

---

# SlickML: Slick Machine Learning in Python

---

Amirhessam Tahmassebi\*

Department of Scientific Computing  
Florida State University  
Tallahassee, FL, USA  
atahmassebi@fsu.edu

## Abstract

*SlickML* is an open-source machine learning (ML) library written in Python aiming at accelerating the experimentation time for an ML application by integrating popular state-of-the-art ML algorithms. Data scientists' tasks such as feature selection, model tuning, or evaluating metrics for classification, regression, clustering, and time-series problems can often be repetitive. *SlickML* provides data scientist with a toolbox of utility functions to quickly prototype solutions for a given problem with minimal code. Additionally, an exhaustive set of diagnostic visualization tools are available to quickly address the common issues with an ML application. The main goal of *SlickML* is to provide the users with reliable solutions that could be used for production environments regarding the most common problems in the industry while tackling controversial problems such as imbalanced learning, feature selection, threshold finding, and etc by adding new functionalities based on the novel publications in the research community. The source code, and documentation can be downloaded from <https://github.com/slickml/slick-ml> and the library can be easily installed via *PyPI* at <https://pypi.org/project/slickml/>.

## 1 Introduction

Emerging as one of the most recent machine learning state-of-the-art frameworks, automated machine learning (AutoML) libraries have been a point of interest for reducing the demand for data scientists and enabling domain experts to automatically build ML applications without much requirement for statistical and ML knowledge [1]. To name a few, we can mention packages such as *PyCaret* [2] and *TPOT* [3] which offer numerous automated pipelines for different ML applications. However, you cannot always rely on AutoML pipelines to solve real-world applications. Sometimes it is required to dig into a specific part of the pipeline to ensure that the solution is ready for deployment in production. Therefore, there is a need of some libraries to expedite this process with minimal code. Moreover, most of the data scientists' tasks are repetitive. As an example we can use one of the well-known applications of ML in healthcare which is the risk of readmission for patients within  $N$  days after getting discharged. In practice, to build a ML model (mostly regularized linear models) to predict the risk of readmission, we normally create as many historical lagged features as possible, hoping that during the feature selection process, the salient features are being selected to infer the risk of readmission for the new patient in future. Therefore, the process of feature engineering, feature selection, model training, model calibration, and etc are repetitive that can be automated. Additionally, having multiple test and diagnostic visualization tools would indicate how reliable the pipeline will be in production. In principle, there is a gap for the current need of industrial ML applications; this gap would be filled with a fusion of AutoML pipelines along with reliability tests. *SlickML* aims at fill in the gap by offering reliable solutions and accelerating the experimentation time for an ML application.

---

\*Corresponding Author

## 2 Project Vision

The project's goal is to provide reliable implementations of the favorite ML algorithms for the most essential tasks in any ML projects in industry. Code quality is ensured by implementing unit tests and using static code linters such as *flake8* and *black* in addition to providing exhaustive docstring for each class along with using consistent naming conventions for classes, functions, and parameters based on Python coding guidelines and *scikit-learn* library. The project is being developed using collaborative tools such as git, github, and public mailing lists following our contributing instructions (<https://github.com/slickml/slick-ml/blob/master/CONTRIBUTING.md>) under MIT license (<https://github.com/slickml/slick-ml/blob/master/LICENSE>) and our code of conduct ([https://github.com/slickml/slick-ml/blob/master/CODE\\_OF\\_CONDUCT.md](https://github.com/slickml/slick-ml/blob/master/CODE_OF_CONDUCT.md)). *SlickML* provides a quick-start documentation, class references, installation instructions, as well as detailed examples developed in *jupyter notebooks* (<https://github.com/slickml/slick-ml/tree/master/examples>) of all of the underlying classes. The source code, and documentation can be downloaded from <https://github.com/slickml/slick-ml> and the library can be easily installed via *PyPI* at <https://pypi.org/project/slickml/>.

## 3 Underlying Technologies

Similar to *scikit-learn* package, the base data structures used for data model are *Pandas* dataframes and *Numpy* arrays. Additionally, the compressed sparse row (CSR) matrix representation is developed for all of the models to boost the run-time speed and optimize memory. *xgboost* and *glmnet* are used for the main classification and regression models, in which they are compatible with *shap*, which is used for explainability. Bayesian optimization and Tree of Parzen Estimators are the main algorithms for hyper-parameters tuning functionality. Lastly, the visualizations are based on *matplotlib* and *seaborn*.

## 4 Current & Future Developments

In this section, the main developed/planned functionalities of the library are presented. More details on <https://github.com/slickml/slick-ml/issues/62>.

### 4.1 Current Developments

1. Classification: The main classifiers are *xgboost* and *glmnet*. Both of them have internal cross-validation functionality to find the best number of boosting round and penalty parameters.
2. Regression: Similar to classification, the main regressors are including *xgboost* and *glmnet*. Both of the developed classifiers have internal cross-validation functionality to find the best number of boosting round and penalty parameters.
3. Optimization: Two main algorithms for hyper-parameters tuning are Bayesian optimization [4] and Tree of Parzen Estimators (TPE) [5].
4. Feature Selection: The main feature selection algorithm is based on the highest frequency of the features that are survived through target permutation and added structured noisy features at each iteration of n-folds cross-validations. The algorithm is developed for both *xgboost* and *glmnet* as base estimators.
5. Metrics: A set of complete metrics for binary classification developed along with diagnostic plots to find the best threshold based on three methodologies, including Youden's J index [6], maximizing sensitivity-specificity, and maximizing precision-recall. Similarly, a set of complete metrics is available for regression problems. The metrics functionality is invariant of the models' objects and they use the predicted and ground truth arrays as the inputs that can be used as a post-processing step in any ML project.
6. Plotting: The plotting functionality is embedded within every single class through the library. In better words, for every single instance that is initiated, the user can directly call the plotting module and there is no need to load any other classes or functions. In this case, the user can enjoy the diagnostic visualizations for all of the steps in a ML model including training/validation/testing without worrying about caching the data.

## 4.2 Future Developments

1. Classification/Regression: In addition to the current developed models, it is planned to add a probabilistic based prediction model such as *ngboost* [7].
2. Optimization: *optuna* [8] has recently shown great success in multiple industrial use-cases as it is planned to be added to *SlickML* library.
3. Plotting: It is planned to follow the same embedding architecture for plotting modules similar to the developed modules as we progress on the planned functionalities.
4. Calibration: It is planned to add prediction calibration instead of model calibration to the library. Available packages such as *scikit-learn* limit their calibration to the developed models based on their API while there will be numerous models that are not compatible with said API. Therefore, similar to the developed metrics functionality, a prediction-based calibration feature is planned to be added which would be invariant of the model's API. In addition to the common parametric methods including Platt's scaling (logistic regression) or non-parametric methods like isotonic regression, it is planned to add two more implementations: histogram binning and Platt's scaling binning. Histogram binning is a stricter version of isotonic regression while Platt's scaling binning is a blend of Platt's scaling and histogram binning. The big motivation is to introduce a lower calibration error.
5. Imbalanced Learning: Imbalanced learning has presented itself to be an essential part in many critical real-world applications. Thus, this is one of the major planned features to be added. There are plenty of novel research in this field including minority oversampling technique (SMOTE), adaptive synthetic (ADASYN), EasyEnsemble, BalanceCascade, SMOTEBoost, RAMOBoost, DataBoost-IM, one-side selection (OSS), Tomek links, neighborhood cleaning rule (NCL), active learning, one-class learning and etc [9]. The *imbalanced-learn* [10] library has recently shown great success in incorporating some of the abovementioned algorithms. However, the API is being setup as an experimental step which demands more effort for the user to grasp the under-the-hood algorithm along with estimators. Our goal is to minimize this effort as much as possible and incorporate the abovementioned algorithms within the developed estimators; so, the user can easily see the performance of the model with or without the embedded imbalanced learning strategy.
6. Time Series: Time series problems can be one of the most challenging types of ML problems. It is planned to add an exhaustive set of solutions including auto-regressive models along with the most recent solutions such as *prophet* [11] and *tsfresh* [12] to the library.
7. Genetic Programming: It is planned to add an evolutionary classifier and an evolutionary regressor using multi-objective genetic programming (MOGP) algorithm. The MOGP algorithm has shown great success in solving real-world problems [13, 14].
8. Survival Analysis: Inference on the risk of customer attrition or the risk of cancer recurrence in a patient who underwent chemotherapy can be always challenging. Recent development of multiple survival packages in Python such as *lifelines* and *scikit-survival* or embedded solutions using accelerated failure time (AFT) algorithm and *xgboost* such as *xgbse* are the reasons that we have added survival analysis functionality on the roadmap.
9. Documentation: Currently, preparation of the the documentation is in progress and the website (<https://www.slickml.com>) is under construction.

## 5 Conclusions

The project's goal is to provide reliable implementations of the favorite ML algorithms for the most essential tasks in some of the ML projects in industry. Therefore, the users would be able to quickly prototype solutions for a given problem with minimum lines of code. Additionally, the users have access to an exhaustive set of diagnostic visualization tools to address the common issues with an ML application.

## Acknowledgments and Disclosure of Funding

The author would like to thank Trace Smith, Eitan Lees, Behshad Mohebali, and other contributors that created examples, wrote documentation, and reported bugs. You can find an up-to-date list of contributors on GitHub.

## References

- [1] X. He, K. Zhao, and X. Chu, “Automl: A survey of the state-of-the-art,” *Knowledge-Based Systems*, vol. 212, p. 106622, 2021.
- [2] M. Ali, *PyCaret: An open source, low-code machine learning library in Python*, April 2020, pyCaret version 2.3.1. [Online]. Available: <https://www.pycaret.org>
- [3] T. T. Le, W. Fu, and J. H. Moore, “Scaling tree-based automated machine learning to biomedical big data with a feature set selector,” *Bioinformatics*, vol. 36, no. 1, pp. 250–256, 2020.
- [4] J. Snoek, H. Larochelle, and R. P. Adams, “Practical bayesian optimization of machine learning algorithms,” *arXiv preprint arXiv:1206.2944*, 2012.
- [5] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, “Algorithms for hyper-parameter optimization,” in *25th annual conference on neural information processing systems (NIPS 2011)*, vol. 24. Neural Information Processing Systems Foundation, 2011.
- [6] W. J. Youden, “Index for rating diagnostic tests,” *Cancer*, vol. 3, no. 1, pp. 32–35, 1950.
- [7] T. Duan, A. Anand, D. Y. Ding, K. K. Thai, S. Basu, A. Ng, and A. Schuler, “Ngboost: Natural gradient boosting for probabilistic prediction,” in *International Conference on Machine Learning*. PMLR, 2020, pp. 2690–2700.
- [8] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, “Optuna: A next-generation hyperparameter optimization framework,” in *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, 2019, pp. 2623–2631.
- [9] H. He and Y. Ma, “Imbalanced learning: foundations, algorithms, and applications,” 2013.
- [10] G. Lemaître, F. Nogueira, and C. K. Aridas, “Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning,” *Journal of Machine Learning Research*, vol. 18, no. 17, pp. 1–5, 2017. [Online]. Available: <http://jmlr.org/papers/v18/16-365>
- [11] S. J. Taylor and B. Letham, “Forecasting at scale,” *The American Statistician*, vol. 72, no. 1, pp. 37–45, 2018.
- [12] M. Christ, N. Braun, J. Neuffer, and A. W. Kempa-Liehr, “Time series feature extraction on basis of scalable hypothesis tests (tsfresh—a python package),” *Neurocomputing*, vol. 307, pp. 72–77, 2018.
- [13] K. Veeramachaneni, I. Arnaldo, O. Derby, and U.-M. O’Reilly, “Flexgp,” *Journal of Grid Computing*, vol. 13, no. 3, pp. 391–407, 2015.
- [14] Y. Yang, “Adaptive regression by mixing,” *Journal of the American Statistical Association*, vol. 96, no. 454, pp. 574–588, 2001.

## Checklist

1. For all authors...
  - (a) Do the main claims made in the abstract and introduction accurately reflect the paper’s contributions and scope? [Yes]
  - (b) Did you describe the limitations of your work? [Yes] For more details please take a look at the section 4.
  - (c) Did you discuss any potential negative societal impacts of your work? [N/A]
  - (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...

- (a) Did you state the full set of assumptions of all theoretical results? [N/A]
- (b) Did you include complete proofs of all theoretical results? [N/A]
- 3. If you ran experiments...
  - (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] For more details please take a look at <https://github.com/slickml/slick-ml>.
  - (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] For more details please take a look at <https://github.com/slickml/slick-ml>.
  - (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] For more details please take a look at <https://github.com/slickml/slick-ml>.
  - (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [N/A]
- 4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
  - (a) If your work uses existing assets, did you cite the creators? [Yes]
  - (b) Did you mention the license of the assets? [Yes] For more details please take a look at section 2.
  - (c) Did you include any new assets either in the supplemental material or as a URL? [Yes]
  - (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
  - (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
- 5. If you used crowdsourcing or conducted research with human subjects...
  - (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
  - (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
  - (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

## A Appendix

In this section, some of the SlickML code snippets along with the resulted visualizations are presented as supplementary materials. More details on <https://github.com/slickml/slick-ml/tree/master/examples>.

```

1 # run feature selection using loaded data
2 from slickml.feautre_selection import XGBoostFeatureSelector
3 xfs = XGBoostFeatureSelector()
4 xfs.fit(X, y)
5
6 # plot cross-validation results
7 xfs.plot_cv_results()
8
9 # plot feature frequency after feature selection
10 xfs.plot_frequency()
```

Listing 1: Feature Selection

```

1 # apply BayesianOpt to tune parameters of classifier using loaded train/
  test data
2 from slickml.optimization import XGBoostClassifierBayesianOpt
3 xbo = XGBoostClassifierBayesianOpt()
4 xbo.fit(X_train, y_train)
5
6 # best parameters
```

```

7 best_params = xbo.get_best_params()
8 best_params
9
10 {'colsample_bytree': 0.8213916662259918,
11  'gamma': 1.0,
12  'learning_rate': 0.23148232373451072,
13  'max_depth': 4,
14  'min_child_weight': 5.632602921054691,
15  'reg_alpha': 1.0,
16  'reg_lambda': 0.39468801734425263,
17  'subsample': 1.0}

```

Listing 2: Hyper-Parameter Tuning

```

1 # train a classifier using loaded train/test data and best params
2 from slickml.classification import XGBoostCVClassifier
3 clf = XGBoostCVClassifier(params=best_params)
4 clf.fit(X_train, y_train)
5 y_pred_proba = clf.predict_proba(X_test)
6
7 # plot cross-validation results
8 clf.plot_cv_results()
9
10 # plot features importance
11 clf.plot_feature_importance()
12
13 # plot SHAP summary violin plot
14 clf.plot_shap_summary(plot_type="violin")
15
16 # plot SHAP summary layered violin plot
17 clf.plot_shap_summary(plot_type="layered_violin",
18                       layered_violin_max_num_bins=5)
19
20 # plot SHAP waterfall plot
21 clf.plot_shap_waterfall()

```

Listing 3: Model Training with XGBoost

```

1 # train a classifier using loaded train/test data and your choice of
  params
2 from slickml.classification import GLMNetCVClassifier
3 clf = GLMNetCVClassifier(alpha=0.3, n_splits=4, metric="roc_auc")
4 clf.fit(X_train, y_train)
5 y_pred_proba = clf.predict_proba(X_test)
6
7 # plot cross-validation results
8 clf.plot_cv_results()
9
10 # plot coefficients paths
11 clf.plot_coeff_path()

```

Listing 4: Model Training with GLMNet

```

1 # plot binary metrics
2 from slickml.metrics import BinaryClassificationMetrics
3 clf_metrics = BinaryClassificationMetrics(y_test, y_pred_proba)
4 clf_metrics.plot()

```

Listing 5: Binary Classification Metrics

```

***** Iteration 1 *****
***** Fold = 1/4 -- Train AUC = 0.929 -- Test AUC = 0.836 *****
***** Fold = 2/4 -- Train AUC = 0.934 -- Test AUC = 0.826 *****
***** Fold = 3/4 -- Train AUC = 0.923 -- Test AUC = 0.906 *****
***** Fold = 4/4 -- Train AUC = 0.937 -- Test AUC = 0.833 *****
***** Internal 4-Folds CV: -- Train AUC = 0.941 +/- 0.004 -- Test AUC = 0.852 +/- 0.015 *****
***** External 4-Folds CV: -- Train AUC = 0.931 +/- 0.005 -- Test AUC = 0.850 +/- 0.032 *****

***** Iteration 2 *****
***** Fold = 1/4 -- Train AUC = 0.918 -- Test AUC = 0.860 *****
***** Fold = 2/4 -- Train AUC = 0.935 -- Test AUC = 0.874 *****
***** Fold = 3/4 -- Train AUC = 0.947 -- Test AUC = 0.788 *****
***** Fold = 4/4 -- Train AUC = 0.925 -- Test AUC = 0.874 *****
***** Internal 4-Folds CV: -- Train AUC = 0.941 +/- 0.010 -- Test AUC = 0.842 +/- 0.020 *****
***** External 4-Folds CV: -- Train AUC = 0.931 +/- 0.011 -- Test AUC = 0.849 +/- 0.036 *****

***** Iteration 3 *****
***** Fold = 1/4 -- Train AUC = 0.933 -- Test AUC = 0.830 *****
***** Fold = 2/4 -- Train AUC = 0.891 -- Test AUC = 0.785 *****
***** Fold = 3/4 -- Train AUC = 0.924 -- Test AUC = 0.865 *****
***** Fold = 4/4 -- Train AUC = 0.854 -- Test AUC = 0.818 *****
***** Internal 4-Folds CV: -- Train AUC = 0.910 +/- 0.029 -- Test AUC = 0.843 +/- 0.016 *****
***** External 4-Folds CV: -- Train AUC = 0.900 +/- 0.031 -- Test AUC = 0.824 +/- 0.029 *****

```

Figure 1: The performance of the feature selection algorithm at each iteration with internal/external 4-folds cross-validation.

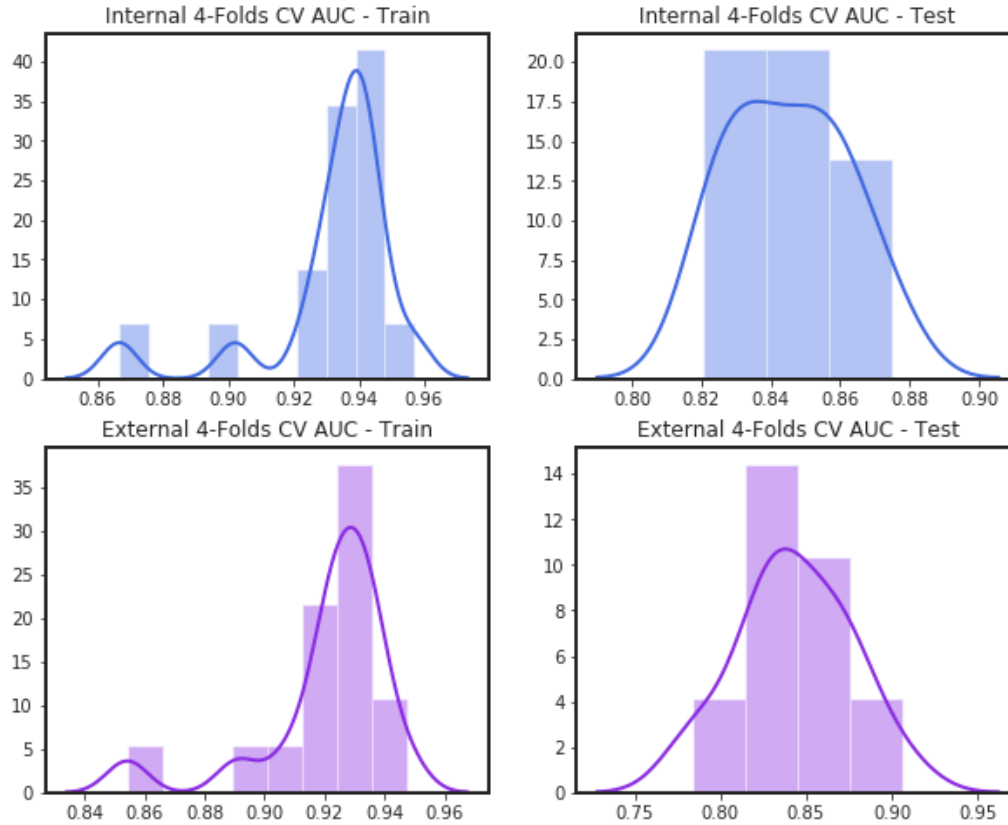


Figure 2: The internal and external cross-validation results (here AUC) for the feature selection process.

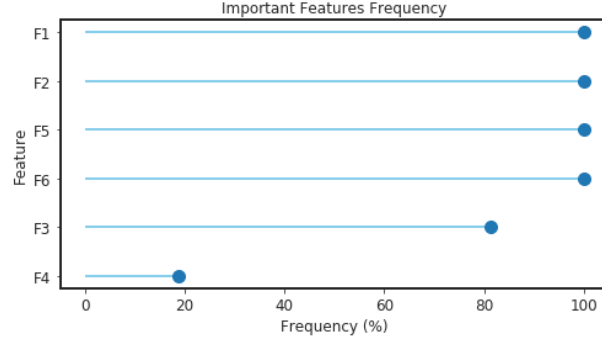


Figure 3: Feature frequency after the iterative feature selection process.

iter	target	colsam...	gamma	learn...	max_depth	min_ch...	reg_alpha	reg_la...	subsample
1	0.8245	0.8975	0.04571	0.6628	4.238	1.436	0.3064	0.7136	0.1931
2	0.8374	0.7904	0.6447	0.9152	3.334	3.238	0.7772	0.269	0.9726
3	0.8199	0.8498	0.6044	0.6874	6.651	15.7	0.061	0.5114	0.6848
4	0.8551	0.7297	0.8513	0.4627	4.757	4.965	0.9328	0.363	0.9365
5	0.8173	0.5425	0.5451	0.8782	6.633	5.028	0.1845	0.333	0.9125
6	0.8099	0.4336	0.282	0.3017	4.288	16.0	0.06196	0.8237	0.8923
7	0.8386	0.5277	0.7101	0.3167	2.901	9.436	0.6913	0.1258	0.6468
8	0.8075	0.1291	0.225	0.8967	6.216	3.886	0.5093	0.2564	0.3911
9	0.8546	0.7847	0.8845	0.4636	4.454	4.869	0.969	0.3627	0.98
10	0.8644	0.8214	1.0	0.2315	4.571	5.633	1.0	0.3947	1.0

Figure 4: The performance of hyper-parameter tuning using Bayesian optimization.

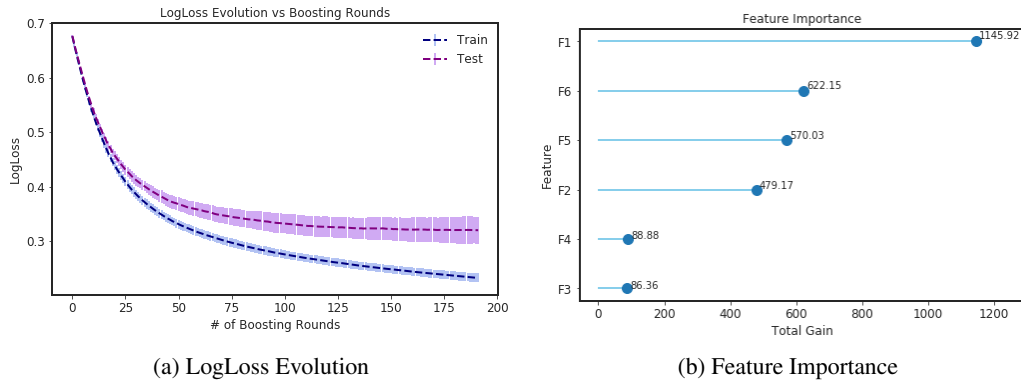


Figure 5: (a) The LogLoss evolution vs number of boosting rounds for both train/test sets. (b) the feature importance based on the final trained model.



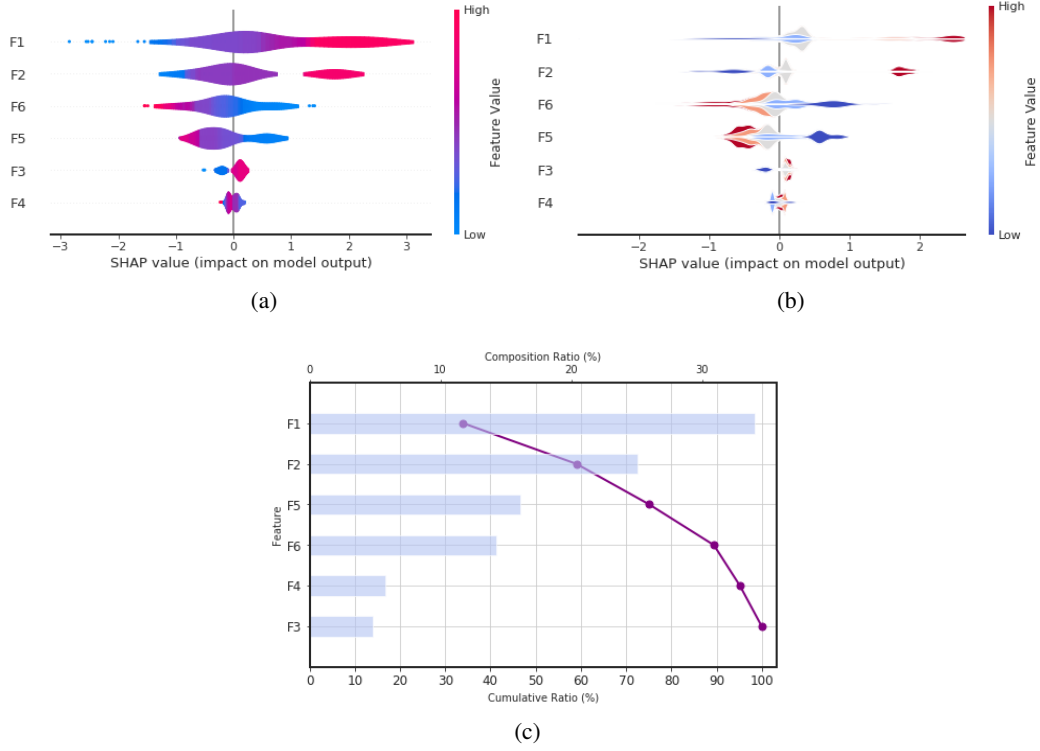


Figure 6: Shap summary plots.

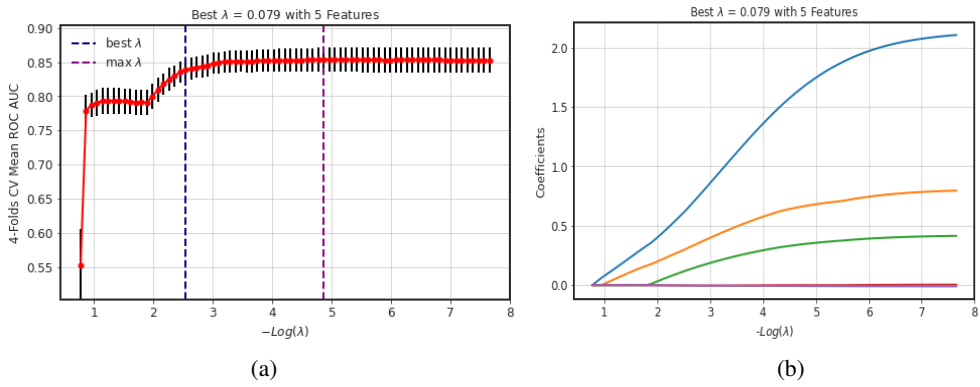


Figure 7: (a) The performance results (here mean AUC) of 4-folds cross-validation to tune the best penalty parameter. (b) The linear coefficients paths based on the best penalty parameter.

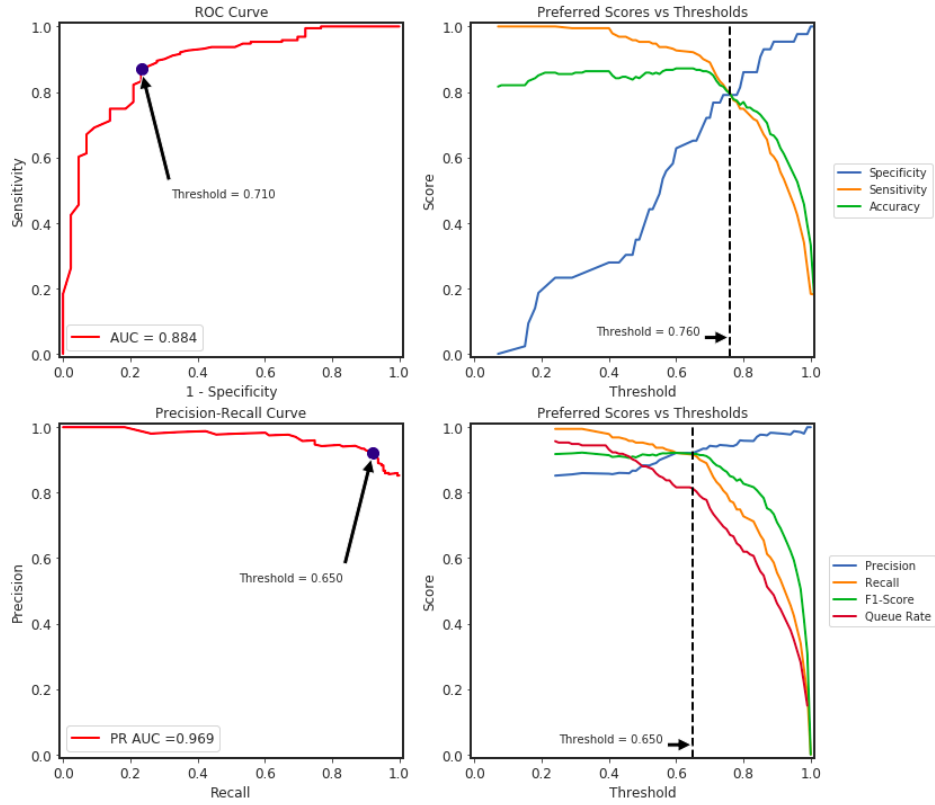


Figure 8: The diagnostic plots (ROC and PR curves) to find the best threshold based on trained XGBoost. The top left refers to the Youden's J index, the top right refers to the threshold that maximizes the sensitivity-specificity, and bottom right refers to the threshold that maximizes the precision-recall.

	Accuracy	Balanced Accuracy	ROC AUC	PR AUC	Precision	Recall	Average Precision	F-1 Score	F-2 Score	F-0.50 Score	Threat Score	TP	TN	FP	FN
Threshold = 0.850   Average = Binary	0.744000	0.780000	0.827000	0.945000	0.952000	0.723000	0.941000	0.821000	0.759000	0.895000	0.697000	138	36	7	53
Threshold = 0.830   Average = Binary	0.761000	0.763000	0.827000	0.945000	0.935000	0.759000	0.941000	0.838000	0.789000	0.894000	0.721000	145	33	10	46
Threshold = 0.680   Average = Binary	0.833000	0.718000	0.827000	0.945000	0.896000	0.901000	0.941000	0.898000	0.900000	0.897000	0.815000	172	23	20	19
Threshold = 0.850   Average = Weighted	0.744000	0.780000	0.827000	0.945000	0.851000	0.744000	0.941000	0.771000	0.746000	0.813000	0.630000	138	36	7	53
Threshold = 0.830   Average = Weighted	0.761000	0.763000	0.827000	0.945000	0.840000	0.761000	0.941000	0.784000	0.765000	0.814000	0.656000	145	33	10	46
Threshold = 0.680   Average = Weighted	0.833000	0.718000	0.827000	0.945000	0.832000	0.833000	0.941000	0.833000	0.833000	0.832000	0.763000	172	23	20	19
Threshold = 0.850   Average = Macro	0.744000	0.780000	0.827000	0.945000	0.678000	0.780000	0.941000	0.683000	0.724000	0.673000	0.536000	138	36	7	53
Threshold = 0.830   Average = Macro	0.761000	0.763000	0.827000	0.945000	0.677000	0.763000	0.941000	0.690000	0.723000	0.677000	0.546000	145	33	10	46
Threshold = 0.680   Average = Macro	0.833000	0.718000	0.827000	0.945000	0.722000	0.718000	0.941000	0.720000	0.718000	0.721000	0.593000	172	23	20	19
Threshold = 0.850   Average = Micro	0.744000	0.780000	0.827000	0.945000	0.744000	0.744000	0.941000	0.744000	0.744000	0.744000	0.697000	138	36	7	53
Threshold = 0.830   Average = Micro	0.761000	0.763000	0.827000	0.945000	0.761000	0.761000	0.941000	0.761000	0.761000	0.761000	0.721000	145	33	10	46
Threshold = 0.680   Average = Micro	0.833000	0.718000	0.827000	0.945000	0.833000	0.833000	0.941000	0.833000	0.833000	0.833000	0.815000	172	23	20	19

Figure 9: The complete set of metrics based on the test set.