Copula for Instance-wise Feature Selection and Ranking (Supplementary Material)

Hanyu Peng, Guanhua Fang, Ping Li

Cognitive Computing Lab
Baidu Research
No.10 Xibeiwang East Road, Beijing 100193, China
10900 NE 8th St. Bellevue, Washington 98004, USA
{hanyu.peng0510, fanggh2018, pingli98}@gmail.com

1 ANALYSIS WITH RESPECT TO COMPLEXITY

The computational complexity can be estimated as $O(d^3)$, where d is the number of features. Since we have to do matrix decomposition. To reduce the computational complexity even further, additional techniques can be applied. As one of the proposed solutions in the article Lee et al. [2022], we can threshold the correlation matrix $\mathbf{\Sigma}$, and grouping features based on agglomerative clustering using the correlation matrix as the similarity measure. This reduces the number of computations required by conducting block-wise matrix multiplication, which scales quadratically with respect to the largest block size (i.e., the number of features grouped in the largest block). By maintaining the correlation structure within each group, the generated gates for features within the same group can be used to select features that are highly correlated with the target variable, while reducing the computational complexity of the algorithm. If the largest block size remains the same, the complexity of generating the correlated gate vectors will only increase linearly with the feature dimension (since the number of blocks would increase linearly).

The CPU calculation time for copula-based feature selection also depends on the implementation, hardware, and software used. To reduce CPU calculation time, some techniques such as parallelization and approximation methods can be used. For example, using GPU computation can significantly speed up feature selection process, particularly for large datasets.

In summary, copula-based feature selection can be computationally expensive, particularly for large datasets with many features. The computational complexity can be estimated as $O(d^3)$, and the CPU calculation time depends on the implementation, hardware, and software used. However, optimization techniques such as parallelization and approximation methods can be used to reduce the computational complexity and CPU calculation time.

2 VISUALIZATION OF CHOSEN PIXELS

Now, we present a visual representation of the top-120 features with the most significant values in the $\boldsymbol{\alpha}$ for each sample on the MNIST dataset. For each image, we display the most informative features. As is evident from Figure 1, we can observe a clear segmentation in the shape of the classification object, indicating that our method is capable of identifying the most important and meaningful features. This visualization demonstrates the remarkable interpretive power of our method.

3 NEURAL REPARAMETERIZATION OF CORRELATED UNIFORM NOISE

In Algorithm 1, we present the workflow to produce correlated uniform noise $\boldsymbol{u} \in \mathbb{R}^d$. Here, we offer a further elucidation of the neural reparameterization of it. We denote the output of the hidden layer in ChoiceNet as $f(\boldsymbol{w}; \boldsymbol{x})$ as $\boldsymbol{O}_h \in \mathbb{R}^{h_c}$, and the weight parameter of the layer which produces σ as $\boldsymbol{W}_\sigma \in \mathbb{R}^{h_c \times d}$. Additionally, $\boldsymbol{W_L}$ stands for the weight parameter of the layer which generates matrix $\boldsymbol{L}$. If we opt for the low-rank approximation, the shape of $\boldsymbol{W_L}$ is $h_c \times (p \times d)$, otherwise $h_c \times (d \times d)$. The concrete implementation via neural reparameterization is provided as follows:

Accepted for the 39th Conference on Uncertainty in Artificial Intelligence (UAI 2023).

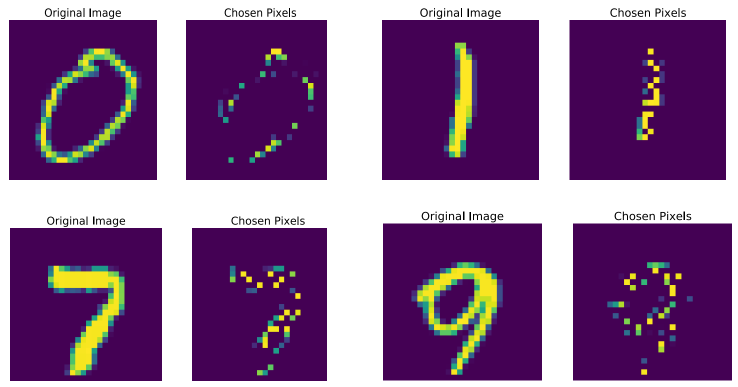

Figure 1: Visualization of chosen pixels, we can observe that our method can select meaning and intuitive image features.

Algorithm 1: Neural Reparameterization of Correlated Uniform Noise

```
Input: Activation $O_h$ of the hidden layer in ChoiceNet $f(\boldsymbol{w};\boldsymbol{x})$, weight parameters $\boldsymbol{W}_\sigma$ and $\boldsymbol{W}_{\boldsymbol{L}}$
Output: Correlated uniform noise $\boldsymbol{u}$.
1 $\boldsymbol{L} = \text{ReLU}(\boldsymbol{O}_h \boldsymbol{W}_{\boldsymbol{L}})$;
2 $\sigma = \text{Tanh}(\boldsymbol{O}_h \boldsymbol{W}_{\boldsymbol{\sigma}})$ ;
3 Obtain the covariance matrix via low-rank approximation $\boldsymbol{\Sigma} = \boldsymbol{L}^T\boldsymbol{L} + \sigma^2\boldsymbol{I}$ or full-rank approximation $\boldsymbol{\Sigma} = \boldsymbol{L}^T\boldsymbol{L}$;
4 Perform Cholesky factorization on $\boldsymbol{\Sigma}$ to get Cholesky factor $\boldsymbol{V}$ ;
5 Generate a Gaussian noise vector $\boldsymbol{\zeta}$ from standard normal distribution $\boldsymbol{\zeta} \sim \mathcal{N}(0, \boldsymbol{I})$;
6 Calculate the Gaussian vector $\boldsymbol{q} = \boldsymbol{V}\boldsymbol{\zeta}$;
7 Apply Gaussian copula to obtain $\boldsymbol{u}$ as $u_i = \Phi_{\boldsymbol{R}}(q_i), \forall i = 1, \ldots, d$;
```

4 DETAILS OF BASELINE METHODS

The summary of some baseline methods of binary feature selection are as follows.

- **Xgboost** We used the Gini index as the splitting criterion. Specifically, we used the DecisionTreeClassifier function from the scikit-learn library with default parameters. The DecisionTreeClassifier function builds a decision tree by recursively splitting the data based on the feature with the highest Gini importance score. The Gini importance score measures the total reduction of impurity brought by a feature in the decision tree, and features with higher Gini importance scores are considered more important. After building the decision tree, we selected the top-k features with the highest Gini importance scores as the selected features. The value of k was determined using the same experimental setup and evaluation protocol as our proposed method.

  LASSO In the paper, LASSO is a linear regression-based feature selection method, where we used L_1 regularization to encourage sparsity in the model coefficients. Specifically, we used the LogisticRegression function from the scikit-learn library with L_1 penalty and default parameters. The L_1 penalty in the LogisticRegression function encourages sparsity in the model coefficients by adding a penalty term to the loss function that is proportional to the absolute value of the coefficients. This penalty term forces the model to select only a subset of the most important features, effectively performing feature selection. After fitting the logistic regression model with L_1 regularization, we selected the top-k features with the highest absolute coefficients as the selected features. The value of k was determined using the same experimental setup and evaluation protocol as our proposed method, including a hold-out strategy and 5-fold cross-validation for hyperparameter tuning.

- **L2X** [Chen et al., 2018] introduces a new model interpretation way from the feature selection perspective, it aims to learn a feature selection network that maximizes the mutual information between selected feature subsets and corresponding outputs, we use the official implementation to evaluate the result from the link:
  https://github.com/Jianbo-Lab/L2X

- **INVASE** [Yoon et al., 2019] proposes an instance-wise feature selection algorithm based on the actor-critic framework to selects most relevant features that minimizes the Kullback-Leibler (KL) divergence between full conditional

distribution and suppressed feature distribution, we use the official implementation to evaluate the result from the link:
https://github.com/jsyoon0823/INVASE

- **LIME** [Ribeiro et al., 2016] is a model-agnostic explanation algorithm, it learns an interpretable model locally in a non-redundant and faithful manner by formulating the task as a submodular optimization problem, we use the official implementation to evaluate the result from the link:
  https://github.com/marcotcr/lime
- **Shap** [Lundberg and Lee, 2017] proposes a novel framework that employs the shapely value to calculate the feature importance, we use the official implementation to evaluate the result from the link:
  https://github.com/slundberg/shap
- **Knockoff** [Barber and Candès, 2015] aims to find which variables are important to the response by comparison between knock-off variables and original variables. we use the official implementation to evaluate the result from the link:
  http://web.stanford.edu/group/candes/knockoffs/software/knockoff/

In our experiments, we used a neural network as the predictive model in conjunction with Shap and LIME. The neural network had the same size and architecture as the one in our proposed method, in order to ensure fairness and exclude the influence of other factors such as network architecture and size. Regarding the cutoffs, we used a threshold of 0.5 for the neural network to predict the binary class labels. We applied the same threshold when generating the Shap and LIME explanations, to ensure consistency in the interpretation of feature importance across different methods.

The description of some baseline methods of top-k feature ranking are as follows:

- **STG** [Yamada et al., 2020] provides a novel algorithm that depends on the Gaussian-based relaxation of the Bernoulli distribution to select relevant features, we use the official implementation to evaluate the result from the link:
  https://github.com/runopti/stg
- **CAE** [Abid et al., 2019] introduces an auto-encoder architecture for global feature selection while reconstructing the input, we use the official implementation to evaluate the result from the link:
  https://github.com/mfbalin/Concrete-Autoencoders

4.1 DISCUSSIONS OF BASELINE METHODS

Our method is capable of discerning pertinent features on a global scale (Syn1, Syn2, and Syn3) as well as on an individual basis (Syn4, Syn5, and Syn6). Notably, our approach surpasses prior neural network-based approaches (INVASE, L2X) in terms of individual performance, with the improvement being more pronounced in Syn4 and Syn5 than in Syn6. Random Forests (RFs) can select global features, thus performing better on Syn1, Syn2, and Syn3, but not as well on Syn4, Syn5, and Syn6. Shapley-based methods calculate the variable importance to elucidate the linear dependency for each sample; however, it is difficult to capture the non-linear relationships in synthetic data, thus rendering it less effective in high-dimensional data. LIME utilizes simple functions to interpret complexity locally; however, it can only explain the particular instance, meaning that it may not be able to accommodate for unseen instances. Knockoff filters features according to certain criteria, yet there is no assurance that this metric is uniquely optimal, thus its performance may vary across datasets.

5 IMPLEMENTATION DETAILS

5.1 SYNTHETIC DATASETS

We have employed the same datasets and network structure as in Yoon et al. [2019]. For all experiments on the six synthetic datasets, the hyperparameters h_c and h_p were set to 100 and 200, respectively. The activation function of the last layer was a sigmoid. The entire network (ChoiceNet and PredictNet) was trained for 1,000 epochs using Adam with a batch size of 1,000, a weight decay of 0.001, and coefficients of 0.9 and 0.999 for computing running averages of gradient and its square. The constant learning rate was set to 0.0001, and the temperature parameter t was set to either 3 or 5. Finally, cross-validation was employed to tune the hyperparameter λ.

5.2 REAL DATASETS

When performing top-k feature ranking, h_c and h_p were both set to 16. In all the experiments, we discovered that the Cholesky decomposition to obtain the Cholesky factor matrix $\boldsymbol{L}$ was time-consuming, so we employed the full-rank scheme. We trained the network for 100 epochs using Adam, with coefficients used for computing running averages of gradient set to 0.9 and 0.999. The constant learning rate was set to 0.001 for Fashion-MNIST and MNIST, and 0.0001 for ISOLET. The batch size was set to 1,000 for MNIST and Fashion-MNIST, and 64 for ISOLET, while the temperature t was set to 1 for all the experiments evaluated on real datasets. All the parameters of the neural networks were randomly initialized. For a fair comparison with baseline methods based on neural networks such as INVASE, L2X, STG, and CAE, we employed the same hyper-parameters and architecture to evaluate the performance.

References

Abubakar Abid, Muhammad Fatih Balin, and James Zou. Concrete autoencoders for differentiable feature selection and reconstruction. *arXiv preprint arXiv:1901.09346*, 2019.

Rina Foygel Barber and Emmanuel J Candès. Controlling the false discovery rate via knockoffs. *The Annals of Statistics*, 43(5):2055–2085, 2015.

Jianbo Chen, Le Song, Martin J. Wainwright, and Michael I. Jordan. Learning to explain: An information-theoretic perspective on model interpretation. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, pages 882–891, Stockholmsmässan, Stockholm, Sweden, 2018.

Changhee Lee, Fergus Imrie, and Mihaela van der Schaar. Self-supervision enhanced feature selection with correlated gates. In *International Conference on Learning Representations*, 2022.

Scott M. Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *Advances in Neural Information Processing Systems (NIPS)*, pages 4765–4774, Long Beach, CA, 2017.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. “why should I trust you?”: Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 1135–1144, San Francisco, CA, 2016.

Yutaro Yamada, Ofir Lindenbaum, Sahand Negahban, and Yuval Kluger. Feature selection using stochastic gates. In *Proceedings of the 37th International Conference on Machine Learning (ICML)*, pages 10648–10659, Virtual Event, 2020.

Jinsung Yoon, James Jordon, and Mihaela van der Schaar. INVASE: instance-wise variable selection using neural networks. In *Proceedings of the 7th International Conference on Learning Representations (ICLR)*, 2019.