```

In [ ]:
```
predict = clf.predict(model.wv.syn0[:100, :])
# Calculating the score of the predictions
score = clf.score(model.wv.syn0, training_targets[:max_dataset_size])
print("\nPrediction word2vec : \n", predict)
print("Score word2vec : \n", score)
```

In [ ]:

In [ ]:
```
print(model['horrible'])
```

In [ ]:
```
w1 = "ok"
model.wv.most_similar (positive=w1)
```

In [ ]:
```
model.save('./imdb.d2v')
```

In [ ]:
```
new_model = gensim.models.Word2Vec.load('./imdb.d2v')
```

- https://machinelearningmastery.com/develop-word-embeddings-python-gensim/ (https://machinelearningmastery.com/develop-word-embeddings-python-gensim/)

- https://kavita-ganesan.com/gensim-word2vec-tutorial-starter-code/#.XoEUvohKiUI (https://kavita-ganesan.com/gensim-word2vec-tutorial-starter-code/#.XoEUvohKiUI)
- https://stackoverflow.com/questions/49643974/how-to-do-text-classification-using-word2vec (https://stackoverflow.com/questions/49643974/how-to-do-text-classification-using-word2vec)
- https://radimrehurek.com/gensim/auto_examples/tutorials/run_doc2vec_lee.html (https://radimrehurek.com/gensim/auto_examples/tutorials/run_doc2vec_lee.html)
- https://github.com/ibrahimsharaf/doc2vec/blob/master/models/doc2vec_model.py (https://github.com/ibrahimsharaf/doc2vec/blob/master/models/doc2vec_model.py)