Copycat: Mimicking Tree Boosters with Neural Nets Using SHAP Values

Dana Cohen Tomer Ronen Tomer Shanny

Abstract

Neural nets are state of the art on image and textual data. However, tree boosting algorithms reign supreme when tabular data is considered. In this paper we suggest a novel technique of training a Student network which learns from a Teacher XGBoost model, mimicking SHAP values as a surrogate to the rich feature representation usually mimicked when both Student and Teacher models are neural networks. Code is available at https://github.com/DanaCohen95/copycat

1. Introduction

Diagnosing a patient, picking a candidate for a job or deciding on a destination for a trip - these are some few examples of problems in which we make decisions based on tabular information. Such problems are of great interest among data scientists, who try to train models to make the best decisions for this kind of problems. Finding a good approach for handling this family of problems has great significance, and therefore machine learning on tabular data is widely studied.

Two model type we examine in this paper are neural-networks and tree boosting algorithms. It is widely known that neural networks are state of the art on image data and textual data, while tree boosting algorithms are much better when tabular data is considered. However, it becomes more complicated when facing holistic datasets that combine images or textual data with tabular data.

One could concatenate the tabular features with a neural net-induced feature vector as data for a tree-boosting model, but this seems sub-optimal compared to training individual nets for each different data type, wiring them together and then jointly fine-tuning them, as is very successfully done for the image captioning problem.

In the following paper we suggest a way to train a neural network that attempts to gain results as good as a tree boosting algorithm on tabular datasets, by directly mimicking the tree booster. This tree-boosting-motivated neural net would hopefully also gain other advantages of tree boosters, such as greater stability. Besides from being a good ML model in itself, this net can also be a good building-block for handling problems with holistic datasets.

Naturally, due to the fact that neural-networks and tree-boosting algorithms are two different kind of machine learning models, it is clear that their performances will differ not only in the accuracy of their classifications but in their generalization and stability characteristics as well. Therefore, testing and comparing our hybrid model with NN and tree-boosting references, should include generalization and stability tests we will describe in details bellow.

2. Problem Statement

As presented above, our goal in this paper is to develop a NN-model that learns tabular data.

2.1. Data Properties

In this paper we focus on classification problems based on tabular features. Tabular features are usually either binary, numerical or categorical(variables that can take on one of a limited number of possible values).

There are two datasets on which we based our study:

- Costa Rican household poverty level prediction - A data set that contains a set of features describing a variety of financial properties regarding a household. The objective is to classify each household into one of four poverty levels.
- Rental Listing Inquiries - A dataset that contains a set of features describing house characteristics such as number of bedrooms, location etc. The objective is to classify each house into one of three value levels.

The processing started in adapting each of the datasets by removing sparsed features and by balancing over represented classes.

2.2. Models

We trained and compared three types of models:

- **Vanilla-NN** - basic neural-network with 2 hidden layers and tabular crossentropy loss.

- **XGBoost** - a parallel tree boosting algorithm with 30 trees of depth 10.
- **Copycat-NN** - a neural network with equivalent capacity to the Vanilla-NN, which aims to mimic feature importance of the XGBoost on each sample, in addition to achieving good results on the train set. This model will be discussed in depth in section 4.

Each of the above models was trained on the same training set and was evaluated on the same test set.

2.3. Evaluation

In this paper we decided that the evaluation of our models will be based on two main criteria:

2.3.1. Quality Measures

In order to evaluate the quality of our models, we used two popular evaluation measures: Accuracy and Mean F_1 score. Accuracy, the percentage of correct predictions, is an extremely common measure, but it can be misleading since it doesnt take into account class imbalance, and since it doesn't distinguish between False Positives and False Negatives. Therefore, we prefer the Mean F_1 score, which is the arithmetic mean of the F_1 scores of all classes, where each F_1 score is the harmonic mean of the recall and the precision for the specific class.

$$MeanF_1 = \frac{1}{\#classes} \sum_{cls} F_1^{(cls)} \tag{1}$$

2.3.2. Generalization Measures

In addition to the measurements presented above, our evaluation takes into account some generalization measures that express the quality of each model:

1. **Distribution Scale**
   We are interested to find out whether the results of each model are stable upon multiple repetitions of the training process. Therefore, we trained and tested every model 100 times and created histograms for each model based on its results. From these histograms we extracted the means and the STDs of all models.
2. **Train-Test Accuracy Gap**
   Another interesting property of a model is the difference in its quality score (Accuracy or MeanF$_1$) on the train set and on the test set. This gap is an indicator of the overfitting property of the model.
3. **Classification Instability**
   In order to evaluate the stability of each model we split the train set into two. We trained each model over the two training sets and tested it on the same test set. The two results were used in order to calculate two properties:
   - **Percentage of Disagreement** - The percentage of the samples on which the two trained instances disagreed.
   - **Jensen-Shannon Divergence** - A method of measuring the dissimilarity between two probability distributions. The Jensen-Shannon Divergence is a symmetrized and smoothed version of the Kullback-Leibler divergence D, given by

$$JSD(P||Q) = \frac{1}{2}\Big(D(P||M)+D(Q||M)\Big) \tag{2}$$

$$M = \frac{1}{2}(P+Q)$$

3. Related Work

Student-NN models are often intended to mimic a well-trained heavy Teacher-NN model, using a smaller and faster architecture. This way, a portable student network with significantly fewer parameters and faster runtime can achieve comparable results to that of the teacher network. The Student-Teacher approach makes the assumption that such smaller architectures, when trained independently, tend to achieve worse results than those achieved by mimicking teacher nets.

Several studies have been conducted in order to find good feature representations to mimic, and good ways to mimic them. SNNs (Shallow Neural Networks) (Ba & Caruana, 2014) minimize the euclidean distance between the features extracted from the last layer of the teacher and student nets, and FitNet (Romero et al., 2014) minimizes the difference between the features extracted from the middle layers of student and teacher networks, using linear projection layers to train the narrower student. KD (Hinton et al., 2015) uses "knowledge distillation" and encourages the student to match the full logits distribution of the teacher. WideResNet (Zagoruyko & Komodakis, 2016) matches the attention map of the student and the teacher at the end of each residual stage, and SobolevTraining (Czarnecki et al., 2017) optimizes the student to not only approximate the teacher's outputs but also the teacher's derivatives.

4. Copycat-NN approach

4.1. SHAP values

SHAP (SHapley Additive exPlanations)(Lundberg & Lee, 2017) is a unified approach to explain the output of machine learning models. Lunberg et al. present three conditions for an explanation model to be "good":

1. **Local Accuracy**: The feature importances must sum to the model's prediction.
2. **Missingness**: Missing features have 0 importance.
3. **Consistency**: If some feature affects model A more than model B, its importance for model A should be bigger.

SHAP takes into account these three conditions and attempts to serve as a unified framework for interpreting predictions.

SHAP values attribute to each feature the change in the expected-model-prediction when conditioning on this feature. They address the value $E(f(z))$, that would be predicted if we did not know any feature, to the model's output $f(x)$.

4.2. XGBoost's SHAP Teacher & NN Student

For student-teacher approach to be effective, student nets need some kind of rich feature representation to mimic. However, generally, we cannot extract explicit features from tree boosters. As a substitute, we suggest to mimic the SHAP values of XGBoost model, calculated with the tree-booster model (Lundberg et al., 2019), as they form a decomposition of the model score according to the original tabular features, thus hinting at the way the tree-booster model interprets the data.

4.3. Model Description

The model we suggest is a 2 fully connected hidden layer NN that predicts the SHAP values of a XGBoost model on the training set. The output of the model is a tensor of length $num\_classes \cdot num\_shap\_features$.

The loss is the average of two objective functions:

1. The SHAP loss, which is the MSE of the output of the model and of the ground truth SHAP values calculated on the training data of the trained XGBoost model.
2. The cross entropy between the actual labels and the labels which are calculated from the output of the model.

The labels are calculated with a parameter-less layer using **shaps_to_probs** function, which is a function that calculates each class probability according to the following equation:

$$p_{cls} = \frac{exp(logits_{cls})}{\sum_{cls \in classes} exp(logits_{cls})} \tag{3}$$

$$logits_{cls} = expected\_logits_{cls} + \sum_{f \in features} \widehat{shaps}^{(f)}_{cls}$$

The architecture of the model can be seen in Fig. 1.

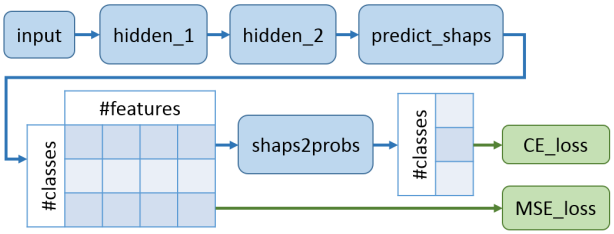

Figure 1. Copycat NN Architecture.
hidden_1, hidden_2 and predict_shaps are dense layers. shaps_to_probs is a custom layer. The Mean Square Error regression loss is computed directly on the predicted SHPAPs, and the Cross Entropy Loss is computed on the labels and the shap-induced predicted class probabilities.

4.4. Matching NN's capacities

In order to make sure that our comparison between Copycat-NN and Vanilla-NN is balanced, we want both the NNs to have the same capacity. For simple nets, the capacity is a function of the number of layers and the number of parameters. One option is to have both nets using the same architecture, while turning off their regression loss. However, this creates a weird classification net architecture, which isn't exactly Vanilla. Another option is to strengthen the Vanilla NN with more parameters or more layers. We decided that more layers is too much of an advantage, so we changed the number of neurons in the last layer of the Vanilla net.

$$Vanilla\_params = n\_features \cdot h_1 + (h_1 + 1) \cdot h_2 + (h_2 + 1) \cdot h_3 + (h_3 + 1) \cdot n\_classes$$

$$student\_params = n\_features \cdot h_1 + (h_1 + 1) \cdot h_2 + (h_2 + 1) \cdot h'_3 + (h'_3 + 1) \cdot n\_classes \cdot n\_features$$

In order to match the capacities we chose h'_3 such that the model would have the same number of parameters.

4.5. Reducing Overfitting By Partial Mimicking

Tabular datasets are often consisted of many features compared to the number of samples. Therefore, in order to avoid overfitting and improve our model's convergence, we decided to mimic only a limited number of features which have the largest SHAP values when averaging over all the training samples and over all the classes.

5. results

5.1. Quality Measures

We compared the models' qualities based on two quality scores: *Accuracy* and *MeanF*$_1$. Table 1 shows the results of the trained models on the test set.

Table 1. Quality Comparison
Average of 100 cross-validation splits ± 95% Confidence Interval

MODEL	ACCURACY	MEANF$_1$
XGBOOST	0.766±0.023	0.763±0.024
COPYCAT-NN	0.713±0.034	0.708±0.036
VANILLA-NN	0.667±0.040	0.659±0.045

As shown in Table 1, XGBoost shows the best results while Vanilla-NN shows the worst test score results. An interesting result is the performance Copycat-NN shows. From Table 1 it is clear that Copycat-NN is more accurate over the test set than an ordinary NN. Furthermore, its results are closer to those of the XGBoost model than to those of the Vanilla-NN. We conclude that the Copycat model indeed gains an advantage from mimicking XGBoost's SHAP values.

5.2. Generalization

In addition to the quality measures presented above, we base our comparison on several generalization measures.

We trained and tested our models multiple times using Stratified Cross-Validation splits. The statistics we get thanks to this random repetition help us gain a deeper understanding of the models.

5.2.1. SCORE DISTRIBUTION SCALE

To understand the **stability** of the different models, we examined the scale of the models' score distribution over both the train set and the test set. In the following subsection we will focus on the MeanF$_1$ score.

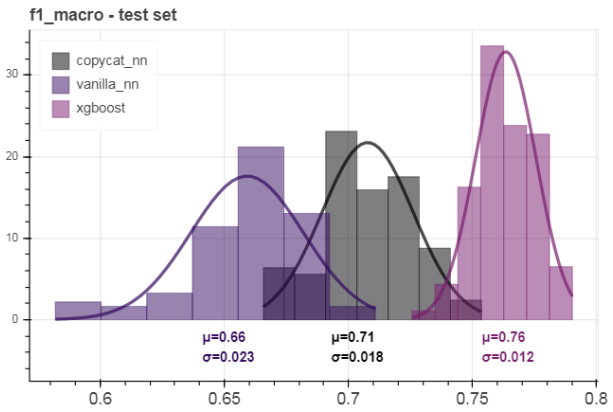

Figure 2. Prediction results of the models on the **test set**.
These histograms were created from the results of 100 cross-validation splits.

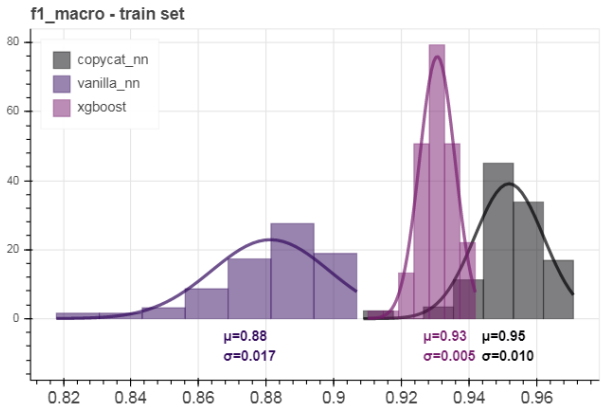

Figure 3. Prediction results of the models on the **train set**.
These histograms were created from the results of 100 cross-validation splits.

Table 2. MeanF$_1$ Standard Deviation over train and test sets

	XGBOOST	COPYCAT-NN	VANILLA-NN
TEST	0.012	0.018	0.023
TRAIN	0.005	0.010	0.017

Just like with the quality score, here the XGBoost shows the best results as well. Its results are the most consistent for both the train set (Fig. 3) and the test set (Fig. 2). In addition, the results in Table 2 suggest that mimicking SHAP values leads to more consistent results than designing an ordinary NN.

An interesting result is that Copycat-NN shows better results on the train set than those of the XGBoost model (Fig. 3). This is reflected even more sharply when examining the Train-Test Score Gap - we will relate to this phenomenon in the following subsection.

5.2.2. TRAIN-TEST SCORE GAP

In order to assess the overfitting property of our models, the differences in the results of each model on the train set and the test set should be taken into account. We split our data 100 times using Stratified Cross-Validation, and for each split calculated the difference in MeanF$_1$ score.

Table 3. Train-Test MeanF$_1$ Score Difference

XGBOOST	COPYCAT-NN	VANILLA-NN
0.167±0.024	0.244±0.032	0.222±0.038

A good indication for how much a model is overfitting is

the difference in its performance on the train set and the test set. According to Table 3 it is clear that XGBoost overfits less than both the NN models. This can be explained by the fact that we trained the XGBoost model with its built-in L2 regularization while we didn't use any regularization in our NN models.

Another observation worth explaining is that Vanilla-NN overfits less than Copycat-NN. We believe the cause of this phenomenon is the fact that Copycat-NN has much more fine-grained supervision. Copycat-NN learns to optimize more variables than the Vanilla-NN - in addition to the class labels that both NNs recieve, Copycat-NN also learns to mimic high-dimension tensors of SHAP values, via the additional MSE loss.

5.2.3. Classification Instability

Our third means of evaluating the generalization and stability of our models is splitting the **training set** into two, training each model twice and testing each instance on a shared test set. For a stable algorithm, we expect both trained instances to have similar performance. We compared the results of both instances according to the Percentage of Disagreement (*PoD*) and the Jensen-Shannon Divergence (*JSD*) measures.

Table 4. Classification Instability

MODEL	*PoD*	*JSD*
XGBOOST	0.341±0.040	0.238±0.017
COPYCAT-NN	0.445±0.058	0.282±0.027
VANILLA-NN	0.465±0.041	0.400±0.026

Table 4 is in line with the results we saw so far. Both the *PoD* and the *JSD* values of the XGBoost model are the smallest, then comes the Copycat-NN, and the Vanilla-NN shows the greatest disagreement.

It is worth mentioning that while the *PoD* of the Copycat-NN is closer to that of the Vanilla-NN, the *JSD* of the Copycat-NN is much closer to that of XGBoost. *JSD* is a richer measure that expresses the dissimilarity of two probability distributions by using all class probabilities in its calculations, while *PoD* takes into account only the max-probability class. Therefore, the fact that Copycat-NN is closer to XGBoost according to *JSD* is a more significant result for our analysis.

5.3. Compatibility of Predicted SHAPs And Features

The predicted SHAP values our Copycat net produces are optimized to mimic the actual SHAP values produced by the XGBoost model. However, we have no explicit guarantee that these values actually represent feature importances. We suggest a way to quantify compatibility between predicted SHAP values and the corresponding features they supposedly represent.

Theoretically, when changing the value of a specific input feature, we would like the corresponding channel in the predicted SHAP tensor to be most affected. If, however, our predicted SHAP values don't actually correspond to specific feature importances, we would expect an approximately uniform change in the predicted SHAP tensor. Let X be an input matrix and f be a specific feature. Given a modified input matrix X' that is identical to X except for a constant additive offset in feature f, we define the *Relative Predicted SHAP Change* to be the following:

$$change^{(f)} = \sum_{i=1}^{n}\sum_{cls}\left|PredShap(X)^{(f)}_{i,cls} - PredShap(X')^{(f)}_{i,cls}\right| \tag{4}$$

$$rel\_change^{(f)} = \frac{change^{(f)}}{\sum_{f'} change^{(f')}}$$

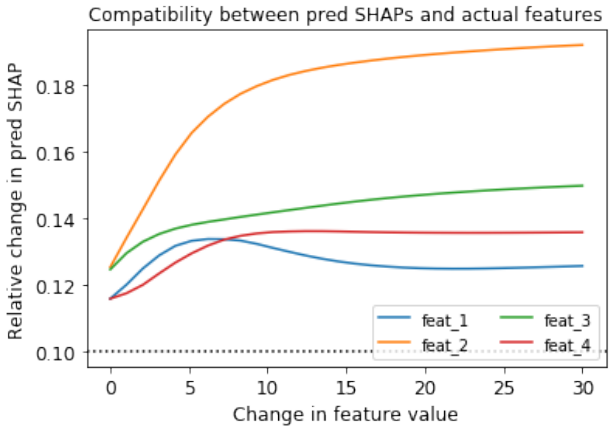

Figure 4. For each of the 4 most important features, we show the relative change in the corresponding SHAP channel when changing the feature value. Random compatibility is 0.1 since our Copycat model predicts 10 feature importances.

5.4. Another Dataset

In contrast to the promising results we got on the Costa Rica data, re-executing the research on another dataset (*Rental Listing Inquiries*) showed unclear results.

In *Figure 5* we present histograms of the predictions results of all trained models on the test set:

On the *Rental Listing Inquiries* dataset the averages of the results of all models are the same. As a result, trying extracting interesting insights from it was almost impossible.

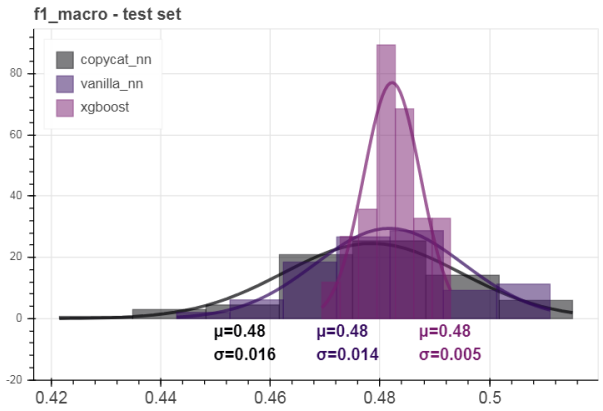

Figure 5. Prediction results of the models on the **test set**. These histograms were created from the results of 100 cross-validation splits.

However, these results strengthen the fact that accuracy doesn't "comes from thin air" - it is impossible to train a student NN to outwit its teacher, the XGBoost model. On the dateset discussed, XGBoost and Vanilla-NN showed the same results, therefore it is clear that there is no gap to reduce by utilizing the student-teacher approach.

In addition, from the results in *Figure 5* (as for all other results in this project) it is clear that XGBoost is has the smallest STD value, while both the NNs showed the same performance in this field. From this fact we can learn that some characteristics cannot be mimicked by the student-teacher approach we presented in this paper.

6. Conclusions and Further Work

6.1. Conclusions

As mentioned in the beginning, our goal in this project was to present a new approach to design NN for handling tabular data.

From the analysis demonstrated throughout the paper, we can summarize our results with the following:

- XGBoost is the best method for handling tabular data from the methods referred to in this project.
- Training a NN to mimic SHAP values of XGBoost shows improves its quality scores and generalization results. However this fact was found to be true only if XGBoost shows significantly better results than the NN on the dataset.
- There is a definite correspondence between the student's predicted SHAPs and the SHAPs were extracted from the XGBoost model.
- SHAP-based NN overfits more than both XGBoost and ordinary NN.

To conclude, in this paper we described a new approach for designing a NN to handle tabular data, based on knowledge distillation from XGBoost model. It is shown that this approach gains some good results. However, the approach was found to be effective only in cases in which the XGBoost model, the teacher, was much better than the Vanilla-NN. Therefore, in order to verify that Copycat-NN is a good approach to handle tabular data in general, more work should be done on datasets where there is a significant advantage to the XGBoost.

6.2. Further Work

As we've seen, our Copycat network indeed achieves better results than a Vanilla neural network, but this improvement in quality comes with the price of overfitting - a significantly bigger gap between train and test scores. It would be interesting to experiment with various kinds of net regularization, such as weight decay, dropout and batch normalization. These methods are supposed to mitigate overfitting, and hopefully wouldn't affect the test scores.

A notable advantage tree boosting algorithms have over Vanilla neural networks is their ability to deal well with difficult data properties: small amounts of labeled data, missing values, class imbalance, etc. Since the Copycat model aims to mimic XGBoost, it would be interesting to see if it naturally obtains this robustness.

Assuming the Copycat method truly creates better tabular data nets, it holds promise for holistic datasets. Holistic problems are consisted of multiple data types, and are often solved by wiring together different kinds of pretrained neural nets and jointly optimizing them for the specific problem using a multi-task loss. For example, image captioning architectures are often composed of a CNN for images and an LSTM for text, wired together. Other holistic problems, that incorporate tabular data with other data types, are therefore in need of a strong tabular neural network - this is exactly where Copycat comes in handy, and where both Vanilla NNs and XGBoost models fall short.

References

Ba, J. and Caruana, R. Do deep nets really need to be deep? In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N. D., and Weinberger, K. Q. (eds.), *Advances in Neural Information Processing Systems 27*, pp. 2654–2662. Curran Associates, Inc., 2014.

Czarnecki, W., Osindero, S., Jaderberg, M., Swirszcz, G., and Pascanu, R. Sobolev training for neural networks.

Advances in Neural Information Processing Systems, 06 2017.

Hinton, G., Vinyals, O., and Dean, J. Distilling the knowledge in a neural network. *arXiv*, 03 2015.

Lundberg, S. M. and Lee, S.-I. A unified approach to interpreting model predictions. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 30*, pp. 4765–4774. Curran Associates, Inc., 2017.

Lundberg, S. M., Erion, G., Chen, H., DeGrave, A., Prutkin, J. M., Nair, B., Katz, R., Himmelfarb, J., Bansal, N., and Lee, S.-I. Explainable ai for trees: From local explanations to global understanding. *arXiv preprint arXiv:1905.04610*, 2019.

Romero, A., Ballas, N., Kahou, S. E., Chassang, A., Gatta, C., and Bengio, Y. Fitnets: Hints for thin deep nets. *arXiv*, 12 2014.

Zagoruyko, S. and Komodakis, N. Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer. *International Conference on Learning Representations*, 12 2016.