Predict NAS Multi-Task by Stacking Ensemble Models using GP-NAS

Ke Zhang
kezzhang@gmail.com

Abstract

Accurately predicting the performance of architecture with small sample training is an important but not easy task. How to analysis and train dataset to overcome overfitting is the core problem we should deal with. Meanwhile if there is the mult-task problem, we should also think about if we can take advantage of their correlation and estimate as fast as we can. In this track, Super Network builds a search space based on ViT-Base. The search space contain depth, num-heads, mpl-ratio and embed-dim. What we done firstly are pre-processing the data based on our understanding of this problem which can reduce complexity of problem and probability of over fitting. Then we tried different kind of models and different way to combine them. Finally we choose stacking ensemble models using GP-NAS with cross validation. Our stacking model ranked 1st in CVPR 2022 Track 2 Challenge.

1. Introduction

As limited data sources, the cost of collection high precision large-scale dataset, the difficult of labeling data manually, limitation of computing power, the requirement of fast estimation, training based on small size and predict accurately is more and more important in widely different industry such as computer vision, finance, E-commerce, medicine and so on. Meanwhile, as we have more and more different source to collect data, we should consider how to handle mult-task simultaneously without sacrificing precision or take advantage of their collaboration.

Neural Architecture Search (NAS) [5] [13] [15] has been used to design deep neural network architectures successfully. It outperform hand-designed models in many computer vision tasks. However, there are still several issues need to be solved when we use NAS kind of algorithm. For example, inconsistent with the performance of the same network trained independently. Poor performance in small sample sized training and prediction. Recently GP-NAS(Gaussian Process based Neural Architecture Search) was introduced [11], the correlation are modeled by the kernel function and mean function. Bayesian theorem and information theory was implemented deeply in this model which theoretically ensure the good performance especially when sample size is small.

However, we find that naively and simply throw data into GP-NAS can not give us relatively good result. We think there are several reasons. First of all, this kind of model is sensitive to the input which require accurately pre processing data. Secondly, purely running this model can not fully mining most of information from data especially when there are nonlinear or complex feature inside the dataset.

In this paper, after we deeply analysis this problem and try different kind of model, we finally choose to stack several ensemble models by GP-NAS which give us quite good performance in CVPR 2022 Track2 Challenge.

2. How we deal this problem

2.1. Raw data

In this challenge, Super Network builds a search space based on ViT-Base for 8 tasks. The search space involves the number of depth, the number of heads, the mlp-ratio and embed-dim. For each task, the training data include 500 samples whose input is the model structure and whose label is the relative ranking of struture. While the test data includes 99500 samples. We can find training sample size is relatively small compare with test data which required the ability of accurately fitting based on small sample dataset. Obviously this is the purpose of this track challenage that predicting the performance of any architecture accurately with small or without training.

2.2. Data pre processing

For this problem, most straight forward and direct way to do is training all the network using default structure value. The problem of this way is singularity (embed-dim doesn't contain information) and does not take advantage of background information. Therefore, we pre process the data by following steps:

1. Transfer the depth encoding(j,k,l) to integer. Here we consider ordinal encoding instead of one-hot encoding as we assume depth has some monotonical correlation with predictability, that deeper depth with more information. the experiment also confirm our thought. [12]

Ordinal encoding:

$$(j, k, l) \Rightarrow (1, 2, 3) \tag{1}$$

One-Hot encoding:

$$(j, k, l) \Rightarrow \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{2}$$

2. Getting rid of embed-dim feature parameters as it doesn't include any information which avoid singularity and reduce the dimension of problem by 1/4. On the other hand when the actual depth of a sub-network is less than 12, its encoding trailing end is padded with 0, here we change 0 to 2 as we assume 2 represent neutral information. After this transformation, depth encoding feature has average 0.015 correlation with last four encoding features instead of average 0.75. Finally we normalize data from (1,2,3) to (-1,0,1).

3. Using inverse sigmoid function as activiation function [2]. As we known, ranking data follow uniform distribution. By inverse sigmoid function we transfer uniform distribution to Gaussian distribution which is best choice for most model. After we get prediction, we use sigmoid function to transfer output back to uniform distribution as rank output, then we round up to nearest integer.

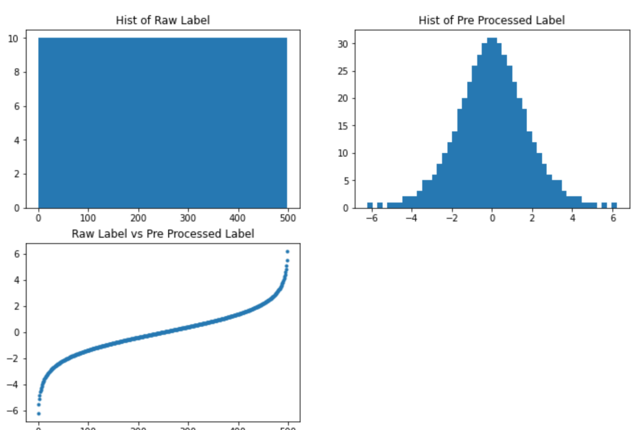

Figure 1. Using inverse sigmoid function as activiation function to transfer rank label (Uniform distribution) to new label (Gaussian distribution)

The above figures shows the histgram of raw label, histgram of pre processing label and relationship between raw label and pre processing label.

$$z = \frac{\log(y+1)}{n-y}, y \in [0, n-1] \tag{3}$$

Eq. (3) is the inverse sigmoid function we mentioned above.

4. we also try lots of other ideas such as different activiation function, adding Gaussian noise and so on. None of them obviously improve across different task.

2.3. Model Construction

After we pre processed our raw dataset, then we consider to construct our training and testing framework. we tried lots of models and different combined ways (details are in next section), finally we choose to stack the ensemble models [14] [8] [7] with final estimator GP-NAS [4].

There are several steps to get our final result. Firstly, we split our dataset into training dataset and testing dataset. We train our sub model by full training dataset. Then we split our training dataset to K fold and use K-1 to get prediction from each sub model [10]. Secondly, for each K, we use prediction from each sub model as training dataset and remain 1 fold as validated data to train GP-NAS and repeat K times. Now we can predict based on GP-NAS to get our second layer prediction.

Finally, we post processing our dataset that using sigmoid function as activiation function to transfer our second layer prediction to uniform distribution and round it up to nearest integer as the rank prediction.

$$s = \frac{n-1}{1+\exp(-z)}, z \in [-\infty, +\infty] \tag{4}$$

Eq. (4) is the sigmoid function we used.

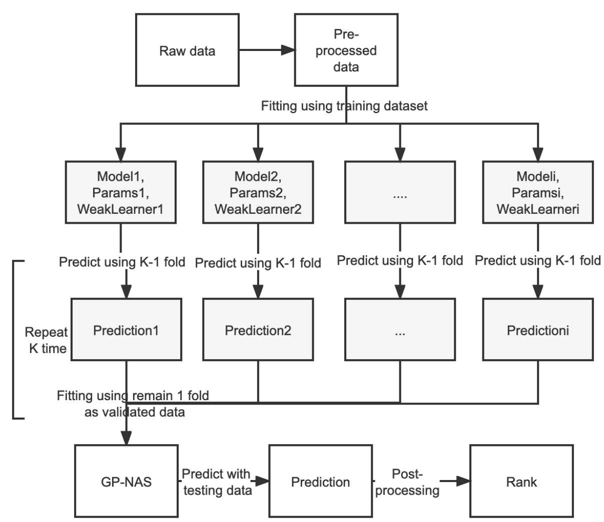

Figure 2. End to end training and predicting process

Until now, we almost constructed our model framework. However, this doesn't mean we already get the good result from this framework. At beginning we use same parameters such as learning rate, loss function and depths of model for different tasks,but the output is not as good as final result. This phenomenon definitely make sense because different dataset have different signal noise ratio and property. For example, higher signal noise ratio task require larger learning rate, more non-linear and complexity structure need larger maximum of depth. In a word, depending on the problem, we should find a different variance and bias trade off to predict well but avoid overfitting.

2.4. Technical discussions and improvements

Previous section we introduce our framework to deal with this task. Before that we tried different models and ways to combine. Now let's look at what we tried and how we get our final model.

1. Firstly, baseline code was provided to us. With only GP-NAS model [11], the final score is about 0.67 (all the score we calculated is Kendalltau).

2. Besides, we considered other machine learning models such as linear regression, random forest model, kernel ridge regression, tree model,light Gradient Boosting [9], XGBoost [1],Cat Gradient Boosting [3] and so on. We find Gradient Boosting or other Boosting based algorithm [6] have best average result. When we firstly try it, we get score about 0.73. after tunning parameters, we get 0.78. This experiment let us narrowed our submodel pool inside boosting based algorithm.

Boosting based algorithm is a tree-based ensemble model that ensemble of several weak learners which usually are decision trees. Decision trees are relatively weak on their own usually solely on yes/no questions. For boosting based algorithm, Each tree's goal is not to maximize its own utility function on a randomly delegated subset of data, but to fit on residual of previous tree which performances relatively well in non-linear and complex problem.

Lots of experiment showed boosting based algorithm has these advantages compare with others: usually provides predictive accuracy that hardly be beaten. Very flexible that can optimize on different loss functions and provides several hyper parameter tuning options. Most important, no requirement of data pre processing, not sensitive to distribution of data and missing point.

3. As our problem is multi-task, could we train different task at same time? We consider Multivariate Gradient Boosting based on MultiRMSE [3] from package catboost.

$$MultiRMSE = \sqrt{\frac{\sum_{i=1}^{N}\sum_{d=1}^{dim}(a_{i,d}-t_{i,d})^2 w_i}{\sum_{i=1}^{N} w_i}} \tag{5}$$

However, the result is even worse than univariate gradient boosting method. The reason maybe as follow: different task is so different that should consider different parameters; there are no obvious lead or lag relationship between different task; we can't reduce the noise in this multivariate Gradient Boosting algorithm effectively.

4. As our training sample size is so small, in order to avoid over fitting, we try to ensemble our models. We choose GBRT [6],HISTGB [9],CATGB [3],XGBoost [1],lightGB [9] as our sub models. The obvious way is averaging all the result, this result improved to about 0.79.

5. After naive averaging, then we think about stacking gradient boosting group of model by a final estimator. After several experiment and comparision. We pick GP-NAS as our final regressor. Stacking is a general procedure where final estimator is trained to combine the individual learners. Stacking allows us to use the strength of each individual estimator by using their output as input of a final estimator. In order to aviod overfitting, we use full training dataset only in sub model training, but training our final model with cross validation. We use K-1 fold to predict submodel and remain 1 fold as validate data and repreat for each K. As sklearn already provide us this regressor [14], we modified GP-NAS [11] as a sklearn API. Now we get result about 0.792.

6. Until now our stack model still use same parameters for different task such as loss function, learning rate and depth of model. Then we consider select different loss function, learning rate and depth for different task as we assume different dataset and task have different signal noise ratio and different features. Meanwhile we add two more GBRT, CATGB submodel with different loss funtion(huber or MSE) which also give us improvement. This round we get result about 0.798.

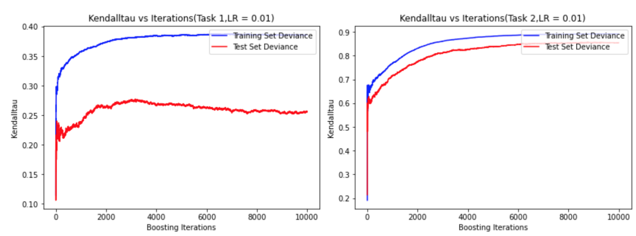

Figure 3. Task 1 and Task 2 Kendalltau with different boosting iter

Fig. 3 shows different task has total different signal to noise ratio and feature even with same model. In this example, we use same parameter and model for Task 1 and Task 2 which show totally different phenomenon.

7. Also we modified and tunning our final estimator GP-NAS, such as ridge parameter and estimator prior. Instead of identity matrix, we choose inv(X.T*X) as our prior covariance matrix like equation (9). We get final score 0.7991 in leaderboard A and 0.79849 in leaderboard B.

$$Mean(\beta|X,Y) = (X^T X)^{-1} X^T Y \tag{6}$$

$$Var(\beta|X,Y) = \sigma^2 (X^T X)^{-1} \tag{7}$$

Eq. (6)Eq. (7) are mean prior and variance prior of estimator.

Model	Kendalltau
GP-NAS Baseline	0.67
Gradient Boosting	0.78
Multivarate Gradient Boosting	0.787
Averaging 5 ensemble algorithm	0.79
Stacked 5 algo with same params of tasks	0.792
Stacked 7 algo with diff params of tasks	0.798
Stacked 7 algo with inverse cov as prior	0.7991

Table 1. Result comparsion

Tab. 1 show the model we tried and their Kendalltau score.

3. Conclusion

In this paper, we explain how we pre processing raw data, the model we tried, and the detail of last verison of model framework. It shows stacking several ensemble models with GP-NAS give us a quite great result. As we consider cross validation during fitting, the prediction result by testing data is also considerably good. As time limitation, there are lots of idea we haven't deeply go through, such as improving the multivariate Gradient Boosting, stacking multivariate ensemble models, different kernel and prior in GP-NAS which maybe consider in the future.

References

[1] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016. 3

[2] Allen Chieng Hoon Choong and Nung Kion Lee. Evaluation of convolutionary neural networks modeling of dna sequences using ordinal versus one-hot encoding method. In *2017 International Conference on Computer and Drone Applications (IConDA)*, pages 60–65. IEEE, 2017. 2

[3] Anna Veronika Dorogush, Vasily Ershov, and Andrey Gulin. Catboost: gradient boosting with categorical features support. *arXiv preprint arXiv:1810.11363*, 2018. 3

[4] Saso Džeroski and Bernard Ženko. Is combining classifiers with stacking better than selecting the best one? *Machine learning*, 54(3):255–273, 2004. 2

[5] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *The Journal of Machine Learning Research*, 20(1):1997–2017, 2019. 1

[6] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001. 3

[7] Benyamin Ghojogh and Mark Crowley. The theory behind overfitting, cross validation, regularization, bagging, and boosting: tutorial. *arXiv preprint arXiv:1905.12787*, 2019. 2

[8] Trevor Hastie, Robert Tibshirani, Jerome H Friedman, and Jerome H Friedman. *The elements of statistical learning: data mining, inference, and prediction*, volume 2. Springer, 2009. 2

[9] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems*, 30, 2017. 3

[10] Ron Kohavi et al. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Ijcai*, volume 14, pages 1137–1145. Montreal, Canada, 1995. 2

[11] Zhihang Li, Teng Xi, Jiankang Deng, Gang Zhang, Shengzhao Wen, and Ran He. Gp-nas: Gaussian process based neural architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11933–11942, 2020. 1, 3

[12] Kedar Potdar, Taher S Pardawala, and Chinmay D Pai. A comparative study of categorical variable encoding techniques for neural network classifiers. *International journal of computer applications*, 175(4):7–9, 2017. 1

[13] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, volume 33, pages 4780–4789, 2019. 1

[14] David H Wolpert. Stacked generalization. *Neural networks*, 5(2):241–259, 1992. 2, 3

[15] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710, 2018. 1