Performance analysis of predictive classification of human instructions using NLP models

Authors: Ziyu Shi, Xiuyuan Chen, Yucheng Yang
NetID: zs148, xc202, yy341

Abstract

This paper presents a comprehensive analysis of the performance of different Natural Language Processing (NLP) models on the task of human instruction classification. Our experiment aims to evaluate and compare the accuracy of various NLP models in categorizing human instructions into ten categories. The NLP architecture implemented and compared in the experiment includes traditional machine learning algorithms such as Naive Bayes, Support Vector Machines (SVM) and Multi-layer Perceptrons (MLP), as well as the state-of-the-art deep learning models like Recurrent Neural Network (RNN) and Long Short-Term Memory Networks (LSTM).

1 Introduction

In the fast-changing world of Natural Language Processing (NLP), accurately figuring out what people mean when they give instructions is really important. It's used in many things like making virtual assistants smarter or understanding how people feel in sentiment analysis. This research looks at lots of different NLP models, both old-school ones and the newest deep learning methods, to see how well they can understand and classify human instructions.

We used the dataset obtained from Hugging Face Datasets called “no_robots”, URL: https://huggingface.co/datasets/HuggingFaceH4/no_robots. This dataset consists of many human instructions and reflects the variations in natural language. It is high-quality and standardized data.

In our research, we looked at a mix of old-school and new-school NLP models. We checked out the classics like Naive Bayes, Support Vector Machines (SVM), and Multi-layer Perceptrons (MLP). Besides, we used Recurrent Neural Networks (RNN) and Long Short-Term Memory Networks (LSTM) for comparison and analysis. We're basically playing detective, trying to figure out what each model is good at and what it's not so great at when it comes to understanding human instructions. This detective

work may help other researchers choose the best NLP model for understanding how humans talk.

2 Background

In Natural Language Processing (NLP), choosing between Generative Probabilistic Models and Discriminative Neural Networks is crucial. Generative models focus on joint occurrences but may struggle with intricate language details, while discriminative models excel in diverse tasks [1]. The "no_robots" dataset, rich in linguistic nuances, is essential for applications like virtual assistants and sentiment analysis, emphasizing the real-world importance of accurate human instruction classification.

2.1 Generative Probabilistic Models vs. Discriminative Neural Networks

Generative models, such as Naive Bayes, are created to understand how both the input features and labels are likely to occur together. However, when it needs to classify things like human instructions, generative models might struggle with details found in natural language due to their focus on overall data patterns.

Discriminative models, including Support Vector Machines (SVM), Multi-layer Perceptrons (MLP), and deep learning architectures like Recurrent Neural Networks (RNN) and Long Short-Term Memory Networks (LSTM), focus on the boundary between different classes. Discriminative models often have better performance than generative models in classification tasks, especially with intricate patterns and varied data [1]. Besides, deep learning models show remarkable success in non-linear relationships data.

Generative models help us understand how data is created based on probability, while discriminative models, especially neural networks, are great at learning specific features that help with classification [1]. The decision on which approach to use usually relies on how complicated the task is and how much variety there is in the dataset.

2.2 Significance of Distinguishing "no_robots" Content

The "no_robots" dataset includes a spectrum of linguistic nuances and natural language variations. Identifying "no_robots" content is imperative for practical applications such as enhancing virtual assistant performance, ensuring accurate response to user commands, and refining sentiment analysis for customer feedback analysis. The intricate linguistic nuances within human instructions offer valuable

insights into the adaptability and robustness of NLP models in navigating real-world language variations.

2.3 Importance of Human Instruction Classification Research

Exploring human instruction classification is important in NLP for various reasons. Increasing the precision of human instruction classification enhances user interactions with NLP systems could lead to better user experience. Additionally, understanding the performance of NLP models in classifying human instructions provides key insights into their adaptability across diverse domains. This research not only contributes to improving user interactions but also guides researchers in selecting the most suitable NLP models for the categorization of human instructions, taking into account their individual strengths and weaknesses.

3 Our methods

In this section, we will delve into the detailed exploration of the dataset used in our experiments. We will discuss the processes and principles behind the implementation of three generative language models: Naive Bayes, Support Vector Machine (SVM), and Multilayer Perceptron (MLP). Additionally, we will cover the processes and principles of two discriminative language models: Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM). For more implementation details, please refer to the code of the research paper.

3.1 Data

Our experimental dataset is derived from the No Robots dataset provided by Hugging Face. This dataset comprises a total of 10,000 instructions and demonstrations, meticulously annotated by proficient human annotators [2]. Out of these, 9,500 are categorized as training and validation data, while the remaining 500 serve as test data. Each row of the data contains four main columns: the first column named "prompt" describes the objective the model is expected to fulfill; the second column, "prompt_id," is a 64-character unique identifier for the prompt; the third column is an array of main messages containing the dialogue content of the system, user, and assistant roles; and the last column is the category, indicating to which category the row belongs. Table 1 presents the total of 10 categories included in this dataset along with their respective occurrence frequencies.

Table 1. Categories and their respective occurrence frequencies in the No Robots dataset [2]

Category	Count
Generation	4560
Open QA	1240
Brainstorm	1120
Chat	850
Rewrite	660
Summarize	420
Coding	350
Classify	350
Closed QA	260
Extract	190

As observed in the table, the majority of the tasks undertaken by language models in this dataset fall under the category of generation, followed by Open QA and Brainstorm. Since the downloaded dataset is in Parquet format, we eventually converted it to a CSV file. It's worth noting that we utilized only 9,500 training examples, randomly splitting them into two parts with an 8:2 ratio for training and testing sets. Subsequently, we used TFIDF to transform the message text into a matrix of token counts as input for different models, where TFIDF is an acronym for Term Frequency-Inverse Document Frequency, a numerical metric indicating the significance of a term within a document compared to a corpus of documents.

3.2 Naive Bayes

The Naive Bayes classifier is a probabilistic statistical classifier based on Bayes theorem and falls under the category of generative models [3]. Its fundamental concept relies on the assumption of independence among features, meaning that given a category, the features are assumed to be mutually independent. The generative process involves two main steps: estimation and prediction.

The model first computes the prior probabilities for each category, representing the likelihood of each category occurring without observing any features. Subsequently, using frequency statistics, it calculates the relative frequencies of each feature value within each category based on Bayes theorem. Finally, for a given input feature vector, the model calculates the posterior probabilities for each category and selects the category with the highest posterior probability as the predicted result [3]. Naive Bayes is simple yet efficient, performing well on small-scale datasets and exhibiting good performance when dealing with high-dimensional data. It is commonly employed in text classification tasks such as spam filtering and sentiment analysis.

In terms of code implementation, we simply invoked the 'MultinomialNB' method from the scikit-learn library, which is designed for multinomial distributed data. We also utilized the 'accuracy_score' from the sklearn library to check the testing accuracy of the model.

3.3 Support Vector Machine

Support Vector Machine (SVM) is a supervised classification model used for classification and regression. The primary objective of SVM is to find a hyperplane that separates data points of different classes while maximizing the margin between the two classes [4]. SVM can employ various kernel functions to map non-linear problems into higher-dimensional spaces, making the data linearly separable. Commonly used kernel functions include linear, polynomial, and radial basis function (RBF) kernels [4]. For new input samples, SVM projects them onto the learned hyperplane and determines the sample's class based on the projection's position. This characteristic makes SVM perform well in high-dimensional spaces, particularly in cases where the feature dimensionality exceeds the number of samples. SVM is highly flexible and can effectively handle nonlinear problems by selecting different kernel functions.

When implementing SVM, we also utilized the 'SVC' method from the scikit-learn library. The 'SVC' class provides an implementation of the SVM algorithm for both linear and non-linear classification problems. We also employed the method of controlled variables to explore the impact of different regularization coefficients and kernel functions on the predictive accuracy.

3.4 Multilayer Perceptron

A multilayer perceptron (MLP) is a fundamental feedforward neural network structure, which is a classification model. It consists of an input layer, several hidden layers, and an output layer, with each layer comprising multiple neurons [5]. It belongs to discriminative neural networks. Each connection has a weight, representing the strength of the connection, and each neuron has a bias used to adjust the neuron's activation [6]. These parameters are learned during the training process. In the hidden and output layers, activation functions introduce non-linear properties, allowing the network to learn more complex functions. The discriminative process of an MLP includes forward propagation, loss computation, backward propagation, and iterative training [6]. Typically, the model is based on gradient descent optimization algorithms, continuously adjusting parameters to make the model's predictions as close as possible to the actual labels. In terms of code implementation, we utilized the 'MLPClassifier' method from the scikit-learn library.

3.5 Recurrent Neural Network

A Recurrent Neural Network (RNN) is a specialized neural network architecture designed for handling sequential data [7]. RNN is also considered as a classification model. Unlike traditional neural networks, RNNs incorporate recurrent connections that allow information to be passed and persisted within the network. The key feature of RNN is its recurrent connections, enabling the network to consider past information when processing inputs at each time step. The output at each time step depends not only on the current input but also on the output from the previous time step [7]. Additionally, RNN introduces hidden states that are updated at each time step, carrying the network's memory of the sequence. The hidden state serves as an internal representation capturing contextual information within the sequence. The inference process of an RNN consists of five main components: initializing the hidden state, forward propagation at each time step, predicting outputs, backward propagation, and iterative training.

In the code implementation section, we convert categorical arrays into a one-hot representation. We created a linear stack neural network model. We added a SimpleRNN layer with 1000 neurons, using the Rectified Linear Unit (ReLU) as the activation function. A Dropout layer was included to prevent overfitting by randomly dropping input units, thereby preventing the network from relying too heavily on certain input features. Finally, we added a fully connected layer with an output dimension of 10, utilizing the Sigmoid activation function for handling classification tasks. The detailed structure of our RNN model can be viewed in Figure 1. Additionally, we used the RMSprop optimizer, specified the loss function as

categorical cross entropy, and set accuracy as the performance evaluation metric. Lastly, the model underwent training for 5 epochs, with the validation set comprising 20% of the training set, and each batch having a size of 32.

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 simple_rnn_1 (SimpleRNN)    (32, 1000)                55833000

 dropout_1 (Dropout)         (32, 1000)                0

 dense_1 (Dense)             (32, 10)                  10010

=================================================================
Total params: 55843010 (213.02 MB)
Trainable params: 55843010 (213.02 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Figure 1. Our implemented RNN model

3.6 Long Short-Term Memory

Long Short-Term Memory (LSTM) is a special type of recurrent neural network designed to address the issues of vanishing and exploding gradients that traditional RNNs encounter when processing long sequences [7]. LSTM introduces a structure called gates, which selectively retain or forget information, enabling it to more effectively capture and transmit long-term dependencies within sequences. Within LSTM, there exists a cell state, allowing the network to pass and store information at each time step. The cell state facilitates the retention of information over long durations, thereby resolving the problem of long-term dependencies in RNNs. Typically, LSTM employs three types of gates to control the flow of information: the Forget Gate, the Input Gate, and the Output Gate [7]. The inference process of LSTM involves initializing the cell state and hidden state, calculating gate inputs at each time step, and iteratively processing the entire sequence.

We adopted the same structure, optimizer, and performance evaluation metrics as in the RNN, as well as the number of training iterations, validation set ratio, and batch size, to implement the LSTM. The only difference was that we replaced the first layer in the sequential model from the original RNN to an LSTM, while keeping the activation functions consistent. The structure of our LSTM model can be viewed in Figure 2.

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (32, 1000)                223332000

 dropout_2 (Dropout)         (32, 1000)                0

 dense_2 (Dense)             (32, 10)                  10010

=================================================================
Total params: 223342010 (851.98 MB)
Trainable params: 223342010 (851.98 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Figure 2. Our implemented LSTM model

4 The comparison of two approaches

In this section, we compare and analyze the advantages and disadvantages of two approaches, generative probabilistic model and discriminative neural network, according to the quantitative result on the same task of human instruction classification.

4.1 Generative probabilistic model

Generative probability model is a statistical model that describes the joint distribution of observed data and potential variables. This kind of model aims to find the pattern and structures underlying the data, so that well-trained models can generate new instances that share similar characteristics or make the most appropriate prediction based on the input data. In this experiment, we have implemented one famous generative probabilistic model, Naive Bayes, to complete the same prediction task of human instruction classification.

Naive Bayes is easy to implement and is efficient for high-dimensional data and small datasets. Its simplicity and efficiency are helpful to implement and compare it with other models. In this experiment, the performance of Naive Bayes model ranks the lowest among our methods. The speculated reason is that, despite Naive Bayes being suitable for training data with high-dimensional features, the features in this training

data are not high-dimensional. In addition, Naive Bayes does not consider dependence between features and different scenarios and so oversees the relationship between features. Therefore, Naive Bayes is unable to capture the correlation in the complex context, resulting in poor predictive performance in this experiment. The accuracy of Naive Bayes is 0.451.

4.2 Discriminative neural network

Discriminative neural network is a kind of neural network model designed for discriminative tasks, its primary goal is to learn the mapping relationship between the input data and the corresponding output label. This kind of model typically uses loss functions to measure the disparity between the predicted outputs and true labels and then iteratively back-propagate the loss to update weights at each layer in the network. In this experiment, we have implemented four discriminative neural networks to perform the same classification task using the same dataset.

The first implemented model is SVM. The computational complexity of SVM is much higher than Naive Bayes, so SVM takes much more training time. We have implemented 9 SVM models with different values of C and kernels. SVM is well-suited for classification tasks and various kernels can be fit for different data. The accuracy of the best result with SVM is 0.811. The predictive accuracy of 9 SVMs is exhibited in Table 2.

Table 2. Accuracies of predictive classification by SVM with different values of C and kernels

Kernel	C	Accuracy
Linear	0.1	54.89%
	1	78.78%
	10	81.05%
Radial Basis Function, RBF	0.1	46.37%
	1	73.63%
	10	75.79%
Polynomial	0.1	45.00%
	1	49.68%
	10	51.95%

The second implemented model is MLP. Training an MLP is even more computationally expensive than SVM because the result of MLP is heavily influenced by its hyperparameters and the scale of datasets. With small datasets, MLP is prone to overfitting and leads to worse accuracy than SVM. The accuracy of MLP is 0.768.

The third implemented model is simple RNN. RNN is a kind of neural network architecture designed to process sequential data by maintaining hidden states that capture past information observed in the sequence. It has been applied in many domains including natural language processing and achieved extraordinary accomplishments. Its architecture is complicated compared to previous generative probabilistic models and so takes more time to train and predict. However, RNN can easily overfit using small datasets. To prevent heavy overfitting, we implemented dropout on the model to deactivate half of the neurons in the hidden layers. After five epochs of training, the testing accuracy of RNN is 0.817.

The fourth implemented model is LSTM, which is an advanced RNN that is designed to address the vanishing gradient problem and capture long-range dependencies. After the addition of memory units and three types of gates, LSTM can selectively retain important information over extended sequence, which makes it more suitable for tasks where understanding context is crucial. In addition, LSTM mitigates the problem of vanishing and exploding gradients that exist in simple RNN, as well as alleviates overfitting prevalent in simple RNN. Despite the advantages, the architecture of LSTM is more complicated than simple RNN and the computational complexity is more expensive. After five epochs of training using the same parameters, the testing accuracy of LSTM is 0.821.

4.3 Comparison on time and space complexity

The time complexity of Naive Bayes is $O(MK)$, where M is the number of features and K is the number of classes. The space complexity of Naive Bayes is $O(MK)$.

The time complexity of SVM is $O(N^2\text{M})$ to $O(N^3M)$, where N is the number of training instances and M is the number of features.The space complexity of SVM is $O(N)$.

The time complexity of MLP is $O(NHE)$, where N is the number of training instances and H is the number of hidden neurons and E is the number of training epochs. The space complexity of MLP is $O(IH)$, where I is the number of input features.

The time complexity of simple RNN is $O(n)$, where n is the length of the input sequence.The space complexity of simple RNN is $O(IH)$.

The time complexity of LSTM is $O(IHNE)$, where I is the number of input features, H is the number of hidden neurons, N is the number of training instances, and E is the number of training epochs. The space of LSTM is $O(IH)$.

The time complexity of Naive Bayes is $O(NM)$, where N is the number of training instances and M is the number of features. The space complexity of Naive Bayes is $O(MK)$, where M is the number of features and K is the number of categories.

Table 3. The time complexity and space complexity of five implemented models

Model	Time complexity	Space complexity
Naïve Bayes	$O(MK)$	$O(MK)$
SVM	$O(N^2\text{M})$ - $O(N^3M)$	$O(N)$
MLP	$O(NHE)$	$O(IH)$
Simple RNN	$O(n)$	$O(IH)$
LSTM	$O(IHNE)$	$O(IH)$

The complexity of discriminative neural networks is typically more complicated than generative probabilistic models. Therefore, discriminative neural networks are well-suited for tasks with extended input sequence and better understanding context. In this NLP experiment, the average accuracy of discriminative neural networks is higher than generative probabilistic models. The result is shown in Table 4.

Table 4. The average accuracy of generative probabilistic models and Discriminative neural network

Approach	Accuracy
Generative probabilistic model	45.05%
Discriminative neural network	80.42%

5 Conclusion

In our study of human instruction classification, we compared generative probabilistic models, represented by Naive Bayes, with discriminative neural networks, including SVM, MLP, simple RNN, and LSTM. The generative model showed limitations in capturing complex relationships, achieving an accuracy of 45.05%. In contrast, discriminative neural networks outperformed, collectively reaching an average accuracy of 80.42%.

Discriminative models, especially SVM, MLP, RNN, and LSTM, showcased effectiveness in handling the intricacies of human instructions. SVM's adaptability with different kernels and MLP's competitive performance were notable. Despite computational challenges, RNN achieved 81.68% accuracy, emphasizing the significance of sequence handling. LSTM outshone other models with 82.11% accuracy, showcasing its ability to capture long-range dependencies.

The time and space complexity analysis highlighted the increased demands of discriminative neural networks. However, their capacity to comprehend extended input sequences and contextual nuances justified the complexity.

In summary, our findings underscore the superiority of discriminative neural networks in human instruction classification tasks. This research provides practical insights for model selection in NLP applications, guiding researchers and practitioners toward effective choices for similar tasks.

6 Reference

[1] Wu, Ying Nian, et al. "A tale of three probabilistic families: Discriminative, descriptive, and generative models." Quarterly of Applied Mathematics 77.2 (2019): 423-465.

[2] https://huggingface.co/datasets/HuggingFaceH4/no_robots

[3] Vikramkumar, Vijaykumar, B., & Trilochan (2014). Bayes and Naive Bayes Classifier.

[4] M. A. Hearst, S. T. Dumais, E. Osuna, J. Platt and B. Scholkopf, "Support vector machines," in IEEE Intelligent Systems and their Applications, vol. 13, no. 4, pp. 18-28, July-Aug. 1998, doi: 10.1109/5254.708428.

[5] G. Singh and M. Sachan, "Multi-layer perceptron (MLP) neural network technique for offline handwritten Gurmukhi character recognition," 2014 IEEE International Conference on Computational Intelligence and Computing Research, Coimbatore, India, 2014, pp. 1-5, doi: 10.1109/ICCIC.2014.7238334.

[6] Tolstikhin, Ilya O., et al. "Mlp-mixer: An all-mlp architecture for vision." Advances in neural information processing systems 34 (2021): 24261-24272.

[7] Sherstinsky, Alex. "Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network." Physica D: Nonlinear Phenomena 404 (2020): 132306.