# Evaluating ML Models on Binary Classification Problems

**Ha Duong – A16510699**

GITHUB REPO – HDUONG@UCSD.EDU

**ABSTRACT**

In this paper we compare the performance of multiple machine learning models such as NeuralNet, Boosted Trees, Stochastic Gradient Descent (SGD) and RandomForest. We evaluate these models' performance on 4 datasets of varying characteristics.

## 1. INTRODUCTION

Machine Learning has been increasingly important in solving a wide range of real-world problems. Especially in recent years with the advent of LLMs like OpenAI's ChatGPT or Anthropic's Claude. From problems like predicting progression of medical conditions to predicting the quality of a wine based on its physicochemical lab results. In this study we evaluate and compare the performance of some machine learning algorithms empirically. We evaluate models such as Gradient Boosted Tree, RandomForest, NeuralNet, Stochastic Gradient Descent and then custom meta-model SVM that aggregates the results of the other models. The study aims to look at the strengths and weaknesses of each model by training them on 4 data sets with unique challenges, varying in size, dimension and imbalance.

## 2. METHODOLOGY

### 2.1. Learning Algorithms

We performed a grid search on a space of parameters. We then trained and tested the model with the best parameters on three different data partitioning schemes (80/20, 50/50, and 20/80). This section summarizes the machine learning models tested and the parameters search space for each of them.

**Gradient-Boosted Tree (GBDT)**: we're using the XGBoost implementation (Chen & Guestrin, 2016). We consider boosted trees with [100, 200] boosting rounds, with learning rates [0.01, 0.1] and tree complexity (depth) of [3, 7, 10]. We kept the L1 regularization constant at 0.5 and the L2 regularization at 5.

**Feed-Forward NN**: we used the Tensorflow neural network implementation (Abadi et al., 2015) with 2 hidden layers with 64 neurons and 32 neurons. The optimizer is the default "Adam" and we vary between learning rate values of [0.001, 0.01]. We also vary the batch size between [64, 128, 256]. The model ran for 200 epochs maximum, stopping early if there are 10 epochs with no improvement in loss.

**RandomForest (RF)**:(Breiman, 2001) we considered [50, 100, 200] trees with a max depth for each tree of [10, 20]. The minimum samples before splitting are [2, 5].

**Stochastic Gradient Descent (SGD)**: we're using the Scikit-learn implementation (Pedregosa et al., 2011). We use this with the 'modified-huber' loss function which effectively approximates a linearSVM. We used ElasticNet regularization, varying alpha between [$10^{-5}$, $10^{-3}$, $10^{-1}$] and L1-L2 mixing ratios of [0.15, 0.5, 0.85]. The stop tolerances are [$10^{-6}$, $10^{-5}$].

**Meta-Model SVM**: an SVM with the RBF kernel acts as a meta model and predicts the target variable using our other model's predictions. This model is trained on a meta feature which is an aggregate of GBDT, NN, RF, and SGD predictions. We vary regularization between [0.1, 1, 10] and gamma values between [0.01, 0.1, 1].

### 2.2. Sampling, Training, and Validation

The datasets are inherently unbalanced so over-sampling of the minority class was used on all of them using SMOTE. Synthetic Minority Oversampling Technique (SMOTE) generates synthetic data points using *k* neighbors in the minority class until the classes are balanced. The drawback of this technique is that not all data points the models were trained on were real data, in some cases synthetic data points made up roughly 60% of data in the minority class. However, during exploration we found that without oversampling, models always settled on the simple heuristic of predicting the majority class in order to maximize the evaluation metric which goes against the prediction/detection goals for these datasets.

All these models are first cross validated on a data split of 80/20 (train/test). A grid search is performed which goes through every combination of parameters in the search space. This is done alongside 5-folds cross validation.

We then evaluate the model with the best parameters from the grid search. We evaluate the model on 3 data splits (80/ 20, 50/50, and 20/80) and compare performance metrics.

### 2.3. Metrics

We evaluate and compare these models based on accuracy (ACC), area under the ROC curve (ROC-AUC), and average precision (APR). Accuracy gives us a general idea of how the model's performing. ROC-AUC measures the model's ability to distinguish between 2 classes, however, it's not sensitive to data imbalance (e.g. the model can simply only predict the majority class and still get a decent ROC-AUC score). APR measures the quality of the model's positive predictions, which is more sensitive to imbalanced datasets and better aligns with the prediction goals of some of these data sets. Including both ROC-AUC and APR can give us an idea of whether or not the model is overfitting. A very high ROC-AUC but not-that-good APR can indicate the model is "memorizing" the training data.

### 2.4. Data Sets

We trained and tested these models on 4 different binary classification problems with all data imported from the UCI ML repository.

WINE's target variable was a quality rating of wine from 0-10 (actual data only contained 3-9) which we transformed to binary form where wines with rating 1-5 were "worse" and assigned 0.0, and wines with rating 6+ were "better" and assigned 1.0. For SEPSIS and CDC all numerical features were normalized and some features were dropped for very low 'feature importance' found during exploration (using RandomForest). PHISH had some features with 3 levels which were encoded into binary features.

*Table 1.* Description of problems

| PROBLEM | #ATTR | #INSTANCE | %POZ |
|---|---|---|---|
| SEPSIS | 3 | 110341 | 7% |
| CDC | 18 | 235795 | 14% |
| PHISH | 29 | 11055 | 44% |
| WINE | 11 | 4898 | 63% |

## 3. RESULTS

On average over all problems, the custom ensemble with SVM as the meta model performed the best; followed by NN, RF, GDBT then SGD. On the SEPSIS and CDC problems where there's a big class imbalance, the models' performance were very similar. They all performed poorly, either solely predicting the majority class, or achieving very low precision on one class in exchange for high precision on another. If we only consider the two problems where classes were relatively balanced (WINE and PHISH), then the

*Table 2.* Model's performance metric on problems (averaged over 3 metrics ACC, ROC-AUC and APR). Sorted by aggregate mean over all models.

| MODEL | WINE | PHISH | SEPSIS | CDC | MEAN |
|---|---|---|---|---|---|
| SVM | 0.946 | 0.976 | 0.722 | 0.752 | 0.849 |
| NN | 0.814 | 0.962 | 0.867 | 0.628 | 0.818 |
| RF | 0.863 | 0.971 | 0.742 | 0.674 | 0.813 |
| GDBT | 0.818 | 0.971 | 0.741 | 0.689 | 0.805 |
| SGD | 0.790 | 0.954 | 0.748 | 0.649 | 0.785 |

SVM meta model performed the best, backed by good predictions from both of the tree models. Out of the base models, generally RandomForest had the best performance, especially when we're only considering the more balanced datasets. This is followed by Gradient-Descent Boosted Trees, and then Neural Net.

As expected, base models performed slightly better for data splits 80/20 than 20/80 (See Appendix). Counter-intuitively, the SVM meta model sometimes performed worse in larger training partitions. (See Figure 3). We suspect that with larger training partitions the model overfitted to base models' "bad predictions" and picked up some of that behavior, causing poorer performance.

*Figure 1.* ROC-AUC Heatmap for WINE

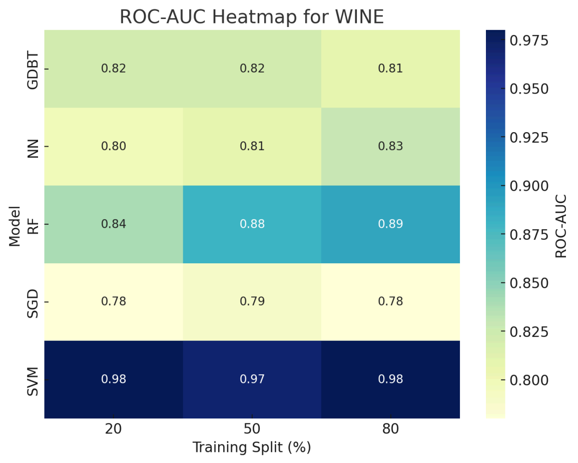
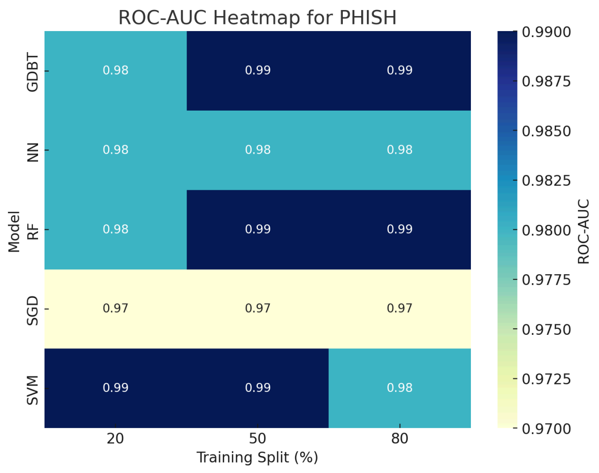


*Figure 2.* ROC-AUC Heatmap for PHISH



## 5. CHALLENGES AND LIMITATIONS

The main challenge that we encountered was the inherent class imbalance in the SEPSIS and CDC datasets. The problem was to identify patients at risk of dying to sepsis and patients who might have diabetes. By nature, the majority class was much larger than the minority and with sampling techniques we attempted to address this imbalance. Even though we used SMOTE sampling, models still performed poorly. This challenge was tied to the lack of computation power, especially on the large CDC dataset. We wanted to experiment with a deeper Neural Network or different architecture but were very limited by computation power and training time.

## 6. CONCLUSION

We found that under these study conditions, RandomForest generally performed the best across all 4 problems, achieving the highest ACC and ROC-AUC. A very close second is Boosted Trees (GDBT) with the XGBoost implementation. In third is Feed-Forward Neural Network (NN). Stochastic Gradient Descent (SGD) performed poorly.

We are happy with the performance of the custom ensemble with SVM as the meta model as it managed to take weaker predictors (the base models) as inputs and make better predictions as evident in the results.

## REFERENCES

Caruana, R., & Niculescu-Mizil, A. (2006). An empirical comparison of supervised learning algorithms. *Proceedings of the 23rd International Conference on Machine Learning (ICML)*, 161–168.

Breiman, L. (2001). Random forests. *Machine Learning, 45*(1), 5–32.

Chen, T., & Guestrin, C. (2016). XGBoost: A scalable tree boosting system. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 785–794).

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., et al. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12, 2825–2830.

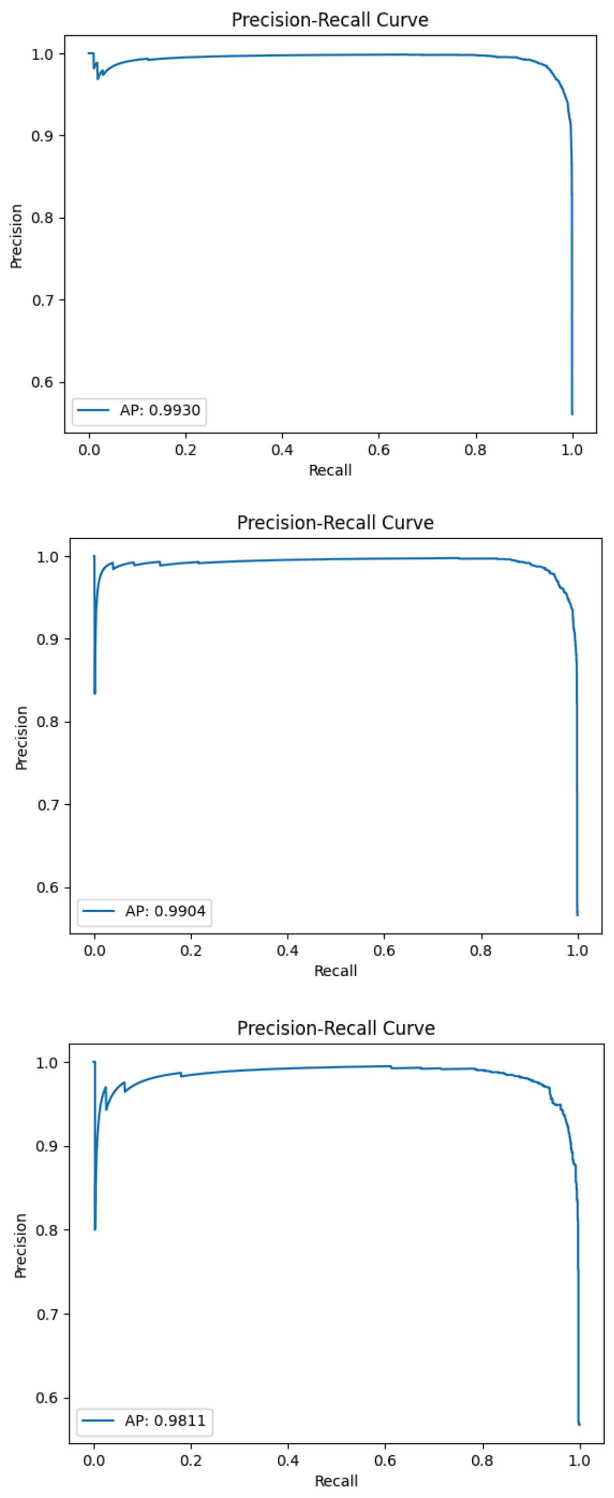


*Figure 3*. Performance reduction as training partition increases for SVM meta model (20/80, 50/50, 80/20) on PHISH problem

## APPENDIX

*Table A1*. Performance Metrics for WINE problem. Including mean for each data splits and aggregate mean.

| MODEL | TRAIN% | ACC | ROC-AUC | APR | MEAN/SPLIT | MEAN |
|---|---|---|---|---|---|---|
| GDBT | 80% | 0.75 | 0.81 | 0.88 | 0.813 | 0.818 |
|  | 50% | 0.76 | 0.82 | 0.88 | 0.820 |  |
|  | 20% | 0.76 | 0.82 | 0.88 | 0.820 |  |
| NN | 80% | 0.76 | 0.83 | 0.9 | 0.830 | 0.814 |
|  | 50% | 0.75 | 0.81 | 0.88 | 0.813 |  |
|  | 20% | 0.73 | 0.8 | 0.87 | 0.800 |  |
| RF | 80% | 0.82 | 0.89 | 0.94 | 0.883 | 0.863 |
|  | 50% | 0.81 | 0.88 | 0.93 | 0.873 |  |
|  | 20% | 0.77 | 0.84 | 0.89 | 0.833 |  |
| SGD | 80% | 0.73 | 0.78 | 0.85 | 0.787 | 0.790 |
|  | 50% | 0.74 | 0.79 | 0.86 | 0.797 |  |
|  | 20% | 0.73 | 0.78 | 0.85 | 0.787 |  |
| SVM | 80% | 0.81 | 0.98 | 0.93 | 0.907 | 0.946 |
|  | 50% | 0.92 | 0.97 | 0.98 | 0.957 |  |
|  | 20% | 0.95 | 0.98 | 0.99 | 0.973 |  |

*Table A2*. Performance Metrics for PHISH problem. Including mean for each data splits and aggregate mean.

| MODEL | TRAIN% | ACC | ROC-AUC | APR | MEAN/SPLIT | MEAN |
|---|---|---|---|---|---|---|
| GDBT | 80% | 0.95 | 0.99 | 0.99 | 0.977 | 0.971 |
|  | 50% | 0.94 | 0.99 | 0.99 | 0.973 |  |
|  | 20% | 0.93 | 0.98 | 0.98 | 0.963 |  |
| NN | 80% | 0.93 | 0.98 | 0.99 | 0.967 | 0.962 |
|  | 50% | 0.92 | 0.98 | 0.99 | 0.963 |  |
|  | 20% | 0.91 | 0.98 | 0.98 | 0.957 |  |
| RF | 80% | 0.95 | 0.99 | 0.99 | 0.977 | 0.971 |
|  | 50% | 0.94 | 0.99 | 0.99 | 0.973 |  |
|  | 20% | 0.93 | 0.98 | 0.98 | 0.963 |  |
| SGD | 80% | 0.92 | 0.97 | 0.98 | 0.957 | 0.954 |
|  | 50% | 0.91 | 0.97 | 0.98 | 0.953 |  |
|  | 20% | 0.91 | 0.97 | 0.98 | 0.953 |  |
| SVM | 80% | 0.94 | 0.98 | 0.98 | 0.967 | 0.976 |
|  | 50% | 0.96 | 0.99 | 0.99 | 0.980 |  |
|  | 20% | 0.96 | 0.99 | 0.99 | 0.980 |  |

*Table A3*. Performance Metrics for SEPSIS problem. Including mean for each data splits and aggregate mean.

| MODEL | TRAIN% | ACC | ROC-AUC | APR | MEAN/SPLIT | MEAN |
|---|---|---|---|---|---|---|
| GDBT | 80% | 0.54 | 0.69 | 0.97 | 0.733 | 0.741 |
| | 50% | 0.59 | 0.69 | 0.96 | 0.747 | |
| | 20% | 0.59 | 0.68 | 0.96 | 0.743 | |
| NN | 80% | 0.93 | 0.7 | 0.97 | 0.867 | 0.867 |
| | 50% | 0.93 | 0.7 | 0.97 | 0.867 | |
| | 20% | 0.93 | 0.7 | 0.97 | 0.867 | |
| RF | 80% | 0.54 | 0.69 | 0.96 | 0.730 | 0.742 |
| | 50% | 0.59 | 0.68 | 0.96 | 0.743 | |
| | 20% | 0.63 | 0.67 | 0.96 | 0.753 | |
| SGD | 80% | 0.57 | 0.7 | 0.97 | 0.747 | 0.748 |
| | 50% | 0.57 | 0.7 | 0.97 | 0.747 | |
| | 20% | 0.58 | 0.7 | 0.97 | 0.750 | |
| SVM | 80% | 0.93 | 0.4 | 0.91 | 0.747 | 0.722 |
| | 50% | 0.93 | 0.31 | 0.89 | 0.710 | |
| | 20% | 0.93 | 0.31 | 0.89 | 0.710 | |

*Table A4*. Performance Metrics for CDC problem. Including mean for each data splits and aggregate mean.

| MODEL | TRAIN% | ACC | ROC-AUC | APR | MEAN/SPLIT | MEAN |
|---|---|---|---|---|---|---|
| GDBT | 80% | 0.86 | 0.82 | 0.41 | 0.697 | 0.689 |
| | 50% | 0.86 | 0.82 | 0.4 | 0.693 | |
| | 20% | 0.85 | 0.8 | 0.38 | 0.677 | |
| NN | 80% | 0.65 | 0.83 | 0.42 | 0.633 | 0.628 |
| | 50% | 0.64 | 0.82 | 0.41 | 0.623 | |
| | 20% | 0.65 | 0.82 | 0.41 | 0.627 | |
| RF | 80% | 0.82 | 0.81 | 0.38 | 0.670 | 0.674 |
| | 50% | 0.82 | 0.81 | 0.4 | 0.677 | |
| | 20% | 0.81 | 0.82 | 0.4 | 0.677 | |
| SGD | 80% | 0.73 | 0.82 | 0.4 | 0.650 | 0.649 |
| | 50% | 0.73 | 0.82 | 0.4 | 0.650 | |
| | 20% | 0.73 | 0.82 | 0.39 | 0.647 | |
| SVM | 80% | 0.84 | 0.68 | 0.32 | 0.613 | 0.752 |
| | 50% | 0.9 | 0.86 | 0.65 | 0.803 | |
| | 20% | 0.91 | 0.9 | 0.71 | 0.840 | |