Sentiment Analysis for Online Shopping _Amazon Products

Team 1 Project Report

Team Member: Ling Ao, Zhoutao Cao, Yanran Min, Xuan Zhao

Course: BIA678 (A)

Professor: David Belanger

Data :12/11/2020

1. Introduction

With the development of e-commerce, people are becoming more and more willing to ship online because they do not need to go out. Before making a purchase decision, an individual usually will ask the opinion from families or friends. If he/she cannot get useful information from people around, reading reviews of others is also a good choice. According to previous research, more than 95% customers view online reviews as an important factor to make purchase decisions. Sentiment analysis is used to forecast the rating and comments of some things, such as movies, products, and shows. Due to the pandemic COVID-19, this year is very special. People cannot go out. Under this situation, Sentiment analysis becomes more important. The sentiment analysis results not only can help customers make purchase decisions faster, but also let sellers know the overall feeling of one product.

As the largest online vendor in the word, Amazon has numerous products, and the product review datasets are very large. The reviews of amazon products are about product and service (Bhatt et al., 2015), which can provide more useful information. Therefore, our research question is:

Which model is the best for sentiment analysis of Amazon product reviews?

In this paper, we choose four models: Long-Short Term Memory (LSTM), Multi-Layer Perceptron (MLP), Logistic regression and Naive bayes to do the sentiment analysis of two products' reviews, CDs_and_Vintl and Office Products. The results show that both Logistic regression and LSTM perform good, and have the accuracy above 80%. The rest of the paper is organized as: part 2 lists some related works, part 3 introduces datasets, part 4 methodology shows the technologies and four models, and part 5 is the discussion and conclusion.

2.Related works

Sentiment analysis can provide useful indicators for different purposes (Prabowo & Thelwall, 2009). The analysis results can help customers make purchase decisions, help companies estimate the extent of products acceptance and make further development strategies, and help policy makers to analyze public sentiments.

Scholars have adopted various different models to do sentiment analysis. Natural Language Processing (Yi et al. 2003; Nasukawa & Yi 2003; Hiroshi et al. 2004; Ko¨nig & Brill 2006) is a very common one. Machine learning (both supervised learning and unsupervised learning), such as Naive Bayes, Support Vector Machine, Maximum Entropy, and unsupervised learning (Pang et al., 2002). Prabowo & Thelwall adopted a combined approach to do sentiment analysis in 2009, and prove that a combined approach performs better than a single algorithm (Prabowo & Thelwall, 2009). Many existing works also use Amazon reviews to do sentiment tasks. Amazon Bhatt et al. used POS tagging technique to do classification and sentiment tasks with Amazon reviews. In 2018, Haque et al used a machine learning method to do the sentiment analysis with Amazon product reviews.

3. Data collection

We downland the amazon review dataset from amazon official website, and it was organized by Jianmo Ni. The dataset contains all reviews about 27 categories, total 233.1 million, in the range of May 1996 to Oct 2018. We choose two subsets: the reviews of CDs_and_Vintl, which contains 4,543,369 reviews; and Office Products, which contains 5,581,313 reviews to do this research. The dataset contains 11: reveiwr ID, asin (ID of the product), reviewer Name, vote (helpful votes), style, reviewText, overall rating, summary, unixreviewTime, reviewTime, image (images that users post after they have received the product. The main attributes we use are reviewText and overall rating. Overall rating is range

from 1 to 5. In this study, we label negative if the rating is from 1 to 3, and label positive if the rating is 4 or 5.

4. Methodology

In this research we choose 4 models: LSTM, Multi-Layer Perceptron (MLP), Logistic regression and Naive bayes. And we choose python to do our machine learning. The data is around 2G before data cleaning. For our personal computers, it is very hard work. As a result, we choose EMR of AWS and Google Cloud to do machine learning at the same time. Due to different models have different algorithms and implementations. It is hard for us to compare and choose the best one. So that, we divided the data into ten data sizes: 100,1000,5000,10000,100000,200000,400000,800000,1200000,1600000. Each model could choose different data sizes depending on the performance of models.

4.1 MLP Model

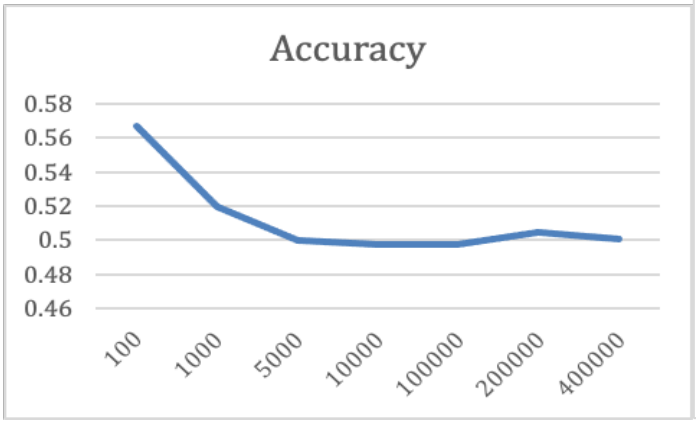
MLP model's full name is Multi-Layer Perceptron. It comes from Perceptron Learning Algorithm. We have a simple introduction for it.

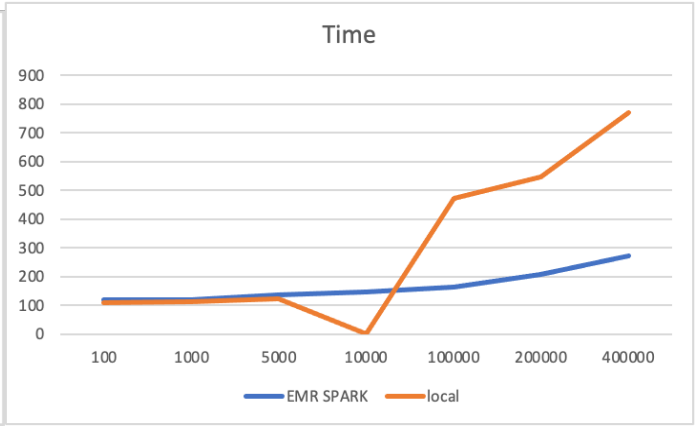
One of the most important properties of a Multi-Layer Perceptron is it has many layers. The first layer is the input layer and there is the output layer as the last layer. The layers between the input layers and output layers are hidden layers. There is no limit for hidden layers. First step of this section is cleaning data. We would like to choose the feature and ratings, then convert them into the data which are suitable for machine learning. At the same time, we need to choose different data sizes. For the result with more accuracy, we randomly choose the data with 50 percent negative and 50 percent positive. For the data size we choose is 100,1000,5000,10000,100000,200000,400000. They are enough for this model. Through this processing, we get the data as below.

Then we split the data to 70 percent training data and 30 percent testing data. The MLP model is already in Sklearning package, we just use the code from package and at the

label	0	1	2	3	4	5	6	7	8	9	...	90	91	92	93	94	95	96	97	98	99
1.0	215	10	42	215	10	27	23	1	62	1015	...	0	0	0	0	0	0	0	0	0	0
1.0	1	285	8	22	3	1	165	244	11	1	...	13	2	14	44	7	72	85	79	47	3
1.0	292	1	187	11	1056	2	81	4	23	10	...	0	0	0	0	0	0	0	0	0	0
0.0	74	22	53	48	45	15	10	17	45	6	...	0	0	0	0	0	0	0	0	0	0
1.0	1059	1060	2	5	247	454	3	1061	20	80	...	218	29	5	1078	1079	1080	168	650	40	75
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1.0	2905	458	2906	17	1	62	390	924	2907	2908	...	0	0	0	0	0	0	0	0	0	0
0.0	6	118	24	368	12	7	145	29	2913	62	...	0	0	0	0	0	0	0	0	0	0
0.0	532	7	522	3	1	1002	2917	125	17	5	...	0	0	0	0	0	0	0	0	0	0
0.0	7	8	437	24	1	62	2920	3	335	2921	...	161	102	2933	51	439	412	34	2934	38	894
0.0	60	3	32	6	218	2937	12	6	39	1	...	9	1	257	324	31	2950	2	346	943	6

same time we use cross validation to get the training score compare with the testing result. However, we found that the training score does not performance better than testing score. At last, the accuracy of this model is around 50%. It is not a ideal result. And this is the part for local machine. Then we need to train the model in the EMR of AWS which is the cloud computing provide by Amazon. In EMR, we use spark and train the MLP model. We can compare the running time for both models.





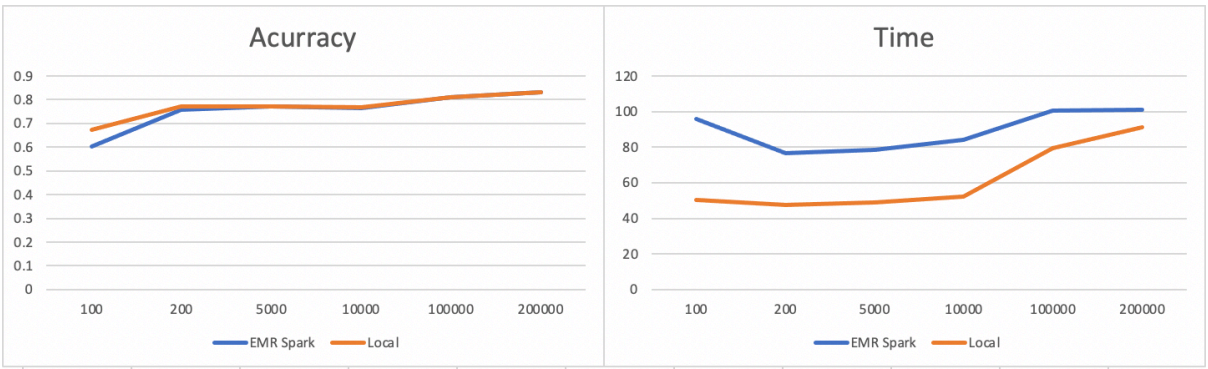
By comparing, we found that if we use the data less than 10000, the personal computing is better than AWS. When the data size increases, the personal computer looks like it uses much more time than AWS. However, The computer which is training this model has a GPU, if we try this model in another personal computer, the result is totally different.

4.2 Logistic Regression

We choose Logistic Regression as our benchmarking model because of its simplicity and reliability. In industrial practice it is also considered as one of the fundamental machine learning algorithms for baseline benchmarking. In our case, the input is the customer's reviews in text format while the output is a boolean value which indicates whether the review is "positive" or "negative". To make the problem simple, we say a rating that is greater than 3 is considered to be "positive" and vice versa. Hence we are able to transform the rate which is a scale from 1 to 5, to a boolean.

Apparently the difficulty here is how to vectorize the input text into the feature vectors we could measure. Here we adopt the TF-IDF method, a.k.a. term frequency–inverse document frequency, which is widely used in NLP. It will transform the text data into a vector space where we can apply our algorithm.

Our next step is to split the data into training sets and testing sets. To make a comprehensive and fair comparison between different algorithms, we make the training data set of size 100, 1000, 5000, 10000, 100000, 200000, 400000 and the testing sets adjusted accordingly to make a 70/30 ratio. After building up the training and testing pipeline, we experiment on both local and AWS for the exact same script and the result shows that AWS has a pretty smooth growth w.r.t. Time spent V.S. data size, while local running time increases dramatically when the input size grows.

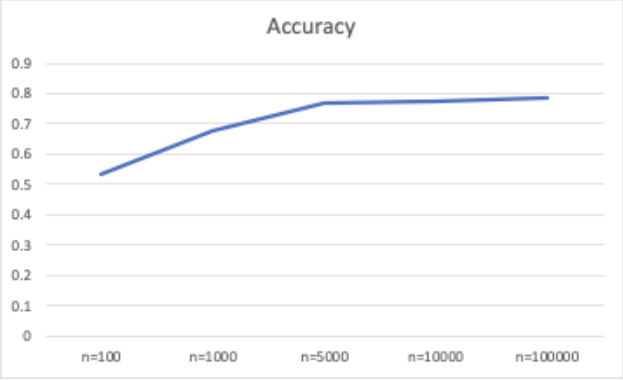


4.3 Naive Bayes Model

Naive Bayes model is greatly used in natural language processing and text classification for its ability to handle a large number of features. In our dataset, there are tens of thousands of words in the "text" column. Multinomial Naive Bayes classifier performs well with larger vocabulary sizes as it treats each word as a unique feature, and captures word frequency information to help classification(Andrew & Kamal Nigam, 1998)

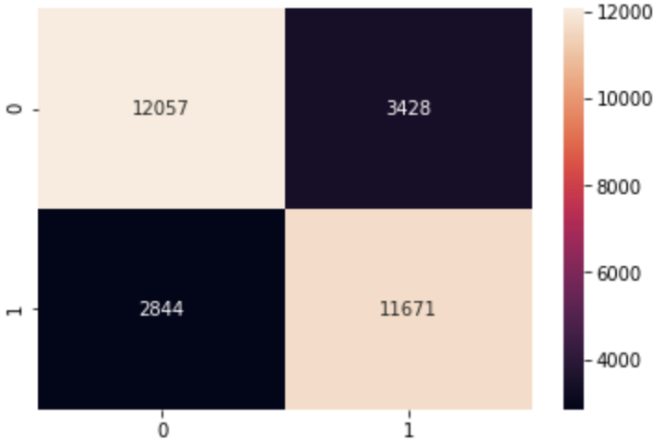
To build the model, we first divide the ratings into positive and negative, labeling positive as 1 and negative as 0. For text data, in order to generate more meaningful results, we use NLTK in Python to tokenize text, remove stop words and punctuation. A new column called "cleaned_reviews" is created to reflect these changes. Next, we split the dataset into 70% of training and 30% of testing. Same as in the logistic regression model, we utilize term frequency TF-IDF to vectorize "cleaned_reviews" text into the feature vectors that we can measure. We create two data frames, count_df and tfidf_df, and calculate their difference.

Then, we start by fitting our model. In order to measure the impact of scale on quality of analysis, we take samples from the dataset at 100, 1000, 5000, 10,000, and 100,000. When the sample size is 100,000, we get the highest accuracy of 0.7909, which indicates that accuracy goes up as we increase the sample size.

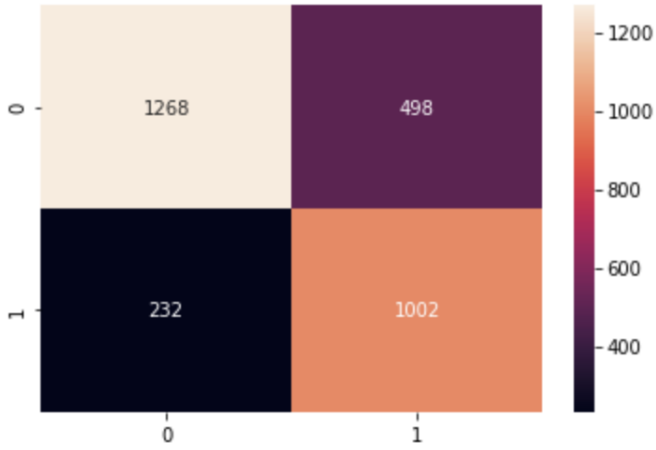


As shown in the Heatmap 1, there are large overlaps in the results of "1" (lower right corner) and "0" (upper left corner) between the prediction model and testing data, meaning

the model can predict positive/negative ratings based on reviews pretty accurately when the sample data size is large enough.



Heatmap1. Sample = 100,000



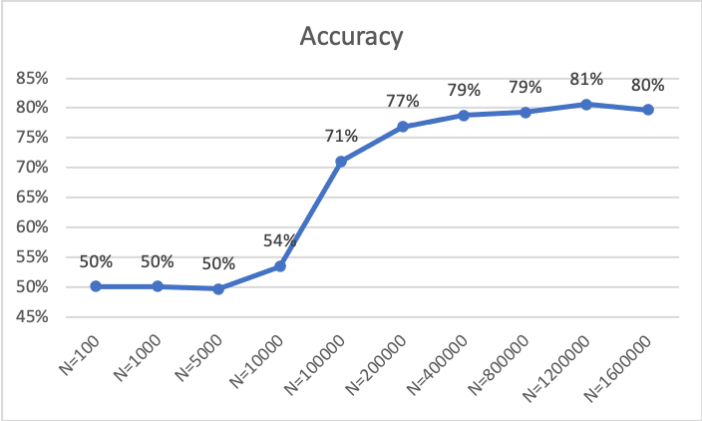
Heatmap2. Sample = 10,000

4.4 LSTM

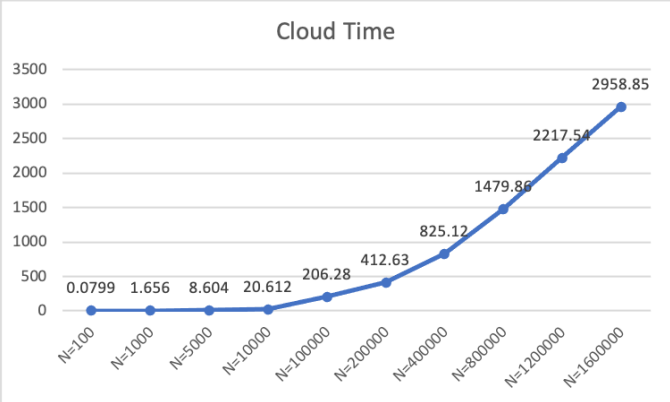
LSTM stands for long-short term memory, which is a special recurrent neural network model used in deep learning. Hochreiter Sepp and Schmidhuber Jurgen proposed LSTM in 1997 (Hochreiter& Schmidhuber, 1997). Compared with the common RNN model, LSTM works better on long sequences due to the function of selecting memory. Wang et al. used LSTM in twitter sentiment prediction and got the results that can be compared with the best data-driven model at that time (Wang et al, 2015). LSTM was applied to do dimensional sentiment analysis in 2016 (Wang et al. 2016). Since the average length of the reviews is 215, LSTM is a good choice to do the sentiment analysis.

Data process is the first step. There are four parts in data processing: delete all unmeaningful reviews, which are blank, convert all characters into lower case, split the words by blank or punctuation, remove all punctuation. Embedding words is the second step. Considering the running time, the maximum length of each review is limited to 100. Then use

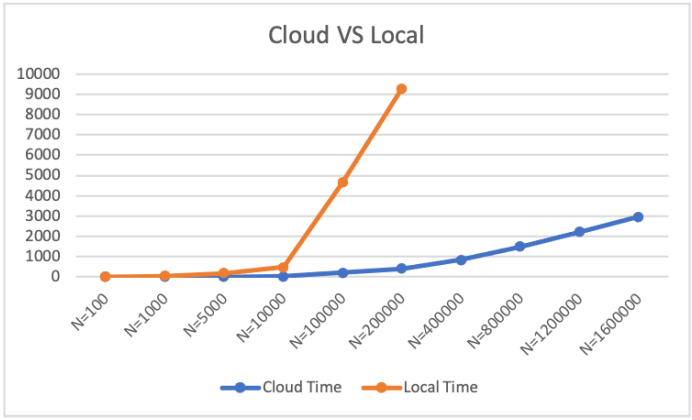
the extracted words to create the dictionary. In order to guarantee the sufficient training samples, training size to testing size is 9:1. Creating an LSTM network is the third step. Package torch.nn is used. Set output size to 1, embedding to 200, hidden dimension to 128, and 3 layers. With existing LSTM, training is the fourth step. Set epochs to 1 and gradient clipping to 500. 10 sizes of samples are used:100, 1000, 5000,10000,100000,200000,400000, 800000,1200000,1600000. Testing is the final step. The accuracy with 800000, 1200000,1600000 are 79.3%, 80.6% and 79.7% respectively. From the accuracy trend, it is obvious that when the sample size is 1600000, overfitting appeared; therefore, I stop adding samples and think the highest accuracy is 80.6% with the above parameters setting.



Adopted measurement of impact of scale on performance is time. From the time trend, it is obvious that with the increasing sample size, running time increases and the slope of the curve also increases.



LSTM is a complex recurrent neural network model, so it's hard to be parallelized. The cloud computing platform called google colab is chosen to run this model. For the time comparison, I run the model by using GPU and CPU. The running time with GPU is cloud time, and running time with CPU is local time. The below chat shows that: cloud time is much shorter than local time; and with the increase of sample size, the gap between cloud time and local time is becoming larger. Since the running time of CPU is too long, I just run the first six samples.



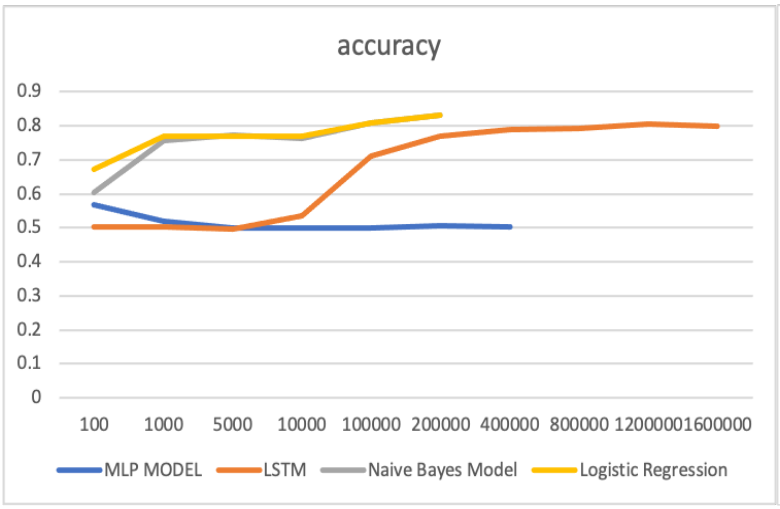
5. Discussion & Conclusion

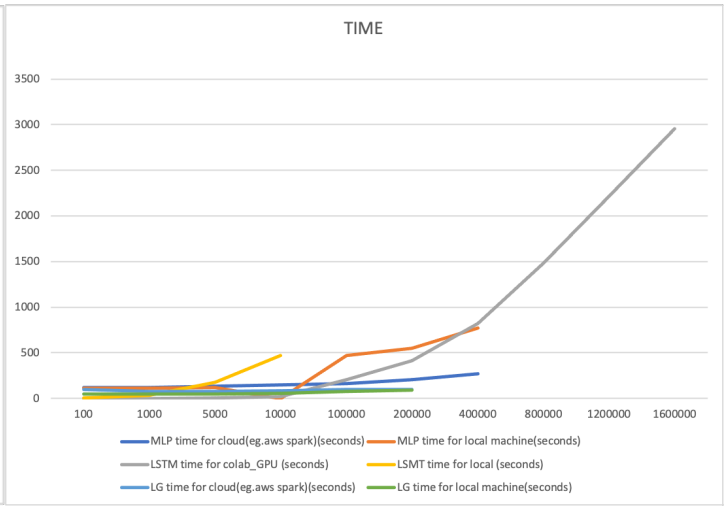
From our project, we learnt that not every machine learning model could use spark or MapReduce. Then let's combine these four methods and compare with the result. For the speed part, the speed of logistic regression is the fastest one. It performs very well on both local machines and EMR of AWS. And LSTM is a little bit slower than others. Because the model is more complex than other models and it could not do parallel computing. When we increase the size of the data, logistic regression's performance is the best one.

For the accuracy part, we found that the MLP is the lowest accuracy 50%. And we change the data size of MLP, the accuracy is also the lowest one. So that MLP may not be very suitable for this model. For Naive Bayes Model, LSTM model and Logistic regression, they perform low at small size of the data. When the data size becomes bigger, the accuracy

of them will increase. At the same time, LSTM grows very slowly, and Logistic regression grows fastest. We could see LSTM need to change the data size to 1200000, then it has the best performance.

As a result, if using a small data size to train a model, Logistic regression and Naive Bayes Model are the first choice. They perform faster and with more accuracy. If the data size is big enough, we should choose LSTM for training.





Reference

Bhatt, A., Patel, A., Chheda, H., & Gawande, K. (2015). Amazon review classification and sentiment analysis. *International Journal of Computer Science and Information Technologies*, *6*(6), 5107-5110.

Brownlee, J. (2019, August 19). When to Use MLP, CNN, and RNN Neural Networks. Retrieved December 11, 2020, from https://machinelearningmastery.com/when-to use-mlp-cnn-and-rnn-neural-networks/

Data description: https://nijianmo.github.io/amazon/index.html#sample-review

Haque, T. U., Saber, N. N., & Shah, F. M. (2018, May). Sentiment analysis on large scale Amazon product reviews. In 2018 IEEE International Conference on Innovative Research and Development (ICIRD) (pp. 1-6). IEEE.

Hiroshi, K., Tetsuya, N., & Hideo, W. (2004, August). Deeper sentiment analysis using machine translation technology. In *Proceedings of the 20th international conference on Computational Linguistics* (p. 494). Association for Computational Linguistics.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural computation, 9(8),1735-1780.

McCallum, A., & Nigam, K. (1998, July). A comparison of event models for naive bayes text classification. In *AAAI-98 workshop on learning for text categorization* (Vol. 752, No. 1, pp. 41-48).

Nasukawa, T., & Yi, J. (2003, October). Sentiment analysis: Capturing favorability using natural language processing. In *Proceedings of the 2nd international conference on Knowledge capture* (pp. 70-77).

Pang, B., Lee, L., & Vaithyanathan, S. (2002). Thumbs up? Sentiment classification using machine learning techniques. *arXiv preprint cs/0205070*.

Prabowo, R., & Thelwall, M. (2009). Sentiment analysis: A combined approach. *Journal of Informetrics*, *3*(2), 143-157.

Wang, J., Yu, L. C., Lai, K. R., & Zhang, X. (2016, August). Dimensional sentiment analysis using a regional CNN-LSTM model. In Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers) (pp. 225 230).

Wang, X., Liu, Y., Sun, C. J., Wang, B., & Wang, X. (2015, July). Predicting polarities of tweets by composing word embeddings with long short-term memory. In Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers) (pp. 1343-1353).

Yi, J., Nasukawa, T., Bunescu, R., & Niblack, W. (2003, November). Sentiment analyzer: Extracting sentiments about a given topic using natural language processing techniques. In *Third IEEE international conference on data mining* (pp. 427-434). IEEE.

Appendix

Naive Bayes Model

In [67]:
```
import numpy as np
import pandas as pd
from string import punctuation
from string import digits
from sklearn.model_selection import train_test_split
```

In [68]:
```
import pandas as pd
large_classification_data = pd.read_csv("/Users/xuanzhao/Desktop/CDs_and
_Vintl_sentiment.csv",error_bad_lines=False)
large_classification_data.index = range(len(large_classification_data))

## drop unmeanningful reviews
new_large_classification_data=large_classification_data.drop(labels=[191
7,43407, 62315,64996,68579,74976,82369,
                                                                    90049,98337
,101219,139591,130787,267578,
                                                                    295804,3358
77,396767,159574,164137,223911,
                                                                    254229,2602
36,261849,271662,287960,288971,
                                                                    374860,3789
99,380784,383545],axis=0)
new_small_classification_data = new_large_classification_data.sample(100
00, replace = False)
```

In [69]:
```
new_small_classification_data.head(5)
```

Out[69]:

	text	label
288852	Let me mention the positive aspects first, the...	0.0
351392	I go look at the CD here on amazon.com and the...	0.0
372751	I can't believe it says they wrote over 150 so...	0.0
104173	2012 proved to be a horrible of year in the mu...	1
242385	This could be a good equation, right? Dan's s...	0

In [70]:
```
from string import punctuation
import nltk
nltk.download('stopwords')
nltk.download('punkt')
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

#convert to lower case and remove punctuations
new_small_classification_data['text'] = new_small_classification_data['t
ext'].astype(str)
new_small_classification_data['text'] = new_small_classification_data.te
xt.apply(lambda x: x.lower())
new_small_classification_data['text'] = new_small_classification_data.te
xt.apply(lambda x: ''.join([c for c in x if c not in punctuation]))

#remove stopwords
stop_words = set(stopwords.words('english'))

new_small_classification_data['cleaned_reviews'] = new_small_classificat
ion_data.text.apply(lambda x: word_tokenize(x))

new_small_classification_data['cleaned_reviews'] = new_small_classificat
ion_data.cleaned_reviews.apply(lambda x: [w for w in x if w not in stop_
words])

new_small_classification_data['cleaned_reviews'] = new_small_classificat
ion_data.cleaned_reviews.apply(lambda x: ' '.join(x))
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/xuanzhao/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to /Users/xuanzhao/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

In [71]:
```
new_small_classification_data.head()
```

Out[71]:

	text	label	cleaned_reviews
288852	let me mention the positive aspects first the ...	0.0	let mention positive aspects first cello prelu...
351392	i go look at the cd here on amazoncom and the ...	0.0	go look cd amazoncom reviews awsome get cd rea...
372751	i cant believe it says they wrote over 150 son...	0.0	cant believe says wrote 150 songs album wonder...
104173	2012 proved to be a horrible of year in the mu...	1	2012 proved horrible year music industry artis...
242385	this could be a good equation right dans spac...	0	could good equation right dans spacy aesthetic...

In [72]:
```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
```

In [73]:
```
import numpy as np
from sklearn.model_selection import train_test_split

new_small_classification_data['label'] = new_small_classification_data[
'label'].astype(str)
y = new_small_classification_data['label']

X_train, X_test, y_train, y_test = train_test_split(new_small_classifica
tion_data["cleaned_reviews"], y, test_size=0.30 ,random_state=53)

count_vectorizer = CountVectorizer(stop_words="english")

count_train = count_vectorizer.fit_transform(X_train)

y_train = np.asarray(y_train.values)

ch2 = SelectKBest(chi2, k = 300)

X_new = ch2.fit_transform(count_train, y_train)

count_test = count_vectorizer.transform(X_test)

X_test_new = ch2.transform(X=count_test)
```

In [74]:
```
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf_vectorizer = TfidfVectorizer(stop_words="english", max_df=0.7)

# Transform the training data: tfidf_train
tfidf_train = tfidf_vectorizer.fit_transform(X_train)

# Transform the testing data: tfidf_test
tfidf_test = tfidf_vectorizer.transform(X_test)
```

In [75]:
```
# Create dataFrame: count_df
count_df = pd.DataFrame(count_train.A, columns=count_vectorizer.get_feat
ure_names())

# Create dataFrame: tfidf_df
tfidf_df = pd.DataFrame(tfidf_train.A, columns=tfidf_vectorizer.get_feat
ure_names())

print(count_df.head())
print(tfidf_df.head())

difference = set(count_df.columns) - set(tfidf_df.columns)
print(difference)
```

```
   000  00001  002  002cheers  007  00s  01  0145  016  0205  ...  zuri
ch  \
0    0      0    0          0    0    0   0     0    0     0  ...
0
1    0      0    0          0    0    0   0     0    0     0  ...
0
2    0      0    0          0    0    0   0     0    0     0  ...
0
3    0      0    0          0    0    0   0     0    0     0  ...
0
4    0      0    0          0    0    0   0     0    0     0  ...
0

   zus  zutons  zwan  zydeco  zydecoreggae  zyx  zz  zztop  \
0    0       0     0       0             0    0   0      0
1    0       0     0       0             0    0   0      0
2    0       0     0       0             0    0   0      0
3    0       0     0       0             0    0   0      0
4    0       0     0       0             0    0   0      0

   zzzzzzzzzzzzzzzzzzzzzzoh
0                         0
1                         0
2                         0
3                         0
4                         0

[5 rows x 53650 columns]
   000  00001  002  002cheers  007  00s   01  0145  016  0205  ...  zur
ich  \
0  0.0    0.0  0.0        0.0  0.0  0.0  0.0   0.0  0.0   0.0  ...
0.0
1  0.0    0.0  0.0        0.0  0.0  0.0  0.0   0.0  0.0   0.0  ...
0.0
2  0.0    0.0  0.0        0.0  0.0  0.0  0.0   0.0  0.0   0.0  ...
0.0
3  0.0    0.0  0.0        0.0  0.0  0.0  0.0   0.0  0.0   0.0  ...
0.0
4  0.0    0.0  0.0        0.0  0.0  0.0  0.0   0.0  0.0   0.0  ...
0.0

   zus  zutons  zwan  zydeco  zydecoreggae  zyx   zz  zztop  \
0  0.0     0.0   0.0     0.0           0.0  0.0  0.0    0.0
1  0.0     0.0   0.0     0.0           0.0  0.0  0.0    0.0
2  0.0     0.0   0.0     0.0           0.0  0.0  0.0    0.0
3  0.0     0.0   0.0     0.0           0.0  0.0  0.0    0.0
4  0.0     0.0   0.0     0.0           0.0  0.0  0.0    0.0

   zzzzzzzzzzzzzzzzzzzzzzoh
0                       0.0
1                       0.0
2                       0.0
3                       0.0
4                       0.0

[5 rows x 53650 columns]
set()
```

In [76]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics
from sklearn.naive_bayes import MultinomialNB

# Naive Bayes classifier
nb_classifier = MultinomialNB()

# Apply the classifier to the training data
nb_classifier.fit(X_new, y_train)
pred = nb_classifier.predict(X_test_new)

# Calculate accuracy
score = metrics.accuracy_score(y_test, pred)
print('Accuracy:',score)
f1 = metrics.f1_score(y_test, pred, average='micro')
print('F score:',f1)
```

```
Accuracy: 0.7496666666666667
F score: 0.7496666666666667
```

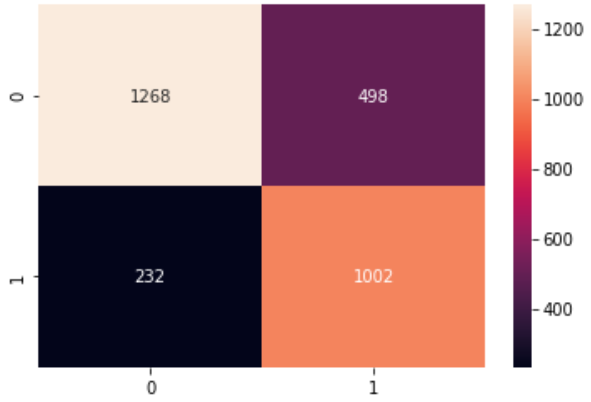
In [77]:
```
# Naive Bayes classifier when remove some meaningless words
nb_classifier = MultinomialNB()

# Apply the classifier to the training data
nb_classifier.fit(tfidf_train,y_train)
pred = nb_classifier.predict(tfidf_test)

# Calculate accuracy
score = metrics.accuracy_score(y_test, pred)
print('Accuracy:',score)
f1 = metrics.f1_score(y_test, pred, average='micro')
print('F score:',f1)
```

```
Accuracy: 0.7566666666666667
F score: 0.7566666666666667
```

In [78]:
```
import matplotlib.pyplot as plt
import seaborn as sns
heatmap2 = sns.heatmap(metrics.confusion_matrix(pred,y_test),annot=True,
fmt='2.0f')
```



MLP model

In [3]:
```
import time

start=time.time()
## ------ Classification data ------- ##
import gzip
import os
import json
import pandas as pd
import numpy as np
from string import punctuation
from string import digits
from gensim.models import word2vec
import gzip
import os
import json
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from keras.preprocessing.sequence import pad_sequences
from pandas.core.frame import DataFrame
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.naive_bayes import MultinomialNB
from sklearn import metrics
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
import torch
import torch.nn as nn
from collections import Counter
import keras
from keras.preprocessing.sequence import pad_sequences
from torch.utils.data import TensorDataset, DataLoader

def un_gz(file_name):
    f_name = file_name.replace(".gz", "")
    g_file = gzip.GzipFile(file_name)
    open(f_name, "wb+").write(g_file.read())
    g_file.close()

un_gz("reviews_CDs_and_Vinyl.json.gz")
reviews_df1 = pd.DataFrame(data_list)
reviews_df2 = reviews_df1[['reviewText', 'helpful', 'overall']]

## create target variable 1 sentiment
## if overall ="4", "5", it means positive, represented by "1".
sentiment = np.array(reviews_df2['overall']) > 3
sentiment = sentiment + 0

review_text = reviews_df2['reviewText']

## Resample the imbalanced data
positive_text = review_text[sentiment == 1]
negative_text = review_text[sentiment == 0]
print(len(positive_text), len(negative_text))

new_positive_text = positive_text.sample(n=200000, replace=False)
new_negative_text = negative_text.sample(n=200000, replace=False)

new_review_text = new_positive_text.append(new_negative_text)
new_sentiment = np.zeros(200000*2)
for i in range(len(new_sentiment)):
  if i < 200000:
    new_sentiment[i] += 1

c = {"text" : new_review_text,
     "label" : new_sentiment}
large_classification_data = pd.DataFrame(c)
small_classification_data = large_classification_data.sample(400000, replace = False)
## Make Dictionary
reviews = ("\n".join(i for i in small_classification_data['text']))
reviews_split = ''.join([c for c in reviews.lower() if c not in punctuation]).split('\n')
all_text = ' '.join(reviews_split)
words = all_text.split()
counts = Counter(words)
vocab = sorted(counts, key=counts.get, reverse=True)
vocab_to_int = {word: ii for ii, word in enumerate(vocab, 1)}
```

```
## Embedding the words
reviews_ints = []
for review in reviews_split:
    reviews_ints.append([vocab_to_int[word] for word in review.split()])

## Padding
MAX_LEN=100
reviews_ints=pad_sequences(reviews_ints, maxlen=MAX_LEN, dtype="long", truncating="post", padding="post")

## Dividing Training and Testing
train_x, val_x, train_y, val_y = train_test_split(reviews_ints, small_classification_data['label'], random_state=2018,
test_size=0.3)


from sklearn.neural_network import MLPClassifier
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split

clf = MLPClassifier(random_state=1, max_iter=300).fit(train_x, train_y)
scores = cross_val_score(clf, train_x, train_y, cv=5)
svm_predicted = clf.predict(val_x)
print("MLP accuarcy is" ,1 - np.sum((svm_predicted - np.array(val_y))**2) / len(svm_predicted))
print(scores)
from sklearn.model_selection import cross_val_score

from sklearn.metrics import classification_report
stop = time.time()
```

```
3191727 557277
MLP accuarcy is 0.49749166666666667
```

In [124]:
```
from pyspark.ml.classification import MultilayerPerceptronClassifier
import time
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
```

In [125]:
```
start_time=time.time()
path = "s3://caozhoutao/data/libsvm2.txt"
data = spark.read.format("libsvm")\
    .load(path)
```

In [126]:
```
splits = data.randomSplit([0.7, 0.3], 1234)
train = splits[0]
test = splits[1]
```

In [127]:
```
layers = [4, 5, 4, 3]
```

In [128]:
```
trainer = MultilayerPerceptronClassifier(maxIter=100, layers=layers, blockSize=128, seed=1234)
model = trainer.fit(train)
```

In [129]:
```
result = model.transform(test)
predictionAndLabels = result.select("prediction", "label")
evaluator = MulticlassClassificationEvaluator(metricName="accuracy")
stop = time.time()
```

In [130]:
```
print(f"Training time: {stop - start_time}s")
```

Logistic Regression

In [1]:
```
import pyspark as ps
from pyspark.sql import SQLContext

sc = ps.SparkContext('local[4]')
sqlContext = SQLContext(sc)
```

In [2]:
```
import time

from pyspark.ml import Pipeline
from pyspark.ml.feature import CountVectorizer, HashingTF, IDF, StringIndexer, Tokenizer
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.evaluation import BinaryClassificationEvaluator
```

:
```
%%time
# mark the start time
start_time = time.time()

df = sqlContext.read.format('com.databricks.spark.csv').options(header='true', inferschema='true', sep=',').load('./dat
df = df.dropna()
(train_set, test_set) = df.randomSplit([0.7, 0.3], seed=1024)
training_data_size = 10000
original_training_data_size = train_set.count()
original_training_data_size_positive = train_set.filter(train_set.overall == 1).count()
train_set = train_set.sampleBy("overall", fractions={1: 0.5*training_data_size/original_training_data_size_positive,
                                                     0: 0.5*training_data_size/(original_training_data_size-original_t

tokenizer = Tokenizer(inputCol="review_text", outputCol="words")
hashtf = HashingTF(numFeatures=(1<<18), inputCol="words", outputCol='tf')
idf = IDF(inputCol='tf', outputCol="features")
label_stringIdx = StringIndexer(inputCol="overall", outputCol="label")
pipeline = Pipeline(stages=[tokenizer, hashtf, idf, label_stringIdx])

pipelineFit = pipeline.fit(train_set)
train_df = pipelineFit.transform(train_set)
test_df = pipelineFit.transform(test_set)

lr = LogisticRegression(maxIter=100)
lrModel = lr.fit(train_df)
predictions = lrModel.transform(test_df)

evaluator = BinaryClassificationEvaluator(rawPredictionCol="rawPrediction")
auc = evaluator.evaluate(predictions)
accuracy = predictions.filter(predictions.label == predictions.prediction).count() / float(test_set.count())

tp = predictions.filter(predictions.label == 1).filter(predictions.prediction == 1).count()
tn = predictions.filter(predictions.label == 0).filter(predictions.prediction == 0).count()
fp = predictions.filter(predictions.label == 0).filter(predictions.prediction == 1).count()
fn = predictions.filter(predictions.label == 1).filter(predictions.prediction == 0).count()
precision = tp / (tp + fp)
recall = tp / (tp + fn)

end_time = time.time()

print(f"Overall time consumption is {end_time - start_time}")
print(f"Accuracy is {accuracy}")
print(f"Precision is {precision}")
print(f"Recall is {recall}")
print(f"AUC is {auc}")
```

```
Overall time consumption is 63.46057415008545
Accuracy is 0.7896157633881732
Precision is 0.8994167608801238
Recall is 0.8038923848609211
AUC is 0.8422244466837637
CPU times: user 57.1 ms, sys: 12.5 ms, total: 69.6 ms
Wall time: 1min 3s
```

LSTM Model:

```
import torch
import torch.nn as nn
import numpy as np
import pandas as pd
from collections import Counter
import keras
from keras.preprocessing.sequence import pad_sequences
from torch.utils.data import TensorDataset, DataLoader
from string import punctuation
from string import digits
from sklearn.model_selection import train_test_split
import time;
```

```
[ ] import pandas as pd
    large_classification_data = pd.read_csv("CDs_and_Vintl_sentiment.csv")
    large_classification_data.index = range(len(large_classification_data))

    ## drop unmeanningful reviews
    new_large_classification_data=large_classification_data.drop(labels=[1917,43407, 62315,64996,68579,74976,82369,
                                                                         90049,98337,101219,139591,130787,267578,
                                                                         295804,335877,396767,159574,164137,223911,
                                                                         254229,260236,261849,271662,287960,288971,
                                                                         374860,378999,380784,383545],axis=0)
    new_small_classification_data = new_large_classification_data.sample(10000, replace = False)
```

```
## Make Dictionary
reviews = ("\n".join(i for i in new_large_classification_data['text'].astype(str)))
reviews_split = ''.join([c for c in reviews.lower() if c not in punctuation]).split('\n')
all_text = ' '.join(reviews_split)
words = all_text.split()
counts = Counter(words)
vocab = sorted(counts, key=counts.get, reverse=True)
vocab_to_int = {word: ii for ii, word in enumerate(vocab, 1)}

## Embedding the words
reviews_ints = []
for review in reviews_split:
    reviews_ints.append([vocab_to_int[word] for word in review.split()])

## Padding
MAX_LEN=100
reviews_ints=np.array(pad_sequences(reviews_ints, maxlen=MAX_LEN, dtype="long", truncating="post", padding="post"))

## Dividing Training and Testing
train_x, val_x, train_y, val_y = train_test_split(reviews_ints, np.array(new_large_classification_data['label']), random_state=2018, test_size=0.1)
train_data = TensorDataset(torch.from_numpy(train_x), torch.from_numpy(train_y))
valid_data = TensorDataset(torch.from_numpy(val_x), torch.from_numpy(val_y))
batch_size = 100
train_loader = DataLoader(train_data, shuffle=True, batch_size=batch_size)
valid_loader = DataLoader(valid_data, shuffle=True, batch_size=batch_size)

train_on_gpu=torch.cuda.is_available()
```

```
import torch.nn as nn

class SentimentRNN(nn.Module):
    """
    The RNN model that will be used to perform Sentiment analysis.
    """

    def __init__(self, vocab_size, output_size, embedding_dim, hidden_dim, n_layers, drop_prob=0.5):
        """
        Initialize the model by setting up the layers.
        """
        super(SentimentRNN, self).__init__()

        self.output_size = output_size
        self.n_layers = n_layers
        self.hidden_dim = hidden_dim

        # embedding and LSTM layers
        self.embedding = nn.Embedding(vocab_size, embedding_dim)
        self.lstm = nn.LSTM(embedding_dim, hidden_dim, n_layers,
                            dropout=drop_prob, batch_first=True)

        # dropout layer
        self.dropout = nn.Dropout(0.3)

        # linear and sigmoid layers
        self.fc = nn.Linear(hidden_dim, output_size)
        self.sig = nn.Sigmoid()
```

```
    def forward(self, x, hidden):
        """
        Perform a forward pass of our model on some input and hidden state.
        """
        batch_size = x.size(0)

        # embeddings and lstm_out
        x = x.long()
        embeds = self.embedding(x)
        lstm_out, hidden = self.lstm(embeds, hidden)

        # stack up lstm outputs
        lstm_out = lstm_out.contiguous().view(-1, self.hidden_dim)

        # dropout and fully-connected layer
        out = self.dropout(lstm_out)
        out = self.fc(out)
        # sigmoid function
        sig_out = self.sig(out)

        # reshape to be batch_size first
        sig_out = sig_out.view(batch_size, -1)
        sig_out = sig_out[:, -1] # get last batch of labels

        # return last sigmoid output and hidden state
        return sig_out, hidden
```

```
    def forward(self, x, hidden):
        """
        Perform a forward pass of our model on some input and hidden state.
        """
        batch_size = x.size(0)

        # embeddings and lstm_out
        x = x.long()
        embeds = self.embedding(x)
        lstm_out, hidden = self.lstm(embeds, hidden)

        # stack up lstm outputs
        lstm_out = lstm_out.contiguous().view(-1, self.hidden_dim)

        # dropout and fully-connected layer
        out = self.dropout(lstm_out)
        out = self.fc(out)
        # sigmoid function
        sig_out = self.sig(out)

        # reshape to be batch_size first
        sig_out = sig_out.view(batch_size, -1)
        sig_out = sig_out[:, -1] # get last batch of labels

        # return last sigmoid output and hidden state
        return sig_out, hidden
```

```
    def init_hidden(self, batch_size):
        ''' Initializes hidden state '''
        # Create two new tensors with sizes n_layers x batch_size x hidden_dim,
        # initialized to zero, for hidden state and cell state of LSTM
        weight = next(self.parameters()).data

        if (train_on_gpu):
            hidden = (weight.new(self.n_layers, batch_size, self.hidden_dim).zero_().cuda(),
                  weight.new(self.n_layers, batch_size, self.hidden_dim).zero_().cuda())
        else:
            hidden = (weight.new(self.n_layers, batch_size, self.hidden_dim).zero_(),
                      weight.new(self.n_layers, batch_size, self.hidden_dim).zero_())

        return hidden
```

```
[ ] # Instantiate the model w/ hyperparams
    vocab_size = len(vocab_to_int)+1 # +1 for the 0 padding + our word tokens
    output_size = 1
    embedding_dim = 200
    hidden_dim = 128
    n_layers = 3


    net = SentimentRNN(vocab_size, output_size, embedding_dim, hidden_dim, n_layers)
```

```
## Training
lr=0.0001
criterion = nn.BCELoss()
optimizer = torch.optim.Adam(net.parameters(), lr=lr)

epochs = 4
counter = 0
print_every = 100
clip= 500 # gradient clipping
max_step = 20000

if(train_on_gpu):
    net.cuda()

time_start=time.time()
```

```
net.train()
for e in range(epochs):
    h = net.init_hidden(batch_size)
    for inputs, labels in train_loader:
        counter += 1
        if counter > max_step:
            break
        if(train_on_gpu):
            inputs, labels = inputs.cuda(), labels.cuda()
        if counter%20 == 0:
            lr = 0.1
        else:
            lr = 0.005
        if counter>600 & counter%20 != 0:
            lr = 0.0003
        if counter>600 & counter%20 == 0:
            lr = 0.003
        if counter>1200 & counter%20 != 0:
            lr = 0.0001
        if counter>1200 & counter%20 == 0:
            lr = 0.001
        optimizer = torch.optim.Adam(net.parameters(), lr=lr)
        h = tuple([each.data for each in h])
        net.zero_grad()
        try:
            output, h = net(inputs, h)
            loss = criterion(output.squeeze(), labels.float())
            loss.backward()
            nn.utils.clip_grad_norm_(net.parameters(), clip)
            optimizer.step()
        except:
            continue
```

```
        if counter % print_every == 0:
            val_h = net.init_hidden(batch_size)
            val_losses = []
            net.eval()
            for inputs, labels in valid_loader:
                val_h = tuple([each.data for each in val_h])
                if(train_on_gpu):
                    inputs, labels = inputs.cuda(), labels.cuda()
                try:
                  output, val_h = net(inputs, val_h)
                  val_loss = criterion(output.squeeze(), labels.float())
                  val_losses.append(val_loss.item())
                except:
                  continue
            net.train()
            print("Epoch: {}/{}...".format(e+1, epochs),
                  "Step: {}...".format(counter),
                  "Loss: {:.6f}...".format(loss.item()),
                  "Val Loss: {:.6f}".format(np.mean(val_losses)))

time_end=time.time()
print(time_end-time_start)
```

```
# Get test data loss and accuracy
test_losses = []
num_correct = 0
h = net.init_hidden(batch_size)
net.eval()
for inputs, labels in valid_loader:
    h = tuple([each.data for each in h])
    if(train_on_gpu):
        inputs, labels = inputs.cuda(), labels.cuda()
    try:
      output, h = net(inputs, h)
      test_loss = criterion(output.squeeze(), labels.float())
      test_losses.append(test_loss.item())
    except:
      continue
    pred = torch.round(output.squeeze())
    correct_tensor = pred.eq(labels.float().view_as(pred))
    correct = np.squeeze(correct_tensor.numpy()) if not train_on_gpu else np.squeeze(correct_tensor.cpu().numpy())
    num_correct += np.sum(correct)

print("Test loss: {:.3f}".format(np.mean(test_losses)))
test_acc = num_correct/len(valid_loader.dataset)
print("Test accuracy: {:.3f}".format(test_acc))
```

```
Test loss: 0.436
Test accuracy: 0.807
```