Sentiment Analysis with Generative and Discriminative Models

Xiaoxi Celia Lyu[1], **Pingcheng Jian**[2], **Jingyang Zhang**[2]
[1]Department of Economics and CS, [2]Department of ECE
{xl343,pj81,jz288}@duke.edu

Abstract

Sentiment analysis is a classic problem in natural language processing. Researchers have proposed lots of methods in this area, which can be summarized in two classes, generative probabilistic (language) model and (discriminative) neural network. In this paper, we use two kinds of generative probabilistic models and two kinds of discriminative neural networks to solve a sentiment analysis problem on the IMDB review data set. We generate synthetic data using the generative models and compare the prediction accuracy of each model on both real data and the synthetic data. Neural networks show better performance on the real data, while generative models have higher accuracy on synthetic data. We analyze the experiment results and draw a preliminary conclusion on the pros and cons of the two approaches.

1 Introduction

In this project, we leverage machine learning techniques to tackle *sentiment analysis*, one of the most fundamental problems in Natural Language Processing (NLP). In particular, we implement and apply **2** *generative* models (N-gram (Brants et al., 2007) and Multinomial Naive Bayes (Abbas et al., 2019)) and **2** *discriminative* models (LSTM (Hochreiter & Schmidhuber, 1997) and BERT (Devlin et al., 2018)) to the IMDB movie review dataset (Maas et al., 2011). We carefully train these models and find that they all can achieve non-trivial accuracy (i.e., noticeably higher than the random-guessing level) on this task, while certain methods outperform the others. We also utilize the trained generative models to synthesize artificial data and conduct evaluation beyond empirically observed/real data.

The report is organized as follows. In Section 2 we first formulate sentiment analysis as a machine learning problem and introduces the used dataset. Section 3 and section 4 will describe the generative and discriminative models, respectively, and present experimental results. A thorough discussion of the methods and results will also be provided as we move forward. We thoroughly compare all methods and discuss their pros and cons in Section 5.

2 Problem Statement

Sentiment analysis as a machine learning problem. In many cases, the sentences we say come with a certain sentiment or attitude such as happy, sad, or angry. For example, the person who left a review of "Friendly staff, great service" was most likely to be happy or satisfied at the time. In the task of sentiment analysis, obviously, the goal is to identify the exact sentiment that is associated with each sentence.

Since there is probably an infinite number of sentiments (depending on the granularity), to make this task tractable, one often formulates it as a classification problem: Given a sentence, we seek to classify the underlying sentiment into one of the pre-defined categories. Accordingly, the performance metric would be classification accuracy (assuming there is no class imbalance).

IMDB dataset. In this work, we focus on the popular IMDB movie review dataset for sentiment analysis. The dataset consists of highly polar movie reviews, where people express either a *positive* or *negative* attitude towards the movie. Therefore it is essentially a binary classification problem. We

Table 1: Sample reviews and their sentiment from the IMDB dataset.

review	**sentiment**
If you like original gut wrenching laughter you will like this movie. If you are young or old then you will love this movie, hell even my mom liked it. Great Camp!!!	positive
It's terrific when a funny movie doesn't make smile you. What a pity!! This film is very boring and so long. It's simply painfull. The story is staggering without goal and no fun. You feel better when it's finished.	negative

show in Table 1 two sample reviews. In total there are 50,000 reviews in the dataset, from which we partition 40,000 reviews as training data and 10,000 reviews as test data (with a fixed random seed). In both the training and test split, there are equal amounts of positive and negative reviews (i.e., they are class-balanced), thus we directly use classification accuracy as performance measurement.

3 Generative Models

3.1 Multinomial Naive Bayes

General concept. Naive Bayes[1] is a family of conditional probability models Rish et al. (2001), which include multinomial naive Bayes, Bernoulli Bayes, Gaussian Bayes, etc. The "naive" assumption means that every pair of a feature is conditional independent. Using Bayes theorem, we calculate a probability $P(c|x)$, where c is the possible classes (in this problem, is 0 for negative and 1 for positive) and x is the feature.

According to Bayes theorem Berrar (2018), we decompose the conditional probability as:

$$p\left(C_k \mid \mathbf{x}\right)=\frac{p\left(C_k\right) p\left(\mathbf{x} \mid C_k\right)}{p(\mathbf{x})}. \tag{1}$$

In the application of sentiment analysis in NLP Abbas et al. (2019), we have a sentence of review in which each word can be considered as a feature. Therefore, to calculate the probability of the sentiment according to every word in a sentence, we are actually calculating $p\left(C_k \mid x_1, \ldots, x_n\right)$. To this end, we can repeatedly calculate the conditional probability according to the chain rules Grover (2016):

$$\begin{aligned}
p\left(C_k, x_1, \ldots, x_n\right) & =p\left(x_1, \ldots, x_n, C_k\right) \\
& =p\left(x_1 \mid x_2, \ldots, x_n, C_k\right) p\left(x_2, \ldots, x_n, C_k\right) \\
& =p\left(x_1 \mid x_2, \ldots, x_n, C_k\right) p\left(x_2 \mid x_3, \ldots, x_n, C_k\right) p\left(x_3, \ldots, x_n, C_k\right) \\
& =\cdots \\
& =p\left(x_1 \mid x_2, \ldots, x_n, C_k\right) p\left(x_2 \mid x_3, \ldots, x_n, C_k\right) \cdots p\left(x_{n-1} \mid x_n, C_k\right) p\left(x_n \mid C_k\right) p\left(C_k\right).
\end{aligned} \tag{2}$$

According to our naive assumption, we have:

$$p\left(x_i \mid x_{i+1}, \ldots, x_n, C_k\right)=p\left(x_i \mid C_k\right). \tag{3}$$

Applying the naive assumption 3 to the chain rule equation 2, we have:

$$\begin{aligned}
p\left(C_k \mid x_1, \ldots, x_n\right) & \propto p\left(C_k, x_1, \ldots, x_n\right) \\
& \propto p\left(C_k\right) p\left(x_1 \mid C_k\right) p\left(x_2 \mid C_k\right) p\left(x_3 \mid C_k\right) \cdots \\
& \propto p\left(C_k\right) \prod_{i=1}^n p\left(x_i \mid C_k\right).
\end{aligned} \tag{4}$$

The Multinomial Naive Bayes assumes that the target classes are subject to the multinomial distribution. To be specific, there are only two classes (positive and negative) in our problem, so they are subject to the binomial distribution.

[1]We refer to Naive Bayes classifier wikipedia for the math equations.

Bag-of-words model. We use the bag-of-words model Zhang et al. (2010) for the data preprocessing. This model represents the text as a multiset ("bag") of its words. It disregards the ordering of the words but keeps the multiplicity of them. This method is widely used in natural language processing and has been proven to be simple but effective.

Training setup. Before training the Multinomial Naive Bayes model, we first remove the stop words in the reviews. These stop words are generated by the `nltk.corpus.stopwords` package. Then we use the bag-of-word model to turn these texts into vectors and store them in a sparse matrix. We use `sklearn.naive_bayes.MultinomialNB` class for the Multinomial Naive Bayes model. We call the "fit" function for training this model, and the "predict" function for testing the trained model.

Synthetic data. We also sample from the trained binomial distribution and generate some synthetic review data. Concretely, we generate 100 items of positive reviews and 100 items of negative reviews (we do not generate more samples due to computation constraints). We find that the generated sentences from the Multinomial Naive Bayes are not readable sometimes.

Results. Table 2 shows the prediction accuracy of the Multinomial Naive Bayes model. We test it on the test set of the real IMDB data and the entire synthetic data generated by itself. This model achieves 74.5% accuracy on the real data. This is not a very high success rate, but it is reasonable considering the Multinomial Naive Bayes model is a very simple and naive model. It achieves an accuracy of 83.0% on the synthetic data, which is higher than that on the real data. This is because we test a model on the synthetic data generated by itself; it is supposed to have good performance.

Table 2: The prediction accuracy of the Multinomial Naive Bayes model.

data type	**accuracy**
real	74.5%
synthetic	83.0%

We also analyze the performance of the synthetic data in detail. We find that it achieves 95% accuracy on the positive synthetic data, but only 71% accuracy on negative data. It is our hypothesis that humans use more complicated ways to express their negative sentiment (e.g. irony) than the positive sentiment (usually have obvious mark words like "good"), and therefore the simple Multinomial Naive Bayes model tends to have better performance on the positive data than that on the negative data.

3.2 N-GRAM

General concept. N-gram language model is a widely-used generative model, which is "one type of language model used to predict upcoming words or the corresponding probability of upcoming words" (Jurafsky & Martin, 2009). Suppose $P(w|h)$ is the probability that we want to compute, where w is the upcoming word, and h is the entire history given. According to the Bayes theorem, we can compute the probability of the entire history h:

$$P(h) = \frac{P(hw)}{P(w|h)}. \tag{5}$$

Let's rewrite history h as $w_1, w_2, ..., w_n$, where w_i is the i_{th} word in the given corpus. To compute the probability of the entire history $w_1, w_2, ..., w_n$, chain rule of probability can be used to decompose this probability:

$$P(w_{1:n}) = P(w_1)P(w_2|w_1)P(w_3|w_{1:2})...P(w_n|w_{1:n-1}) = \prod_{k=1}^{n} P(w_k|w_{1:k-1}). \tag{6}$$

In reality, instead of taking the entire history into account, we can approximate the history in a more efficient way using the last N words. For example, $N = 1$ means unigrams, $N = 2$ means bigrams, and $N = 3$ means trigrams. This is the basic intuition of N-gram language model:

$$P(w_n|w_{1:n-1}) \approx P(w_n|w_{n-N+1:n-1}). \tag{7}$$

To estimate this probability, maximum likelihood estimation (MLE) can be used. Specifically, to compute $P(w_n|w_{n-N+1:n-1})$, we can count the gram $w_{n-N+1:n}$ in the entire corpus, and normalize it by the sum of all the grams that share the same sequence of words $w_{n-N+1:n-1}$. Hence, we can estimates the N-gram probability as follows, where C is the count of a corresponding gram:

$$P(w_n|w_{n-N+1:n-1}) = \frac{C(w_{n-N+1:n-1}w_n)}{C(w_{n-N+1:n-1})}. \tag{8}$$

Even in a very large corpus, some N-grams may have counts of zero. This is what is called the **problem of sparsity**. Specifically, the N-gram model will estimate the probability of such N-grams as 0, which means such N-grams will never occur in predictions. However, this is obviously incorrect as all N-grams should be possible to occur in predictions. In general, to deal with the problem of sparsity, there are three possible ways - smoothing, back-off, and interpolation. After doing experiments, we found that back-off has the best prediction performance on our training set. Therefore, we will mainly introduce how stupid backoff is applied to our training set in this project.

Stupid backoff. In backoff, if the N-gram has a count of zero, we can estimate the conditional probability of the upcoming word in the N-gram by backing off to the (N-1)-gram model. If the (N-1)-gram still has a count of zero, we continue backing off to a lower-order model until the corresponding gram has a non-zero count. For simplicity, we use stupid backoff rather than a more sophisticated method (e.g., Katz backoff) in our project. We follow the stupid backoff algorithm introduced in (Brants et al., 2007):

$$S(w_i|w_{i-N+1:i-1}) = \left\{ \begin{array}{ll} \frac{\text{count}(w_{i-N+1:i})}{\text{count}(w_{i-N+1:i-1})}, & \text{count}(w_{i-N+1:i}) > 0 \\ \lambda \cdot S(\text{count}(w_i|w_{i-N+2:i-1})), & \text{otherwise} \end{array} \right. . \tag{9}$$

The backoff will terminate when $N = 1$, which has probability $S(w) = \frac{count(w)}{N}$. The fixed weight $\lambda = 1$ is context-independent. In this project, we set $\lambda = 1$.

Training setup. Before training the N-gram model, we first add an *end of sentence word (eos)* at the end of each sentence, to help know when the sentence should end when generating new sentences. After that, we remove the unwanted characters in the reviews, including line break elements, periods, commas, exclamation points, ellipses, parentheses, and colons. Then we use the `CountVectorizer` package to convert the given corpus into a matrix, which stores counts of all N-grams. Specifically, we set `min_df` as 5, which is the lower cut-off when building the vocabulary. We set the `ngram_range` as $1, 2, 3, 4, 5$ respectively, for N-gram models with different N values correspondingly. For training this model, we call the "fit" function to extract the fitted vocabulary, and then we call the "transform" function to extract N-gram counts out of raw documents using the fitted vocabulary. Now we have the frequency dictionary of each review.
To do sentiment analysis, we will use a combined model consisting of N-gram and logistic regression. Noted that a single logistic regression is a discriminative model since it calculates the posterior $P(w|h)$ directly rather than estimating $P(w|h)$ using Bayes theorem. However, we use a combination of N-gram and logistic regression in our project, which is a generative model essentially. To make it more clear, this combined model has characteristics of generative models including 1) it can generate new reviews, 2) it captures the joint probability $P(hw)$ and $P(h)$ if the data is unlabelled, 3) it is not only learns the distribution of the given data but also tells the probability of a given example. In this project, we use the `LogisticRegression` binary classifier to take the frequency dictionary and the label of each review as input and go through every review in the train set. For training the logistic regression model, we call the "fit" function, and we call the "predict" function for testing the trained model.

Synthetic data. To avoid plagiarism, we cite the reference codes here[2]. To generate synthetic review data, we use stupid backoff to deal with zero-probability N-grams, where we choose $N = 3$. The generation of each review terminates if the length of the sentence exceeds 100 or an *end of sentence word (eos)* is generated. Concretely, we generate 500 items of positive reviews and 500 items of negative reviews. Note that the N-gram model often generates nonsensical or syntactically incorrect sentences, which makes sense as the given corpus is not large enough.

Results. Table 3 shows that when N=1 or 2 or 3, the prediction accuracy of the logistic regression model is pretty high, greater than 85.0%. We choose $N = 3$ to generate synthetic data using N-gram in this project since qualitatively it generates more reasonable results. Table 4 shows the prediction accuracy of the N-gram model. We test it on the test set of the real IMDB data and the entire

[2] We refer to the GitHub codes for backoff in N-gram.

synthetic data generated by itself. This model achieves 85.0% accuracy on the test set of the real data, which is a pretty high success rate. Interestingly, despite that the synthetic data doesn't make much sense to human readers, the N-gram model is able to achieve an accuracy of 83.7%, which is close to that of the real data. We think that this is because we are applying the model to its own generated data.

Table 3: The prediction accuracy of the N-gram model with different N.

value of N	**accuracy**
1	88.7%
2	88.8%
3	85.0%
4	78.4%
5	68.7%

Table 4: The prediction accuracy of the N-gram model.

data type	**accuracy**
real	85.0%
synthetic	83.7%

4 Discriminative Models

4.1 LSTM

General concept. Long Short-Term Memory (LSTM) is one type of Recurrent Neural Networks (RNNs). Specifically, for each element in the input sequence, LSTM computes the following function[3] (assuming a single layer):

$$\begin{aligned}
i_t &= \sigma(W_{ii}x_t + b_{ii} + W_{hi}h_{t-1} + b_{hi}),\\
f_t &= \sigma(W_{if}x_t + b_{if} + W_{hf}h_{t-1} + b_{hf}),\\
g_t &= \tanh(W_{ig}x_t + b_{ig} + W_{hg}h_{t-1} + b_{hg}),\\
o_t &= \sigma(W_{io}x_t + b_{io} + W_{ho}h_{t-1} + b_{ho}),\\
c_t &= f_t \odot c_{t-1} + i_t \odot g_t,\\
h_t &= o_t \odot \tanh(c_t),
\end{aligned} \tag{10}$$

where x_t is the input at time t, h_t and c_t are hidden and cell state, respectively, i_t, f_t, g_t, o_t are the input, forget, cell, and output gates, respectively. σ is the sigmoid function, and $\odot$ is the Hadamard product. One can see that LSTM explicitly has multiple "gates" controlling how the current hidden state h_t is affected by the current input x_t and past hidden state h_{t-1}. Intuitively, this process simulates the memory system in our human brains: We often choose to memorize more important things and forget those that are trivial.

LSTM classifier.

We build an LSTM classifier for sentiment analysis with the following three components (Figure 1):

- an embedding layer which transforms the numerical token index to a vector/embedding;
- a LSTM module which takes the embeddings sequentially and outputs the final hidden state as the representation;
- a fully-connected/linear layer that takes the extracted representation and outputs class logits.

Note that to mitigate overfitting, during training we apply dropout within the LSTM (when there are multiple layers) and between LSTM and the linear layer. The final prediction is determined by taking the argmax over the class logits.

[3]We refer to PyTorch official documentation.

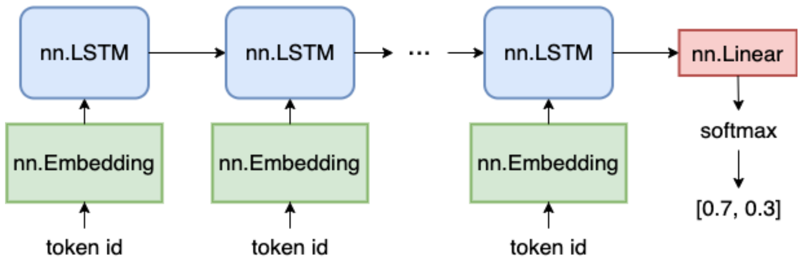

Figure 1: The diagram of the LSTM classifier we built for sentiment analysis. At the end the classifier will output $p(y = \text{positive}|x)$ and $p(y = \text{negative}|x)$.

Training setup. To train this classifier, we minimize the cross-entropy loss between the predicted and ground-truth distribution. We choose to use the RMSprop optimizer (Graves, 2013), as we empirically find that standard optimizer like SGD does not perform well on RNN models. We use a learning rate of 0.001 and keep the batch size at 96. The input sequences are truncated if they exceed the maximal length of 256. We vary other hyperparameters such as the embedding dimension, weight decay, and dropout rate in our experiments. The training is performed using the PyTorch framework (Paszke et al., 2019).

Table 5: Results of the LSTM classifier. We explore various hyperparameters to achieve higher accuracy.

emb. dim	hidden dim	# layers	bidirectional	# params	dropout	weight decay	# epochs	test acc
1	100	1	False	360,318	0	0	5	0.874
1	100	1	False	360,318	0	0	10	0.833
5	50	1	False	1,606,082	0	0	5	0.873
1	100	1	False	360,318	0.1	0	5	0.876
1	100	1	False	360,318	0.2	0	5	0.885
1	100	1	False	360,318	0.3	0	5	0.888
1	100	1	False	360,318	0	5e-5	5	0.882
1	100	1	False	360,318	0	1e-4	5	**0.891**
1	100	1	False	360,318	0	5e-4	5	0.733
1	100	1	False	360,318	0.1	1e-4	5	0.888
1	100	1	False	360,318	0.3	1e-4	5	0.887
1	100	2	False	441,118	0	1e-4	5	0.879
1	100	1	True	401,518	0	1e-4	5	0.881

Results are summarized in Table 5. The best test accuracy we achieve is **89.1%**, which is kind of expected since deep learning models can be really powerful given abundant training data. There are some additional observations from Table 5:

- Training the model for 5 epochs is sufficient for convergence on this dataset; increasing the training epochs does *not* help with the accuracy.
- Increasing the model capacity (in terms of the number of parameters) does *not* necessarily improve the performance; this might be due to overfitting.
- Incorporating regularization techniques such as dropout and weight decay (ℓ_2 regularization) helps the model generalize better. Combining them together, however, does not yield further improvements.
- Having a bidirectional LSTM gives limited benefit over a unidirectional model.

We also qualitatively measure the performance by looking at the successful and failure cases of this model in Table 6. For the two successful/correct cases, there are words with a clear attitude like "good", "nice" for positive, and "bad" for negative. For the failure case in the third row of Table 6, although there are positive words like "beautiful" and "stunning" in the beginning, things could be confusing with the last sentence "It isn't a movie for anyone who wants normality". We suspect that the last sentence is what causes the model to determine it as a negative review. In the other failure case, the review does not have very strong words with negative feelings, which potentially results in the wrong prediction.

Table 6: Examining the LSTM classifier predictions on a few samples.

review	**label**	**prediction**
I found it real shocking at first to see William Shakespeare's love masterpiece reworked into a gory, violent and kinky sensual movie adaptation. But after you watched it once, it sort of grows on you when you watch it the second and third times, as you come over the shock and start appreciating the movie on its own merits - solid acting, good dialogue, nice sequencing and choreography, not-too-bad soundtrack and some of the (special) effects that go on. Oh, and also the ending. What a riot!	positive	positive
The beginning of this movie had me doubting that it would be little more thana typical B sci-fi flick. But, as it progressed I began to get interested and I saw the whole thing through. The premise is interesting, original, and has the makings of making a classic. Alas, it instead ended up a mediocre movie, done in by the usual factors which turn a potentially good movie into a bad movie (bad acting, low budget etc.). I'm interested to see how this would turn out if it were remade with good actors and a big hollywood budget.	negative	negative
This movie is beautiful in all ways. It is visually stunning, and this is a good thing since the dialogue would only take up a page or two of paper. The acting is superb; it is subtle, passionate and intense. Ben Daniels does a fabulous job of turning himself into an animal, and mixing that wild nature with a man's overbearing passion and honor. There is not one flaw, not one mistake or wrong moment to be found anywhere. It is completely perfect, but only if you understand what you're going to experience. It isn't a movie for anyone who wants normality.	positive	negative
Cheesy script, cheesy one-liners. Timothy Hutton's performance a "little" over the top. David Duchovny still seemed to be stuck in his Fox Mulder mode. No chemistry with his large-lipped female co-star. He needs Gillian Anderson to shine. He does not seem to have any talent of his own.	negative	positive

Classifying synthetic data. The LSTM classifier achieves 79.7% and 35.0% accuracy on the sythetic data from the N-gram and Multinomial Naive Bayes model, respectively. Our hypothesis is that since the nature of N-gram is predicting the next word, its generated sentences could be more natural than the ones from the Bayes model; this explains why the accuracy of N-gram's data is a lot higher.

4.2 BERT

General concept. Transformer (Vaswani et al., 2017) has become the dominant model for NLP since its emergence. The essence of the transformer is the self-attention mechanism, which we do not delve into details since it is already covered in the lectures. Devlin et al. (2018) proposed **B**idirectional **E**ncoder **R**epresentations from **T**ransformers (BERT), which is a powerful pre-training methodology for transformers. Specifically, BERT aims to learn rich bidirectional representations by using a "masked language model" (MLM) pre-training objective. The MLM will first randomly mask some tokens within the input sequence, and the goal is to predict/recover the original token id of the masked word based only on its context. In addition, BERT also uses a "next sentence prediction" task to jointly pre-train text-pair representations. Empirically, BERT is shown to benefit a wide range of downstream tasks with its pre-training objective.

BERT classifier. The conceptual diagram of the classifier is shown in Figure 2. Although not depicted in the figure, we again use dropout between the transformer and the linear layer to prevent overfitting. Compared with the LSTM classifier in Figure 1, there are a few differences:

- Unlike LSTM that takes as input the tokens one by one from the sequence, the transformer directly accepts the whole sequence as input.
- For the LSTM classifier, we use our own tokenizer (similar to the N-gram homework). The transformer, however, comes with its own tokenizer.
- Similarly, transformers have their in-built embedding layer so we don't have to define it by ourselves.

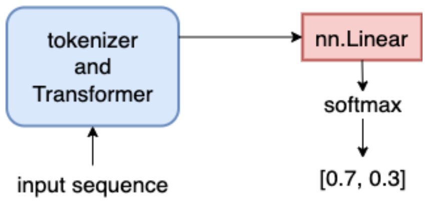

Figure 2: The diagram of the BERT classifier we built for sentiment analysis. At the end the classifier will output $p(y = \text{positive}|x)$ and $p(y = \text{negative}|x)$.

Training setup. We fine-tune the pre-trained BERT model taken from the `pytorch_pretrained_bert` library. Again we use cross-entropy as the loss function but this time with Adam (Kingma & Ba, 2014) as the optimizer. The learning rate is 3e-6; the batch size is 32 so that we could fit the model into one GPU. To ensure a fair comparison with the LSTM experiments, we perform fine-tuning for 5 epochs, and the maximal input length is still 256.

Results. Here we are interested in how BERT's pre-training can benefit our sentiment analysis task. To this end, we vary the percentage of training data that BERT sees during the fine-tuning. One can see from Figure 3 that by fine-tuning only 10% data, BERT already achieves an accuracy of 89.0% which is on par with the LSTM model that is trained on 100% data. The performance further goes up as we increase the number of training samples, and in the end, we achieve **92.1%** accuracy. These results all demonstrate that with BERT pre-training, transformers are indeed a stronger model than LSTM.

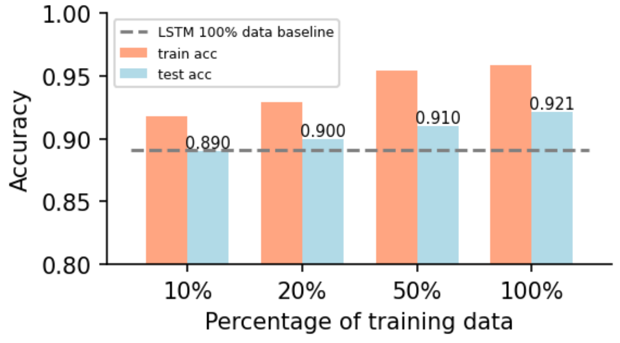

Figure 3: Results of BERT classifier experiments. By fine-tuning on only 10% training data, BERT can already match the LSTM's performance which is trained on 100% data.

Lastly, we again qualitatively check the performance by looking at some predictions made by the model. First, we find that BERT shares many predictions with the LSTM model. Second, even though BERT achieves >90% accuracy, it still fails on some simple cases like the ones in Table 7, which all have strong negative words, but BERT predicts them as "positive". In all, both LSTM and BERT models lack interpretability in their decisions.

Classifying synthetic data. The BERT classifier achieves 86.7% and 32.5% accuracy on the sythetic data from the N-gram and Multinomial Naive Bayes model, respectively. The pattern is similar to that observed with LSTM.

5 DISCUSSION

In this section, we will evaluate the results qualitatively and quantitatively at first.

Quantitative evaluation. We have a preliminary finding according to table 8 which shows the quantitative comparison of all methods.
Both of the two generative probabilistic models have higher or similar prediction accuracy on the

Table 7: Examining the BERT classifier predictions on a few samples. BERT shares some predictions with those made by the LSTM classifier in Table 6.

review	**label**	**prediction**
Sadly it was misguided. This movie stunk from start to finish. It was hard to watch because I used to watch Clarissa Explains It All every day on Nickelodeon. I LOVED her. Then the next thing I found she did a spread in Maxim and she was gorgeous! I haven't really heard anything about her until I watched this movie on accident. I couldn't believe she would even let something like this be seen with her name all over it. Everything about it was wrong but it still looked like someone somewhere in the team was trying really really hard to save a sunk ship. Too bad. I hope she continues to act and I would love to see her with a real cast in a real movie.	negative	positive
This was among the STUPIDEST and PREACHIEST of the anti-nuke films out of the 1980s. The idea that a kid and a basketball star could "change the world" is pretty far-fetched, given how many "children's peace marches" and "celebrity protests" there were and ARE. But the idea that the Soviet Union would agree to a TOTAL nuclear disarmament, because some apparatchik kids learned of a "silent protest" in the West, is ludicrous. What ended the Cold War? America's tough, dare I say "Reaganesqu" stance and the internal failures of socialism. It was NOT the peace marches, the "die-ins" or films like "Amazing Grace & Chuck", "Miracle Mile", or "Testament".	negative	positive

entire synthetic dataset than the test set of the real dataset, and both of the two discriminative neural networks have higher prediction accuracy on the test set of the real dataset than the entire synthetic dataset. Meanwhile, both of the two discriminative neural networks have much higher prediction accuracy on the synthetic dataset generated from the N-gram model than the Multinomial Naive Bayes model.

Qualitative evaluation. It is straightforward that generative models have better or at least similar prediction performance on their own synthetic dataset than discriminative models, and discriminative models have better prediction performance on the real dataset than generative models.
For one reason, the frequency dictionary of generative models for training model and making prediction are exactly the same, while the vectorization of discriminative neural networks for training model and making predictions are different. For another reason, generative models often have poorer generalization performance than discriminative models due to the discrepancy between the model and the true distribution of the data (Bishop & Lasserre, 2007). In general, when using real data which is often much more plentiful than synthetic data, discriminative models can give better generalization performance.

Next, we will discuss the pros and cons of each method in terms of 1) quality/correctness, 2) data, time, and computational requirements, and 3) interpretability.

Quality/Correctness. We show in Table 8 the comparison of classification accuracy of all methods. Clearly, the data-driven discriminative neural networks outperform generative models thanks to their great representation capacity. The less desirable accuracy of generative models may limit their utility in real-world applications. Especially when the data is plentiful, excellent generalization performance of discriminative models will outperform generative models.

Table 8: Classification accuracy of all methods on each type of data.

method type	method	data type	accuracy
generative	Multinomial Naive Bayes	real	74.5%
		synthetic	83.0%
	N-gram	real	85.0%
		synthetic	83.7%
discriminative	LSTM	real	89.1%
		N-gram synthetic	79.7%
		Bayes synthetic	35.0%
	BERT	real	92.1%
		N-gram synthetic	86.7%
		Bayes synthetic	32.5%

Data, time, computational requirements. Deep neural networks (DNNs) are known to be data-hungry. For example, BERT pre-trains on 3,300 million text passages (Devlin et al., 2018). When there is less abundant training data, neural networks are prone to overfitting and can yield much lower test accuracy. In comparison, we suspect that generative models are less sensitive to data amount. Ideally, this hypothesis could be verified by repeating the experiments in Figure 3 on generative models. Meanwhile, DNNs are also much more computationally intensive and time-consuming (in both training and inference) due to their extremely large number of parameters. For example, training the N-gram model only takes a few seconds with CPUs on a standard laptop, while training the BERT model could take several hours with a high-performance GPU on a server. Moreover, when the data is unlabelled or very expensive to be labelled, we can not use discriminative models anymore. Discriminative models can only be used for supervised learning problems, while generative models can work for both supervised and unsupervised learning problems. The goal of discriminative models is to estimate the conditional probability $P(w|h)$, while the goal of generative models is to approximate $P(w)$ and $P(h|w)$ in an unsupervised setting, then calculate $P(w|h)$ in a supervised setting.

Interpretability. Current DNN models lack certain interpretability in their decisions; they operate more like a black-box model which does not explain or cannot give reasoning as to why they give a prediction and how uncertain they are. In the meantime, we do not find generative models to be more interpretable. For example, in our project, the Multinomial Naive Bayes aims to model the probability of the sequence given the sentiment label. However, it is still hard to interpret why certain sequences are assigned higher probability than others. Other models such as decision trees could add more interpretability. In addition, generative models learn the probability distribution of the training data, to have a richer insight into the data's underlying characteristics. In contrast, discriminative models have no idea about how the data was generated.

References

Muhammad Abbas, K Ali Memon, A Aleem Jamali, Saleemullah Memon, and Anees Ahmed. Multinomial naive bayes classification model for sentiment analysis. *IJCSNS Int. J. Comput. Sci. Netw. Secur*, 19(3):62, 2019.

Daniel Berrar. Bayes' theorem and naive bayes classifier. *Encyclopedia of Bioinformatics and Computational Biology: ABC of Bioinformatics*, 403, 2018.

Christopher Bishop and Julia Lasserre. Generative or discriminative? getting the best of both worlds. *Bayesian Statistics*, 8:3–23, January 2007. URL `https://www.microsoft.com/en-us/research/publication/generative-discriminative-getting-best-worlds/`.

Thorsten Brants, Ashok C. Popat, Peng Xu, Franz J. Och, and Jeffrey Dean. Large language models in machine translation. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pp. 858–867, Prague, Czech Republic, June 2007. Association for Computational Linguistics. URL `https://aclanthology.org/D07-1090`.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

Jeff Grover. Statistical properties of bayes' theorem. In *The Manual of Strategic Economic Decision Making*, pp. 37–57. Springer, 2016.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

Daniel Jurafsky and James H. Martin. *Speech and Language Processing (2nd Edition)*. Prentice-Hall, Inc., USA, 2009. ISBN 0131873210.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Andrew Maas, Raymond E Daly, Peter T Pham, Dan Huang, Andrew Y Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies*, pp. 142–150, 2011.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

Irina Rish et al. An empirical study of the naive bayes classifier. In *IJCAI 2001 workshop on empirical methods in artificial intelligence*, volume 3, pp. 41–46, 2001.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Yin Zhang, Rong Jin, and Zhi-Hua Zhou. Understanding bag-of-words model: a statistical framework. *International journal of machine learning and cybernetics*, 1(1):43–52, 2010.