**Model Diagnostics: Overfitting & Robustness**

[[PiML Tutorials](https://piml.medium.com/?source=post_page-----46f26367c8ad--------------------------------)](https://piml.medium.com/?source=post_page-----46f26367c8ad--------------------------------)

[PiML Tutorials](https://piml.medium.com/?source=post_page-----46f26367c8ad--------------------------------)

·

[Follow](https://medium.com/m/signin?actionUrl=https%3A%2F%2Fmedium.com%2F_%2Fsubscribe%2Fuser%2F75c69de4b20b&operation=register&redirect=https%3A%2F%2Fpiml.medium.com%2Fmodel-diagnostics-overfitting-and-robustness-46f26367c8ad&user=PiML+Tutorials&userId=75c69de4b20b&source=post_page-75c69de4b20b----46f26367c8ad---------------------post_header-----------)

8 min read

·

Oct 21

25

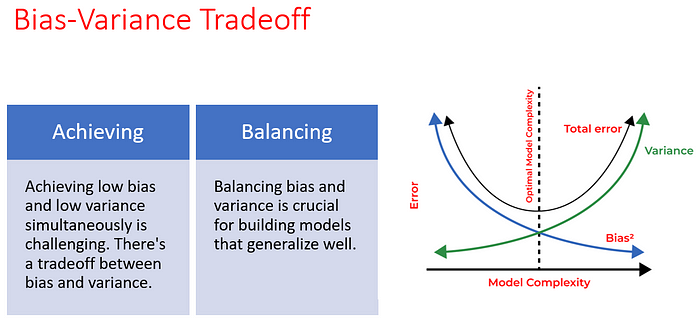
*Today’s PiML tutorial is about model overfitting and robustness assessment, contributed by Nengfeng (a seasoned statistician who takes a principal role in validating data science and artificial intelligence models in the bank).*

**Model Overfitting**

There are two types of model errors:

* **Bias:** Error due to overly simplistic assumptions in the learning algorithm. High bias can cause the model to miss relevant relations between features and target outputs.
* **Variance:** Error due to too much complexity in the learning algorithm. High variance can cause the model to model the random noise in the training data.

It’s important to achieve a tradeoff between bias and variance when building a ML model. We will focus on variance in this discussion. Model with high variance has the problem of overfitting.



Topics of Model Robustness and Regularization, UNCC Master of Data Science, Fall 2023

The followings are the main causes of overfitting.

* **Complex Models :**Models with high capacity can fit the training data too closely, capturing noise and spurious correlations.
* **Insufficient Data :**When there’s not enough data relative to the complexity of the model, it’s easy to fit noise.
* **Noisy Data:**When there’s not enough data relative to the complexity of the model, it’s easy to fit noise.

The consequence of overfitting models is:

* **Poor Generalization:** Overfit models fail to generalize patterns beyond the training set.
* **Reduced Predictive Power**: Performance on unseen data deteriorates significantly.
* **Limited Real-world Applicability**: Overfit models often fail when faced with new, real-world data.

Overfitting can be measured by the following metrics:

* **ROC-AUC Score (for Binary Classification):**Receiver Operating Characteristic — Area Under the Curve measures the area under the ROC curve. AUC close to 1 indicates a good model, but if it’s significantly higher on the training data than the test data, overfitting might be present.
* **R-squared (for Regression):**In regression problems, R-squared measures how well the model’s predictions match the actual data. If R-squared is very high on training data but much lower on test data, overfitting may be occurring.

ML model can be easily become very complicated. It’s very critical to avoid overfitting in a ML model. The following techniques can be used to mitigate overfitting.

* **Model Simplification:**Use simpler models with fewer parameters to avoid fitting noise; identify and utilize only the most relevant features, discarding less informative ones.
* **Regularization:**Add penalty terms to the loss function to discourage large weights, preventing the model from becoming too complex (L1 and L2 Regularization); randomly deactivate neurons during training to prevent over-reliance on specific neurons (Dropout).
* **Early Stopping :**Monitor validation performance and stop training once performance on a separate validation set plateaus or starts to worsen.
* **Data Augmentation:**Collecting more diverse and representative data to reduce the likelihood of overfitting.
* **Cross-Validation :**Use techniques like k-fold cross-validation to assess how well the model will generalize to new data.

**Model Robustness**

**Model robustness** refers to the ability of a machine learning model to maintain its performance and make accurate predictions even when it encounters data points or scenarios that differ from those in the training data. In other words, a robust model is able to handle variations, noise, outliers, or changes in the input without a significant drop in performance.

Testing model robustness involves evaluating how well a machine learning model performs under various conditions, including scenarios it may not have encountered during training. Here are some methods to test model robustness:

1. **Out-of-Distribution Testing:**Evaluate the model on data from a different distribution than the one it was trained on. This helps assess how well the model generalizes to unseen variations.
2. **Adversarial Testing:**Generate or obtain adversarial examples, which are intentionally crafted inputs designed to mislead the model. Test the model’s performance on these adversarial samples.
3. **Noisy Data Testing:**Add random noise to the test data and evaluate how the model handles noisy inputs. This tests the model’s robustness to variations in the input.
4. **Concept Drift Testing:**Simulate or obtain data that reflects changes in the underlying distribution over time. Test the model’s performance on this data to evaluate its adaptability to dynamic environments.

PiML provided a tool box to measure model robustness. It’s a performance based robustness metrics, that is to assess model performance to small changes in the covariate space. It proceeds as follows,

* Perturb x with changes △x.
* Calculate the model’s output f ̂(*x*+ △x ) on perturbed data.
* Evaluate the performance metric of Score(y, f ̂(*x*+ △x ) ).

This process is iterated ten times for all the test samples, with performance metrics recorded for each repetition.

It is important to note that the assumption is made that the response remains unchanged throughout. Perturbation can be applied to both numerical and categorical features. The forthcoming sections will illustrate the perturbation methods for numerical and categorical features, respectively.

There are different ways to perturb the data and the methods of perturbation can have significant impact to the robustness test results. We will talk about two methods of perturbation here.

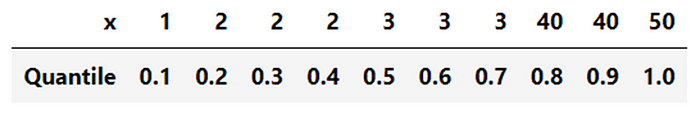
**Raw perturbation**: Directly add i.i.d. Gaussian noise *N(0,*λ *var(x))*to *x*, where λ is the perturbation size. However, this method may not be suitable when

* The data is discrete, e.g., 1, 2, 3, … 10. In this case, the perturbed data, e.g., 1.2 may become invalid.
* The data is skewed and has a long tail distribution. In this case, the calculation of standard deviation may become unstable, and it is relatively hard to choose a suitable perturbation size.
* The data has special values, such as 999999999.

**Quantile Perturbation:**Quantile perturbation can solve the above issues. It works as follows.

1. First, the feature is converted to the quantile space.
2. The uniform noise is then added to perturb the quantiles. Here also represents the perturbation size.
3. Finally, transform the quantiles to the original space.

For example, consider a simple discrete sample with 10 data points:



Consider the observed value 3, along with its corresponding quantile value 0.7. On the quantile scale, a small noise of 0.12 is generated and added to the original quantile value of 0.7. This sum yields a resulting value of 0.82, which is then rounded to the nearest available value, namely 0.8. Finally, the perturbed quantile is transformed back to the original scale, resulting in a value of 40.

**Case Study**

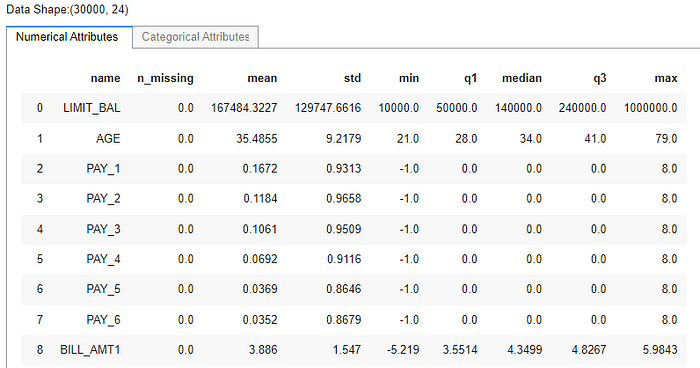
Suppose we are given some example models, for the Taiwan Credit data. The Python notebooks can be found from PiML Github or experimented directly through Google Colab:

<https://colab.research.google.com/drive/1jVVDesIAX4BYZo6gAzhuKmn7hEpDDWtb?usp=sharing>

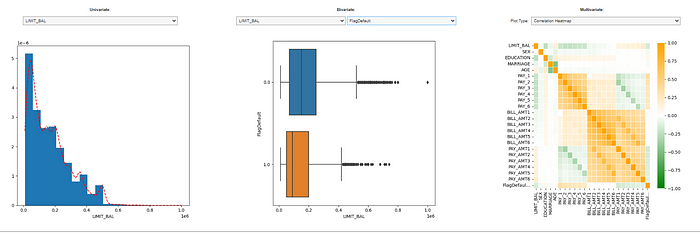
**🤖Data and Model Pipelines**

PiML provides the convenient low-code interfaces for data, model and test pipelines. After installing the package by “***pip install piml***”, one may initiate an experiment and load data:

from piml import Experiment  
exp = Experiment()  
exp.data\_loader(data=’CaliforniaHousing\_trim2')  
exp.data\_summary()



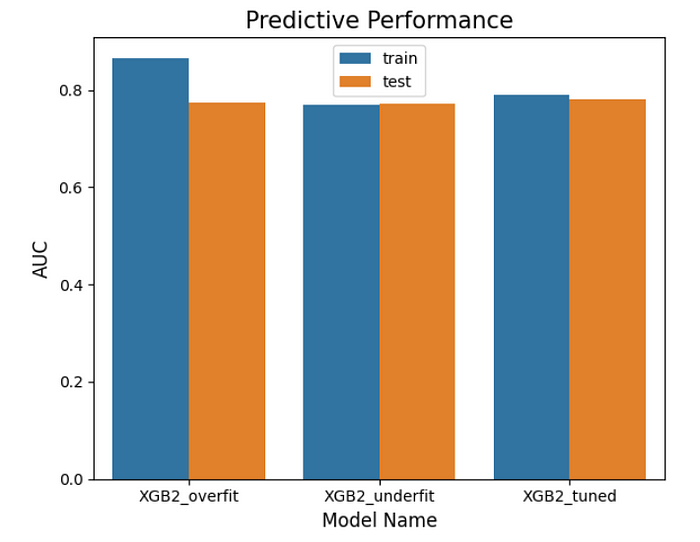
Data summary, preprocessing and visualization can be easily carried out in PiML. The last variable *FlagDefault*is the response; as usual, we split the data into 80–20 for training and testing. Please refer to the example notebook.



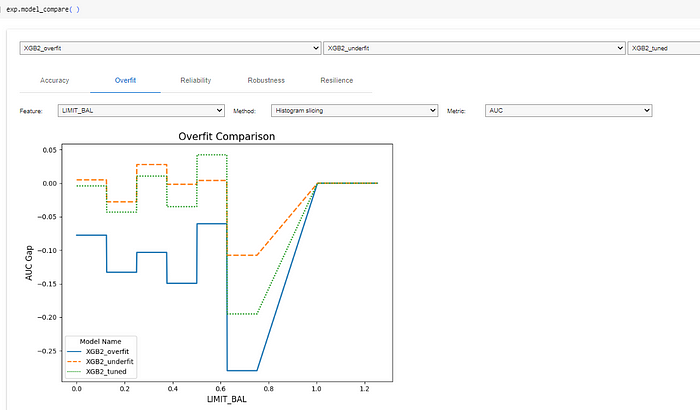
In model pipeline, PiML provides a whole list of interpretable machine learning models that come with model-specific interpretation, as well as post-hoc explanation. In this tutorial, let’s take XGBoost (depth 2) with different configurations to get overfit, underfit and tuned models.

from piml.models import XGB2Classifier   
clf1 = XGB2Classifier(n\_estimators=1000, max\_depth=2, eta = 0.3, max\_bin = 256, feature\_types = "numerical", random\_state=0)  
exp.model\_train(model=clf1, name='XGB2\_overfit')   
exp.model\_diagnose(model="XGB2\_overfit", show='accuracy\_table')  
  
clf2 = XGB2Classifier(n\_estimators=80, max\_depth=2, eta = 0.03, reg\_alpha = 0.3, reg\_lambda = 0.3, max\_bin = 256, feature\_types = "numerical", random\_state=0)  
exp.model\_train(model=clf2, name='XGB2\_underfit')  
exp.model\_diagnose(model="XGB2\_underfit", show='accuracy\_table')  
  
clf3 = XGB2Classifier(n\_estimators=200, max\_depth=2, eta = 0.05, reg\_alpha = 0.2, reg\_lambda = 0.2, max\_bin = 256, feature\_types = "numerical", random\_state=0)  
exp.model\_train(model=clf3, name='XGB2\_tuned')  
exp.model\_diagnose(model="XGB2\_tuned", show='accuracy\_table')

In the testing data, we observe that the AUC values for all three models are comparable. The tuned model displays a slightly higher AUC compared to the other two. However, it’s important to note that the overfit model shows a significantly higher AUC in the training data than the other two models. We aim to demonstrate with robust test data that the overfit model poses problems, despite its similar performance on the test data.



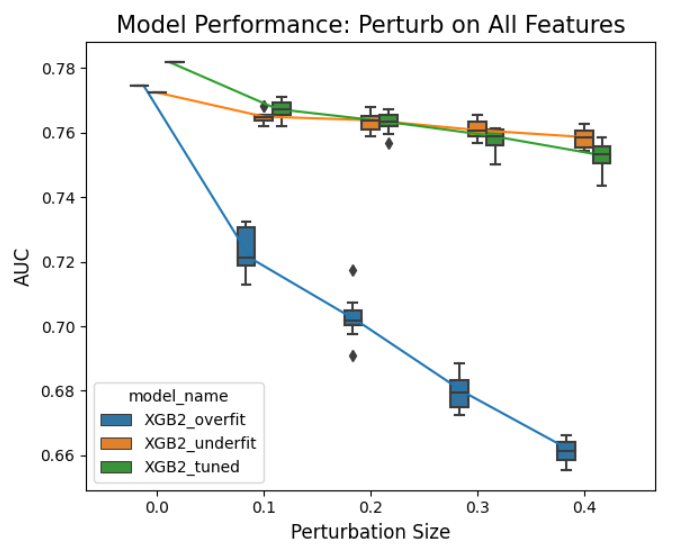
PiML also offers tools for comparing the AUC gap (test AUC — train AUC) across various regions. As illustrated below, the overfit model exhibits a larger gap than the other two models across all regions, with some regions showing a particularly substantial gap. The performance of the models in these regions, such as the range of 0.4 to 0.5 in scaled data, may be severely affected by overfitting and could pose potential issues.



Model overfit diagnostics by feature LIMT\_BAL

The plot below illustrates the model’s robustness test using the raw perturbation method. It’s evident that the performance of the overfit model significantly declines as the perturbation size increases. Specifically, its AUC drops from 0.77 to 0.72 when the perturbation size reaches 0.1. On the other hand, the tuned model maintains superior performance compared to the underfit model until the perturbation size reaches 0.2. Notably, a perturbation size of 0.2 is already quite substantial. Therefore, we have no immediate concerns for the tuned model, even if its performance slightly lags behind that of the underfit model when the perturbation size exceeds 0.2.

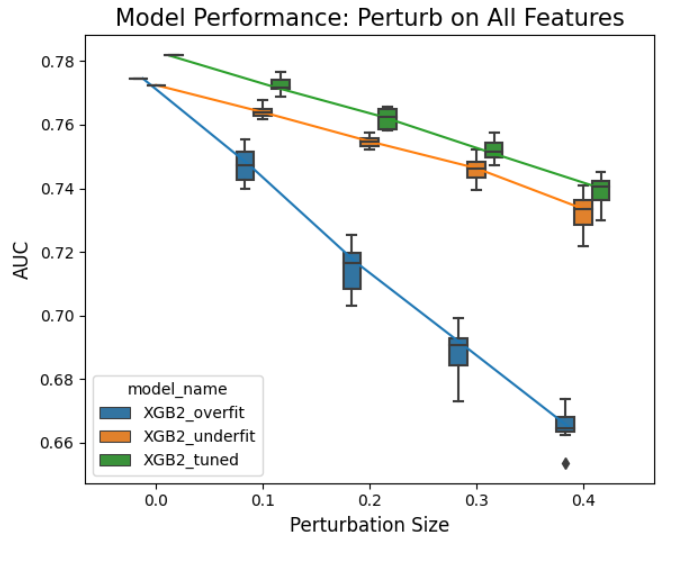
exp.model\_compare(models=[‘XGB2\_overfit’, ‘XGB2\_underfit’, ‘XGB2\_tuned’],   
 show=’robustness\_perf’, perturb\_size=0.1,   
 figsize=(6, 5))



Robustness test for the raw perturbation

The plot presented below depicts the model’s robustness test, this time utilizing the quantile perturbation method. The assessment for the overfit model aligns with that of the raw perturbation method. However, it’s noteworthy that the tuned model consistently outperforms the underfit model, reinforcing the notion of the tuned model’s robustness.

exp.model\_compare(models=[‘XGB2\_overfit’, ‘XGB2\_underfit’, ‘XGB2\_tuned’],   
 show=’robustness\_perf’, perturb\_method= “quantile”,   
 perturb\_size=0.1, figsize=(6, 5))



Robustness test for the quantile perturbation

Evaluating model robustness across various perturbations holds significant importance. In cases where different perturbations yield conflicting conclusions about the model’s robustness, further diagnostic analysis is warranted. Such analysis may include robustness tests based on perturbations focused on individual features (available in PiML). Additionally, scrutinizing model explainability, such as variable importance and the input-output relationship, can provide insights into the underlying causes.

**Thank you for reading!**

If you are, like us, passionate about data science, interpretable machine learning, and/or model risk management, please feel free to contact us on [**LinkedIn**](https://www.linkedin.com/in/ajzhang/).

*All images without a source credit were created by the author or generated by PiML.*

**About PiML***PiML was born as an educational toolbox for interpretable machine learning in Python community. It is designed for both model development and model diagnostics. Since its first launch on May 4th, 2022, PiML has continuously updated with new features and capabilities, together with a complete*[***user guide***](https://selfexplainml.github.io/PiML-Toolbox)*.*