Information Theory in Natural Language Processing Classification

ECE867 Final Report[1]

Lyuxun Yang

Abstract

A recent study [5] revealed a two-phase pattern in a DNN learning experiment using SGD algorithm on simulated data, but it is unclear what the pattern is in other situations. This project aims at observing the information during the learning processes of different models, using different solving algorithms, on a real-world NLP classification data set. It first presented the information measures in training data, feature engineering, and Information Bottleneck. Then, it analyzed the learning patterns of MaxEnt, MLP Network, and AdaBoost. The results show that the two-phase pattern exists in various learning processes, while details of the graphs vary on different algorithms. They support the conjecture that the two-phase pattern generally exists, and is related to the performance (being fast or likely to converge) of algorithms and models.

Index Terms

Information Theory, Natural Language Processing, Classification, Machine Learning

I. Introduction

Machine learning is a field of computer science that gives computers the ability to learn without being explicitly programmed [1]. In other words, machine learning models can derive information from data automatically. Thus, Information Theory has been applied in many aspects of machine learning, such as loss functions and precision measures [2], Information Theoretic Clustering [3], and Entropy Ensemble Filter for Bagging method [4].

In a recent research [5], Shwartz-Ziv and Tishby used entropy and mutual information to monitor the learning process of Deep Neural Network (DNN), and found that the DNN learning has two distinct phases: empirical error minimization and representation compression. This pattern is significant for revealing the principles behind neural network, which has been thought to be a “black box” for a long time.

However, the observation in [5] is based on the experiments where Stochastic Gradient Decent (SGD) is used to solve a DNN, using very balanced simulated data. It is unclear whether other machine learning models, or other solving algorithms, or real-world data, can lead to the same conclusion. So in this project, I studied on different kinds of machine learning models, with different solving algorithms, to learn from a real-world Natural Language Processing (NLP) classification data. The object is to understand and explain the machine learning process by monitoring the information measures.

This paper will first introduce the concepts used in the experiments, including entropy, mutual information and the Information Bottleneck. Then, feature engineering will be explained from the perspective of entropy. Finally, it will include 3 different machine learning models, and the results of their learning graphs observed in my experiments.

The first model is Maximum Entropy Model (MaxEnt). It is based on the principle of maximum entropy, which states that the probability distribution which best represents the current state of knowledge is the one with largest entropy [6]. The main idea is to consider all models that fit the training data, and select the one which has the largest entropy. It has been successfully applied to NLP and many other areas. The second model, Multi-layer Perceptron Neural Network (MLP NN), is very similar the DNN in [5].

[1]This paper uses the IEEE Conference LaTeX Template “IEEEtran.cls”.
URL: `https://www.ieee.org/conferences_events/conferences/publishing/templates.html`

It is an artificial neural network with multiple hidden layers between the input and output layers [7]. I'll use 3 different algorithms to solve the model, and compare their information graphs. The last model is Adaptive Boosting (AdaBoost), which belongs to ensemble models. In an AdaBoost model, multiple weak classifiers are combined into a weighted sum, so that the model can achieve "the best out-of-the-box classifier" [8].

The data I used for experiments are from the Google's Text Normalization Challenge [9]. It is a typical NLP problem. The goal of the challenge contains two parts, classifying the words and convert them to spoken forms. In this project I only focus on the classification.

II. THEORIES

A. Information in training data

Consider the classical setting of supervised learning. Let $\bar{X} = (x_1, x_2, \cdots, x_n)'$ be the input data set, which is a $n \times p$ matrix containing n training sample vectors. Each sample x_i has m elements. In NLP settings, x_i can also refer to the feature vector, for example, the counts of a specific word, character, or sequence. In this paper, original data and features are separated, especially when talking about the feature engineering.

A sample vector x_i is a vector of random variables. Since usually the data are sampled from the same population, it is assumed that $x_1, x_2, \cdots, x_n$ are i.i.d. Let X be the random variable of the distribution. Similarly, the output data set (label set) is $\bar{Y} = (y_1, y_2, \cdots, y_n)$, and $y_1, y_2, \cdots, y_n$ have the same distribution. Let Y be the random variable of the distribution.

In NLP settings, X is usually discrete, because it is often texts, integers, or booleans. In classification settings, Y is also discrete, as one of the classes. Then the entropy of data and the entropy of label set are:

$$H(X) = -\sum_j p(x_j)\log(p(x_j)), H(Y) = -\sum_j p(y_j)\log(p(y_j)) \tag{1}$$

$$H(X,Y) = -\sum_j p(x_j, y_j)\log(p(x_j, y_j)) \tag{2}$$

where x_j, y_j are possible values that X, Y can take.

The purpose of supervised learning is to let the model learn the relationship between X and Y, in other words, to get as much mutual information of X, Y as possible. The Mutual Information is:

$$I(X;Y) = H(X) - H(X|Y) \tag{3}$$

It implies that, the model needs the information in input data X, but does not need the part of X that is not related to the labels. This idea will be explained more in following parts.

A problem is, estimating an entropy is not easy in NLP settings. Actually, in any discrete setting, "no unbiased estimator for entropy or mutual information exists." [14] There are some low bias or low variance estimators, but they often rely on assumptions of the distribution. In NLP, the text data should belong to a multinomial distribution containing a huge number of classes. Some texts have higher possibilities than others, but it can be hardly estimated without a big dataset of the whole language. Therefore, it is almost impossible to only use limited data set to estimate the entropy of the population.

But fortunately, we don't need to get a specific number of the entropy of the population. In this project, we only care about how the entropy change and related to each other. Therefore, the data set will be directly treated as the population, and frequency will be used to represent probability. This will cause inaccuracy, but it will work well in comparison. For example, the entropy in a decision tree model is also calculated directly from the available data, but the comparison results is effective for building the tree.

B. Feature Engineering

Feature engineering is the process of using domain knowledge of the data to create features that make machine learning algorithms work better. It is so important that Andrew Ng said "applied machine learning is basically feature engineering." [12]

Feature extractor is a function f that can convert a sample vactor x_i to a feature vector $x_i^F = f(x_i)$. So $f : X \to X^F$, where X^F is the feature random vector determined by X. Therefore,

$$H(X) \geq H(X^F) = I(X; X^F) \tag{4}$$

It implies, feature engineering can never enlarge information in the data. Then why can it help the learning? In my understanding, a good feature engineering is like a compression on the dataset. It doesn't produce information, but increase the percentage of useful information, so that:

$$\frac{I(X^F;Y)}{H(X^F)} > \frac{I(X;Y)}{H(X)} \Rightarrow \frac{I(X^F;Y)}{I(X;Y)} > \frac{H(X^F)}{H(X)} = \frac{I(X^F;X)}{H(X)} \tag{5}$$

The left equation implies that the percentage of useful information in features are larger than it of original data. In the right equation, $\frac{I(X^F;Y)}{I(X;Y)}$ means the ratio of information extracted from the useful information, while $\frac{H(X^F)}{H(X)}$ means the ratio of all information extracted from original data.

C. Information Bottleneck

To extreme the motivation in feature engineering, it is important to find a "best" statistics that contains full of useful information. Suppose a sufficient statistics $S(X)$ that capture all the mutual information:

$$I(S(X); Y) = I(X; Y) \tag{6}$$

In addition, we do not want $S(X)$ to contain too much unrelated information in X. A minimal sufficient statistics (MSS) $T(X)$ is defined as:

$$T(X) = \arg \min_{S(X): I(S(X);Y) = I(X;Y)} I(S(X); X) \tag{7}$$

that means, T is a map of X, which keeps the information related to Y, but removes as much noise as possible.

Finding such a statistics is not easy. Information Bottleneck (IB) is used to provide a computational framework for finding approximate minimal sufficient statistics [19]. Denote $t \in T, x \in X$, then IB is formulated by finding $p(t|x), p(t), p(y|t)$ to solve the optimization problem:

$$\min\{I(X;T) - \beta I(T;Y)\} \tag{8}$$

It relaxes the restriction of $I(T(X);Y) = I(X;Y)$, and uses the Lagrange multiplier β to determine the trade-off between compression of X and prediction of Y. The solution of this problem is given as following:

$$\begin{cases} p(t|x) = Kp(t) \exp\Big(-\beta\, D_{KL}\Big[p(y|x)\,||\,p(y|t)\Big]\Big) \\ p(t) = \sum_x p(t|x)p(x) \\ p(y|t) = \sum_x p(y|x)p(x|t) \end{cases} \tag{9}$$

where K is a normalization constent. Solutions to these equations can be found by a convergent re-estimation method that generalizes the Blahut-Arimoto algorithm [11]. For different β, it can give different $I(X;T), I(T;Y)$ pairs, so that an information bottleneck curve can be plotted.

Theoratically, the "best" mahcine learning model should produce the MSS during learning. From this perspective, the feature engineering is using human inteligence to help the model to start at a closer point toward the MSS.

D. Maximum Entropy Model

To establish a Maximum Entropy Model (MaxEnt) in NLP settings, we first encode the inpute vector (or feature vector) x into binary code vector x_b. We define a binary-valued function:

$$f_i(x,y) = \begin{cases} 1, \text{ if the i-th code in } x_b \text{ is 1 and the class is } y \\ 0, \text{ otherwise} \end{cases} \tag{10}$$

which represents the realtionship between encoding and labels. Consider the expectation of f, there is an equation:

$$\sum_{x,y} \tilde{p}(x)p(y|x)f_i(x,y) = \sum_{x,y} \tilde{p}(x,y)f_i(x,y) \tag{11}$$

where $\tilde{p}$ means empirical distribution.

MaxEnt is to maximize the entropy of Y given X:

$$\max_{p(y|x)} H(Y|X) = \max_{p(y|x)} \left(-\sum_{x,y} p(x)p(y|x)\log p(y|x) \right) \tag{12}$$

It is proved in [13] that, using eq.(11) and eq.(12), the optimization problem is quavalent to:

$$\Lambda = \arg\max_{\Lambda} \tilde{p}(x,y)\log p_\Lambda(x,y) \tag{13}$$

$$p_\Lambda(y|x) = \frac{1}{Z(x)} \exp\left(\sum_i \lambda_i f_i(x,y) \right) \tag{14}$$

where Λ is the vector of λ_i, the Lagrange multipliers, also the weights associated with function f_i. $Z(x)$ is the normalization constant. These equations are equivalent to a multinomial logistic regression and its maximum likelihood estimation [13].

Many iterative algorithms can be used to solve this optimizaiton problem, such as Generalized Iterative Scaling (GIS), Improved Iterative Scaling (IIS), and Limited-memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS).

E. Multi-layer Perceptron Neural Network

A Multi-layer Perceptron (MLP) [15] is a class of feedforward artificial neural network (ANN). The structure of a one hidden layer MLP is shown in Fig. 1. The leftmost layer, known as the input layer, consists of a set of neurons representing the input data. Each neuron in the hidden layer transforms the values from the previous layer with a weighted linear summation $\sum_i w_i x_i$, followed by a non-linear activation function, such as the hyperbolic tan function. The output layer receives the values from the last hidden layer and transforms them into output values.

Suppose the MLP has more than 1 hidden layers, denoted by $T_1, T_2, \cdots, T_k$, and the output is denoted by $\hat{Y}$. We know that T_1 is a function of X, T_{i+1} is a function of T_i ($i = 1, \cdots, k-1$), and $\hat{Y}$ is a function of T_k. We also know that X is not independent of Y. Thus, we have the following Markov Chain:

$$Y \to X \to T_1 \to T_2 \to \cdots \to T_k \to \hat{Y} \tag{15}$$

which leads to

$$I(X;Y) \geq I(T_1;Y) \geq \cdots \geq I(T_k;Y) \geq I(\hat{Y};Y) \tag{16}$$

$$H(X) \geq I(X;T_1) \geq \cdots \geq I(X;T_k) \geq I(X;\hat{Y}) \tag{17}$$

It implies, the prediction precision of a neural network (related to $I(\hat{Y};Y)$) can never exceed the mutual information in the data, and each layer's leaning is limited by its previous layer. No matter how good a model is, if the mutual information in the data is limited, the model cannot achieve a high precision.

[1]Image Source: `http://scikit-learn.org/stable/modules/neural_networks_supervised.html`

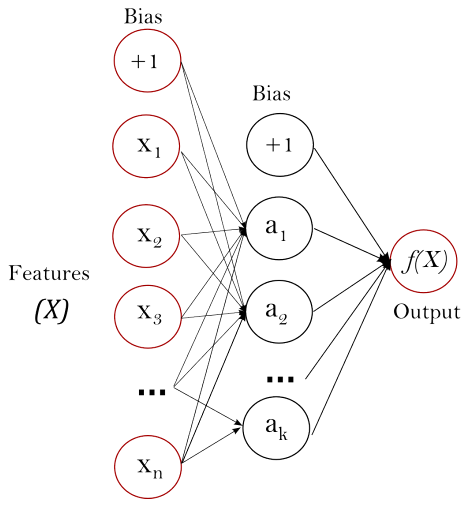

Fig. 1. One hidden layer MLP[1]

F. AdaBoost Model

I add AdaBoost in this project, because ensemble models are another kind of important Machine Learning models. AdaBoost is a specific Boosting algorithm for discrete classification [16]. A Boost classifier refers to the function:

$$F_T(x) = \sum_{t=1}^{T} f_t(x) \tag{18}$$

where f_t is a weak estimator, often a decision tree model. Each weak learner produces an output $h(x_i)$, for each sample in the training set. At each iteration, a weight w_t is assigned to each sample in the training set equal to the current error $E(F_{t-1}(x_i))$ on that sample. These weights can be used to make the training of the next estimator focus more on error samples. Then, the next estimator is selected and assigned a coefficient α_t so that the sum training error E_t is minimized:

$$\min_{\alpha_t} E_t = \min_{\alpha_t} \sum_i E[F_{t-1}(x_i) + \alpha_t h(x_i)] \tag{19}$$

Here $F_{t-1}(x)$ is the boosted classifier in the previous stage, E is some error function and $f_t(x) = \alpha_t h(x)$ is the weak learner that is being considered for addition to the final classifier.

III. Experiments

A. Data Source and Information

The dataset is downloaded from Google's Text Normalization Challenge on Kaggle [9]. There are totally 9,918,441 samples. By my preprocessing, each sample x has 3 words - the previous word (if available), the word itself, and the next word (if available). The label y is a class among all 16 classes: PLAIN (74.14%), PUNCT (18.96%), DATE (2.6%), LETTERS (1.54%), CARDINAL (1.35%), VERBATIM (0.79%), MEASURE (0.15%), ORDINAL (0.13%), DECIMAL (0.1%), MONEY (0.06%), DIGIT (0.05%), ELECTRONIC (0.05%), TELEPHONE (0.04%), TIME (0.01%), FRACTION (0.01%), ADDRESS (0.01%). It is a very unbalanced classification problem.

To reduce computation, in the following part (except Section III-G), only 1,000 samples are used. The classes are also be combined into 2 classes: PLAIN and notPLAIN. But Section III-G are based on all data.

As explained in Section II-A, data are treated as population, and the results are:

$$H(X) = 9.8986, H(Y) = 0.8160$$
$$H(X,Y) = 9.8986, I(X;Y) = 0.8160$$

We can see $H(X) = H(X,Y), H(Y) = I(X;Y)$, because almost all input data (the word and its context) can determine a class, so Y is almost determined by X. That is a usual situation in NLP settings.

B. Feature Engineering

By observation, the types of characters are related to the class of the word. For example, LETTERS class is often uppercase, and MONEY class often includes a money sign. So I extract the counts of following types of characters as features: upper letter, lower letter, digits, special ('.',',','-',':','/', each one is a feature), and others.

The information in the features are:

$$H(X_F) = I(X_F;X) = 4.6664 \;\Rightarrow\; \frac{I(X_F;X)}{H(X)} = 47.14\%$$

$$I(X_F;Y) = 0.7766 \;\Rightarrow\; \frac{I(X_F;Y)}{I(X;Y)} = 95.18\%$$

Therefore, the features only extract around half of the information in input data, but kept around 95% of mutual information of X and Y.

I also considered the binary features, which were the indicators weather a certain feature in a word or not, rather than the counts. The results are:

$$H(X'_F) = I(X'_F;X) = 2.2262 \;\Rightarrow\; \frac{I(X'_F;X)}{H(X)} = 22.49\%$$

$$I(X'_F;Y) = 0.7593 \;\Rightarrow\; \frac{I(X'_F;Y)}{I(X;Y)} = 93.05\%$$

These features use even less information to maintain 93% of useful information.

Although these features are useful for classification, they cannot be perfect. For example, a number text can belong to DATE, CARDINAL, TIME, etc. So I still keep the original words and their contexts, coded by Unicode, as an addition to the features. In this way, 100% information is kept for the model to select and use.

C. Information Bottleneck

I applied a Python program to solve the Information Bottleneck (IB). By shifting β a little by a little and calculating the pair $(I(X;T), I(Y;T))$, it can give a graph of the bound of mutual information.

However, for the data in this project, the result is useless. It is just a straight line whose $(I(X;T), I(Y;T))$ is from (0.8160,0.8160) to (9.8986,0.8160), implying that $I(X;T_0) \leq H(X) = 9.8986, I(Y;T_0) \leq H(Y) = 0.8160$ for any T_0. These bounds are already obvious without IB theories. The reason is that $p(X,Y)$ is hardly smooth [5], i.e., Y is almost a deterministic function of X, so the IB bound is very relaxed.

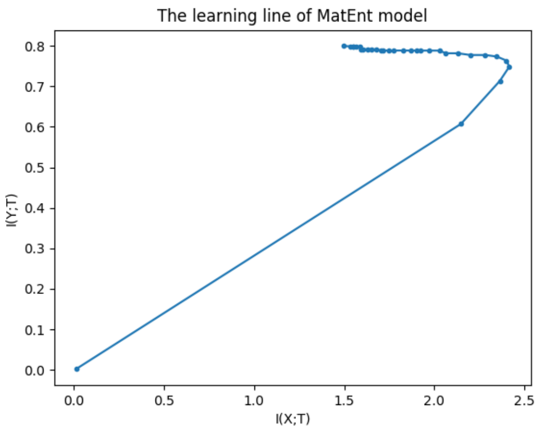

Fig. 2. The information measures during the learning of MaxEnt model: the point starts from the lower left, and moves to the upper right, but then moves to the left.

D. MaxEnt model learning

As illustrated in II-D, all features and data are coded as binary vectors, and their weights will be iteratively solved.

To monitor the information during learning, I rewrote the codes of MaxEnt and Improved Iterative Scaling (IIS) algorithms in Python. For each iteration in IIS, it calculates and records $I(X;T)$ and $I(Y;T)$, where $T = P(\hat{Y} = 1)$ is the probability prediction of the model by current weights. Since T is continuous, it is binned into 10 intervals between 0 and 1, as a discrete variable.

In Fig. 2, the learning process can be splitted into two phases. In phase 1, the model derives the information from both X and Y, so $I(X;T)$ and $I(Y;T)$ increase fastly. In phase 2, the model tries to “compress” the information in X, while keeping mutual information with Y, so $I(X;T)$ decreases. Finally it converges.

Also, information change is fast at the starting iterations, and slow down when the weights almost converges. So the forgetting part of the learning is hard but important.

E. MLP model learning

In this part, I used a standard MLP nueral network illustrated in II-F, including 3 hidden layers. The number of neurons in the hidden layers are 20, 15, 10. It is descending because the number of input bits is larger than the output bits. The activation function for the hidden layers is hyperbolic tan function. Denote T_1, T_2, T_3 as the hidden layers, and $T_4 = P(\hat{Y} = 1)$ the output layer. Since they are continuous, I binned each of them into 20 bins to discretized them. All models below are given 200 iterations for learning.

As shown in Fig. 3, I used different solving algorithms for the same model. They are Stochastic Gradient Descent (SGD), L-BFGS, and Adam. To eliminate the influence of randomness, each point is an average of 5 models starting from different initial states. The general trends are similar to the results in [5] and [17]: During phase 1, the layers increase the information on Y while also learning from X. In phase 2, the layers’ information on X decreases until convergence.

However, different solvers show different features of patterns. The pattern of SGD is more unstable than others, especially in layer 4 (output layer). It may because that it divides the data into some batches, and optimize one batch at each iteration, so the learning process is influenced by different batches. The pattern of L-BFGS has more vertical movement than horizontal swinging. It is an algorithm in the family of quasi-Newton methods, which utilize the first and second derivatives. It moves like that phase 1 and

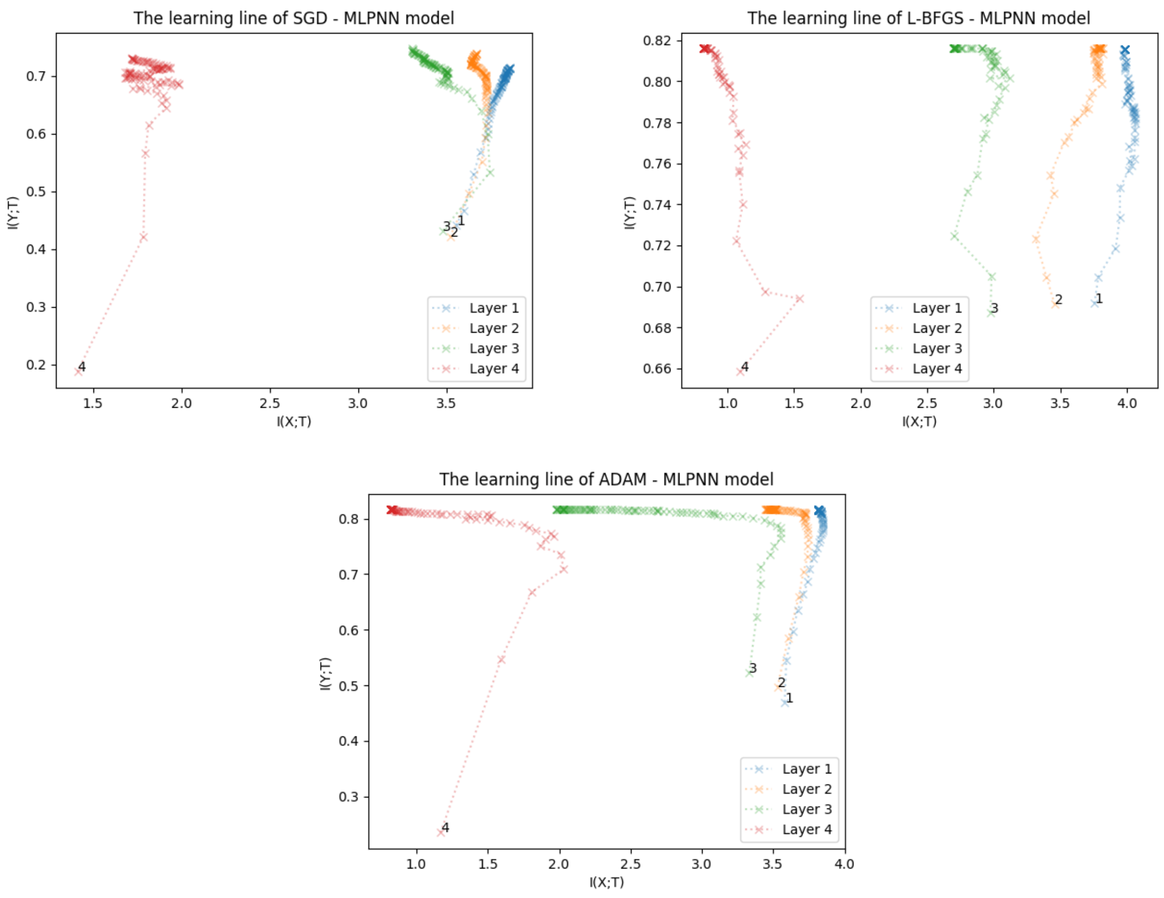

Fig. 3. The information measures during the learning of MLP nueral network model, seperately trained by SGD (upper left), L-BFGS (upper right), and Adam (lower) solvers.

phase 2 are combined together. The pattern of Adam is the most "standard" two-phase pattern, in all 4 layers. It is an extension to SGD, but it combines the advantages of AdaGrad and RMSProp, and also makes use of the average of the second moments of the gradients.

It can be noticed that, phase 2 is most obvious in the graph of Adam, and phase 2 is believed to be most important in learning [5]. Meanwhile, Adam algorithm, is usually faster and easier to converge in various experiments [20]. So a conjecture is that the good performance is related to the good features of the information learning graph.

F. AdaBoost learning

The experiment in this part is to figure out the information change during training an ensemble model. The process is like the one for MaxEnt, but the information changes with the increasing number of estimators in the ensemble, rather than with the updating of the weights.

The result in Fig. 4 does not show an obvious phase 1 pattern like Fig. 2, but phase 2 can still be recognized in the upper right area. The swinging should be caused by the uncertain effects of adding an estimator in the ensemble. It can also be seen that the first estimator (a decision tree) is already very effective, considering it achieves a high $I(Y;T)$ with a relatively small $I(X;T)$. The following estimators slowly help the model to derive more $I(Y;T)$.

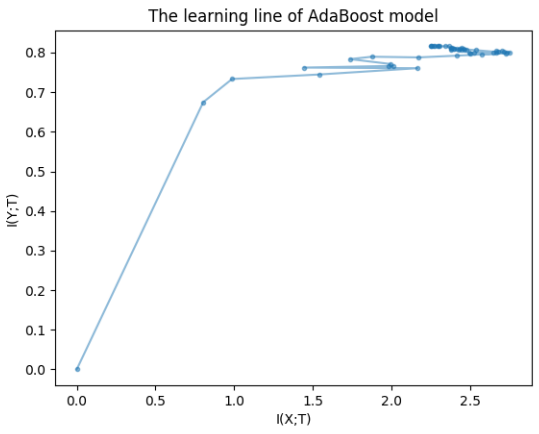

Fig. 4. The information measures during the learning of AdaBoost model: the point starts from the lower left, and moves to the upper right with swinging, but finally moves to the left.

G. Precision Comparison

For solving the specific word classification problem, I calculated the test accuracy of all the models mentioned above. In this case, all data were included. The train set - test set ratio is 9:1. Half of samples in PLAIN class were dropped for reducing computation, and also reducing unbalance.

TABLE I
TEST ACCURACY FOR DIFFERENT MODELS

Model	**Test Accuracy**
MaxEnt	99.0806%
MLP Neural Network[a]	99.4087%
AdaBoost[b]	95.7639%
XGBoost[c]	99.5846%

[a]The structure of hidden layers is (20,15,10).
[b]The number of estimators is 50.
[c]XGBoost is an implement of Gradient Boosting Decision Tree. The number of estimators is 50.

Table I shows that XGBoost and MLP Neural Network performs the best, while AdaBoost performs the worst. Comparing MaxEnt and MLP, we can see that unlinear models such as neural network performs better in this case. Comparing AdaBoost and XGBoost (Gradient Boosting), we can see that the gradient method performs better in this case. Gradient Boosting is also an ensemble model, which learns to fit the loss at each iteration, rather than using a linear weight to add up estimators like in AdaBoost, so it can better fit nonlinear data. In addition, the low accuracy of AdaBoost may also be related to the more "chaotic" pattern in Fig. 4, compared with the other two models.

IV. CONCLUSION

In this project, information theory is used to explain various aspects of natural language processing classification experiments. In summary, the results show the following conclusions:

1) In usual NLP classification settings, $H(X)$, the entropy of language data, is much larger than $H(Y)$, the entropy of labels. Also $I(X;Y)$ is close to $H(Y)$, because usually a fixed language sample belongs to only one class.

2) In the above situation, Information Bottleneck is useless. It can only give $H(X)$ as the upper bound of $I(X;T)$, which is obvious.
3) Comparing the information ratios of $\frac{I(X^F;Y)}{I(X;Y)}$ and $\frac{H(X^F)}{H(X)}$ can explain and measure the effect of a feature extractor.
4) Various learning process of NLP classification models generally have a two-phase pattern: fast learning from X and Y, then compress useless information in X. It shows in the experiments of MaxEnt, MLP Neural Network, and AdaBoost models.
5) Different solving algorithms for neural network have different learning graphs. A conjecture is that, a good pattern of strict two phases in the learning graph is related to a good performance (faster, easier to converge) of the corresponding algorithm.
6) XGBoost and MLP neural network perform the best in this case. A conjecture is that, the bad performance of AdaBoost is related to its chaotic learning pattern.

In future study, the conjectures in 5) and 6) will need more experiments and theories. Meanwhile, the relationship between the learning pattern and the data set, hyper-parameters, or types of learning (unsupervised or semi-supervised), also needs more researches.

V. Appendix

The experiments in this project are done in the environment:

- OS: Linux Mint 18.2 Cinnamon 64-bit
- Processor: Intel Core i7-6600U CPU @ 2.60GHz $\times$ 2
- Memory: 11.5 GB
- Software: Python 3.6.1 (iPythin 5.3.0, Anaconda 1.6.3, 64-bit)

The description of the Python code files:

TABLE II
Description of Python files

File/Folder Name	Description
./1_preprocess.py	The codes for preprocessing the csv data.
./2_xgb.py	The codes for using XGBoost package for classification.
./3_functional.py	The codes for explore the functional features in the data.
./4_nltk.py	The codes for performing MaxEnt, MLP, and AdaBoost for classification.
./5_info.py	The codes for monitoring the the learning process for the above models.
./maxent.py	The MaxEnt model with IIS algorithm, which I rewrote for easily monitoring the variables during iteration.
./sparse_class.py	A sparse data class I modified from NLTK package [21], and used in "./maxent.py".
./IB/	An Information Bottleneck program by Github user "djstrouse", used in "./5_info.py". URL: https://github.com/djstrouse/information-bottleneck

References

[1] Samuel, Arthur L. "Some studies in machine learning using the game of checkers." *IBM Journal of research and development* 44.1.2 (2000): 206-226.

[2] Hu, Bao-Gang. "Information theory and its relation to machine learning." *2015 Chinese Intelligent Automation Conference*. Springer, Berlin, Heidelberg, 2015.

[3] Wang, Meihong, and Fei Sha. "Information theoretical clustering via semidefinite programming." *International Conference on Artificial Intelligence and Statistics*. 2011.

[4] Foroozand, Hossein, and Steven V. Weijs. "Entropy Ensemble Filter: A Modified Bootstrap Aggregating (Bagging) Procedure to Improve Efficiency in Ensemble Model Simulation." *Entropy* 19.10 (2017): 520.

[5] Shwartz-Ziv, Ravid, and Naftali Tishby. “Opening the Black Box of Deep Neural Networks via Information.” *arXiv preprint arXiv*:1703.00810 (2017).
[6] Jaynes, Edwin T. “Information theory and statistical mechanics.” *Physical review* 106.4 (1957): 620.
[7] Bengio, Yoshua. “Learning deep architectures for AI.” *Foundations and trends in Machine Learning* 2.1 (2009): 1-127.
[8] Kgl, Balzs. “The return of AdaBoost. MH: multi-class Hamming trees.” *arXiv preprint* arXiv:1312.6086 (2013).
[9] Text Normalization Challenge - English Language. *Kaggle* and *Google*. `https://www.kaggle.com/c/text-normalization-challenge-english-language`
[10] Bishop, Christopher M. “Pattern recognition and machine learning.” springer, 2006.
[11] Tishby, Naftali, Fernando C. Pereira, and William Bialek. “The information bottleneck method.” *arXiv preprint* physics/0004057 (2000).
[12] Ng, Andrew. “Machine Learning and AI via Brain simulations.” *Stanford University* (2013).
[13] Berger, Adam L., Vincent J. Della Pietra, and Stephen A. Della Pietra. “A maximum entropy approach to natural language processing.” *Computational linguistics* 22.1 (1996): 39-71.
[14] Paninski, Liam. “Estimation of entropy and mutual information.” *Neural computation* 15.6 (2003): 1191-1253.
[15] Tan, Pang-Ning. *Introduction to data mining*. Pearson Education India, 2006.
[16] Freund, Yoav, and Robert E. Schapire. “A desicion-theoretic generalization of on-line learning and an application to boosting.” *European conference on computational learning theory.* Springer, Berlin, Heidelberg, 1995.
[17] Naftali Tishby. “Information Theory of Deep Learning”. *Deep Learning: Theory, Algorithms, and Applications*. Berlin, June 2017 `https://youtu.be/bLqJHjXihK8`
[18] Ratnaparkhi, Adwait. “A simple introduction to maximum entropy models for natural language processing.” *IRCS Technical Reports Series* (1997): 81.
[19] Tishby, Naftali, and Noga Zaslavsky. “Deep learning and the information bottleneck principle.” *Information Theory Workshop (ITW)*. IEEE, 2015.
[20] Kingma, Diederik, and Jimmy Ba. “Adam: A method for stochastic optimization.” *arXiv preprint* arXiv:1412.6980 (2014).
[21] Bird, Steven. “NLTK: the natural language toolkit.” *Proceedings of the COLING/ACL on Interactive presentation sessions*. Association for Computational Linguistics, 2006.