A Comparison of Supervised Machine Learning Algorithms

Ahmad Sadeed **asadeed@ucsd.edu**

Abstract

The goal of this paper is to compare six supervised machine learning algorithms on binary classification tasks. Four balanced and imbalanced datasets are used to evaluate logistic regression, decision trees, random forests, gradient boosting, neural networks, and naïve bayes algorithms using accuracy, average precision and receiver operating characteristic scores. Each classifier was tuned to wide ranges of values and hyperparameters during five trials and 5-fold cross validation using grid search. The results match experiments performed by Caruana and Niculescu-Mizil on a broader set of algorithms, problems, and performance metrics. Alternatives for null hypothesis statistical testing is proposed.

1. Introduction

As the No Free Lunch Theorem states, there is no one classifier that performs best on all datasets and on all metrics (Caruana and Niculescu-Mizil, 2006). Caruana and Niculescu-Mizil (referred to as CNM06 in this paper) experimented to test if there are classifier that generally do well on most of the problem sets or that tend to do poorly across most of the problems. Following CNM06, this paper attempts to replicate their experiment using logistic regression, decision trees, random forests, gradient boosting, neural networks, and naïve bayes classifiers on four problems. The metrics used include accuracy (ACC), average precision (APR) and receiver operating characteristic (ROC) score. Different hyperparameters were explored for each algorithm. This paper used null hypothesis statistical testing (NHST) with p = 0.05 to compare classifiers. In the discussion section, the shortcomings of NHST and alternative approaches are discussed.

2. Methodology

2.1 Algorithms

The following classifiers from Scikit-Learn library (Scikit-Learn, 2019) were used during this experiment:

Logistic Regression (LOGIT): The "liblinear" solver of Scikit-Learn implementation of logistic regression was used. This solver requires fewer iterations to converge and needs the "penalty" parameter to be set. The default "L2" penalty was used. The parameter for inverse of regularization strength or "C" was validated on values from 10^{-8} to 10^{4}.

Decision Tree (TREES): The quality of split was measured using "gini" for impurity and "entropy" for information gain. The "max_depth" (maximum depth of the tree) parameter was cross validated on values in [1, 4, 7, 10, 13, 16, 19, 22, 25]. The "min_samples_split" (minimum number of samples required to split an internal node) was tested on values in [1, 3, 5, 7, 9]. The Minimal Cost-Complexity Pruning parameter "ccp_alpha" gave best result with the default value of 0.0 and was not used in second run of the process. The "min_samples_leaf" parameter (minimum number of samples required to be at a leaf node) which has the effect of smoothing the model was tested on values in [1,3,5,7].

Random Forest (FOREST): Number of trees in the forest or "n_estimators" was tested on values in [2, 4, 8, 16, 32, 64, 128, 256, 512, 1024]. Number of features to consider or "max_features" was evaluated on values in [1,2,4,6,8,12,16,20].

Gradient Boosting (GBOOST): Number of boosting steps (n_estimators) was tested on values in [2,4,8,16,32,64,128,256,512,1024,2048]. In addition, the default "max_depth" of 3 was replaced with values in [1, 3, 5, 7, 9] for further cross validation.

Multi-Layer Perceptron (MLP_ADAM): The "hidden_layer_sizes" or the number of neurons in the only hidden layer was evaluated with values in [1,2,4,8,32,128,256]. Scikit-Learn uses Adam optimizer as default, which is a stochastic gradient-based optimizer and is usually faster and gives better results than "SGD" optimizer.

Naive Bayes (NB): The additive Laplacian smoothing parameter or "alpha" ranged in values from 10^{-4} to 10^{4}.

The datasets were scaled to a mean of 0 and standard deviation of 1 for the LOGIT and MLP_ADAM classifiers. The remaining classifiers were trained on non-scaled datasets similar to CNM06.

2.2 Performance Metrics

Accuracy score (ACC), average precision score (APR), and receiver operating characteristic score (ROC AUC) were used to compare performances of classifiers. ACC belongs to the category of threshold metrics. APR and ROC belong to ordering/rank metrics. In addition to ACC, it is crucial that we evaluate classifiers using other metrics to account for shortcomings of ACC as it is possible to achieve high accuracy rate in cases with class imbalance by always choosing the majority class. Average precision (Scikit-Learn) provides the weighted mean of precisions at each threshold. P_n

and R_n are the precision and recall at nth threshold and the increase in recall from previous threshold $(R_n - R_{n-1})$ is used as the weight:

$$APR = \sum_n (R_n - R_{n-1})P_n$$

Similar to ACC, APR is between 0 and 1 where a higher score is preferred. Receiver operating characteristic (ROC) score computes a single value from the ROC curve and shows the performance of a binary classifier as its discrimination threshold is changed (Wikipedia, 2019). ROC curve incorporates a model's true positive rate (TPR), also known as sensitivity or recall and false positive rate (FPR), also known false alarm and [1 – specificity]. ROC AUC score can assist in identifying optimal and suboptimal classifiers independent from class distribution.

2.3 Data Sets

As shown in Table 1, the six classifiers were evaluated on four data sets from UCI repository: LETTER, ADULT, and COVTYPE. LETTER was converted to create two different datasets. LETTER1 (which is LETTER.P2 in CNM06) was generated by converting letter A-M as positive and the rest as negative class. LETTER2 (LETTER.P1 in CNM06) was generated by treating letter "O" as positive and the remaining letters as negative. LETTER1 is well balanced with 50% positive samples. LETTER2 is an imbalanced set with only 4% positive samples. ADULT was turned into a classification problem by mapping INCOME column to 0 for INCOME <= 50K and 1 for INCOME > 50K, leading to 25% positive samples. For COVTYPE, the largest class for the COV column was mapped to 1 and the rest to 0. COVTYPE had a sample size of over 550K and a smaller subset of dataset was used in this experiment. Categorical variables in ADULT were one-hot-encoded leading to 104 attributes.

Table 1: Description of Datasets

DATASET	#ATTR	TRAIN SIZE	TEST SIZE	%POZ
ADULT	14/104	5000	40222	25%
LETTER1	16	5000	15000	50%
LETTER2	16	5000	15000	4%
COVTYPE	54	5000	24051	49%

3. Experiments and Results

Each classifier was evaluated on each dataset for five trials. In each of the five trials, random 5000 samples were selected from the data for the training set and the remaining large portion of the data was assigned to the test set. Within each trial, the 5000 samples were split into five folds to run cross validation using grid search method to find the best hyperparameter settings. Through using the "multiple metric evaluation" setup within Scikit-Learn GridSearchCV, separate best hyperparameter settings for ACC, APR, and ROC AUC were stored and then utilized to train the models on the 5000 samples as a whole. Table 4 in the appendix section contains the mean training set performance on each metric averaged over four datasets.

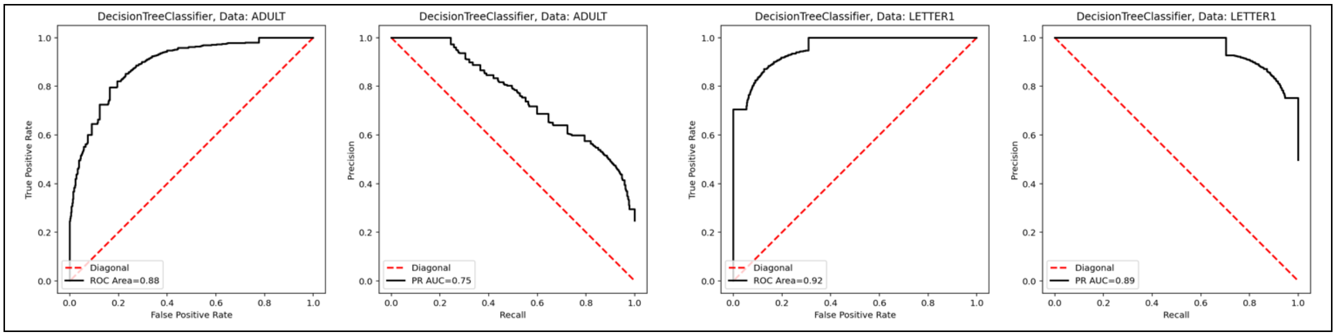
Table 2: Test Set Performance (over four datasets)

MODEL	ACC	APR	ROC	MEAN
LOGIT	0.822754	0.623291	0.850184	0.76540967
TREES	0.867867*	0.779514	0.893362	0.84691433
FOREST	0.900355*	0.897413*	0.946725*	0.914831*
GBOOST	**0.907358**	**0.906995**	**0.950503**	**0.92161867***
MLP_ADAM	0.893842*	0.881304*	0.938188*	0.90444467*
NB	0.749619	0.502576	0.703253	0.651816

The best hyperparameter setting for each metric was then used to test classifiers on the test set. Table 2 contains the test set performance of classifiers averaged over the four datasets. Table 3 includes the test set performance on each dataset averaged over three metrics. In tables 2, 3 and 4, classifiers with best performance in each column is **boldfaced**. Classifiers that are not statistically distinguishable from the best classifier in that column at p = 0.05 are *'ed using ttest_ind from SciPy's stats module on five trials (SciPy.org).

Similar to CNM06, gradient boosting classifier outperformed other algorithms on each metric. Results from random forest and multi-layer perceptron using Adam optimizer were not statistically distinguishable from GBOOST. Surprisingly, decision tree classifier had a higher accuracy than what is achieved in CNM06. This may be due to tuning the hyperparameters for max depth, min_samples_split and min_samples_leaf of the DT. In addition, DT has good performance for LETTER1 and LETTER2, which are easier dataset to classify compared to other datasets. This is also evident in figure 1. DT has higher ROC and precision-recall (PR) score for LETTER1 compared to ADULT dataset. ROC and PR curves for all classifier/dataset combination is included in the appendix section.

Figure1: DT ROC and PR Curve



Naïve bayes and logistic regression are among the poor performing classifiers. This distinction becomes more evident after comparing APR scores for each classifier. Even though NB and LOGIT appear to have "good" ACC, their APR is much worse. This proves the point of using a multi metric evaluation setup. Except MLP on LETTER2, gradient boosting classifier has best performance for all datasets. Neural networks outperform other algorithms on the imbalanced LETTER2 dataset. As in CNM06, NB and LOGIT do poorly on all datasets, except LOGIT on ADULT dataset, which is also an imbalanced dataset with 25% positive class. On the training set, shown in table 4, random forest does better than GBOOST on all metrics. Boosting is more susceptible to be affected by the initial iterations of boosting (CNM06). Besides, random forest is more prone to overfitting.

Table 3: Test Set Performance (over all metrics)

MODEL	ADULT	LETTER1	LETTER2	COVTYPE	MEAN
LOGIT	0.832672*	0.784137	0.650625	0.794205	0.76540975
TREES	0.817271	0.908687	0.879192	0.782507	0.84691425
FOREST	0.839766*	0.975356*	0.976052*	0.868150*	0.914831*
GBOOST	**0.862398**	**0.979594**	0.976011*	**0.868472**	**0.92161875**
MLP_ADAM	0.816646	0.975062*	**0.982937**	0.843133*	0.9044445*
NB	0.777775	0.573208	0.543166	0.713114	0.65181575

4. Discussion and Conclusion

Gradient boosting outperformed all classifiers on the three metrics and was the best on all datasets except LETTER2, in which neural network classifier did best. On ADULT dataset, LOGIT and DT performed better than neural network classifier. This solidifies the point of No Free Lunch Theorem. It was also noteworthy that decision tree classifier performed much better than what was reported on CNM06. Thus, cross validation on different hyperparameter settings is key to finding the true difference among classifiers performance. It is also important to evaluate models on different metrics to get a better measurement of each model's features.

Comparison of the classifiers depended on use of null hypothesis statistical testing (NHST). This paper used p-value of 0.05 or 95% confidence interval. However, this **does not** mean that one classifier outperforms another classifier with 95% probability (Benavoli et al., 2017). According to Benavoli et al, it is “the probability of getting the observed (or a larger) difference between classifiers if the null hypothesis of equivalence was true.” To find if one classifier is better than another or to get the probability of a classifier being better (posterior probabilities), a better approach is to use Bayesian correlated t-test, Bayesian signed rank test or a Bayesian hierarchical model (2017). Similar to realizing the shortcomings of using a single metric to evaluate an algorithm, this paper realizes the shortcomings of using statistical tests that do not appropriately answer the proposed hypothesis.

5. Bonus Points

An additional three classifiers were used to get a better comparison. Averaged over trials, each algorithm and dataset combination was evaluated by plotting ROC and PR curves. The plots are provided in the appendix section.

6. Resources

Benavoli, A., Corani, G., Demšar, J., & Zaffalon, M. (2017). Time for a change: A tutorial for comparing multiple classifiers through bayesian analysis. *Journal of Machine Learning Research*, *18*(77), 1–36. https://jmlr.org/papers/v18/16-305.html

Metrics and scoring: quantifying the quality of predictions — scikit-learn 0.22.1 documentation. (n.d.). Scikit-Learn.org. https://scikit-learn.org/stable/modules/model_evaluation.html

R. Caruana, A. Niculescu-Mizeil, *An Empirical Comparison of Supervised Learning Algorithms*, 2008

Scikit-Learn. (2019). *1. supervised learning - scikit-learn 0.21.3 documentation*. Scikit-Learn.org. https://scikit-learn.org/stable/supervised_learning.html#supervised-learning

Scipy.stats.ttest_ind — SciPy v1.4.1 Reference Guide. (n.d.). Docs.scipy.org. https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.ttest_ind.html

Sklearn.metrics.average_precision_score — scikit-learn 0.24.1 documentation. (n.d.). Scikit-Learn.org. Retrieved March 18, 2021, from https://scikit-learn.org/stable/modules/generated/sklearn.metrics.average_precision_score.html

Sklearn.model_selection.GridSearchCV — scikit-learn 0.22 documentation. (2019). Scikit-Learn.org. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

UCI Machine Learning Repository. (2018). Uci.edu. https://archive.ics.uci.edu/ml/index.php

Wikipedia. (2019, March 20). *Receiver operating characteristic*. Wikipedia; Wikimedia Foundation. https://en.wikipedia.org/wiki/Receiver_operating_characteristic

7. Appendix

P-values for *Table 2*

MODEL	ACC	APR	ROC
LOGIT	0.002565	1.68E-04	9.81E-10
TREES	0.110256	4.99E-06	4.54E-04
FOREST	0.754209	6.98E-01	7.95E-01
GBOOST	1.0	1.0	1.0
MLP_ADAM	0.574147	3.74E-01	4.47E-01
NB	0.000155	1.48E-07	5.97E-11

P-values for *Table 3*

MODEL	ADULT	LETTER1	LETTER2	COVTYPE
LOGIT	0.13882	5.41E-16	0.002483	8.57E-07
TREES	0.044035	5.18E-11	0.003152	3.38E-08
FOREST	0.222889	0.5732501	0.37994	0.9801786
GBOOST	1.0	1.0	0.389911	1.0
MLP_ADAM	0.043496	0.5360766	1	0.05236602
NB	0.001434	1.25E-33	0.000155	2.44E-13

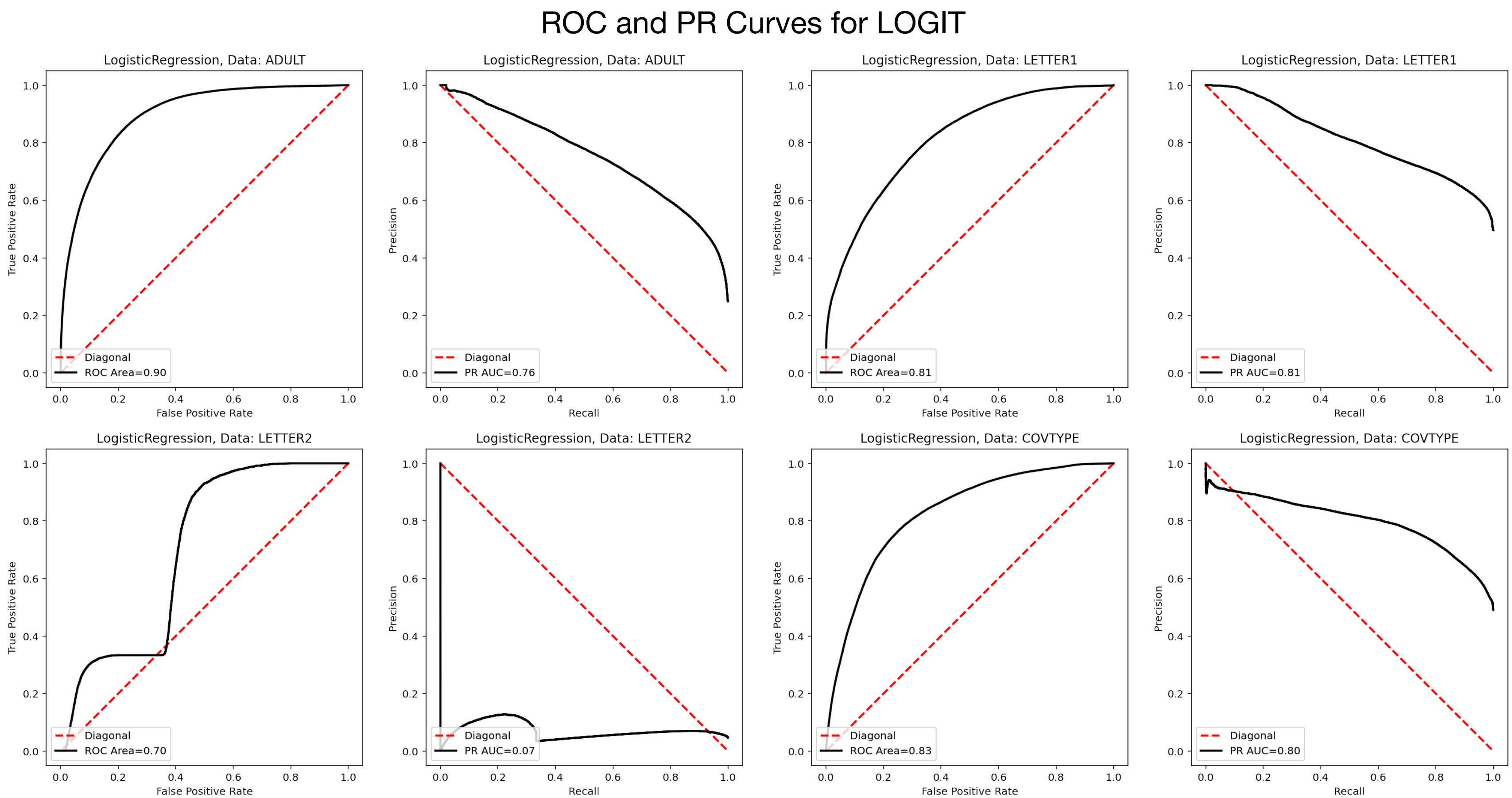
Table 4: Training Set Performance (over four datasets)

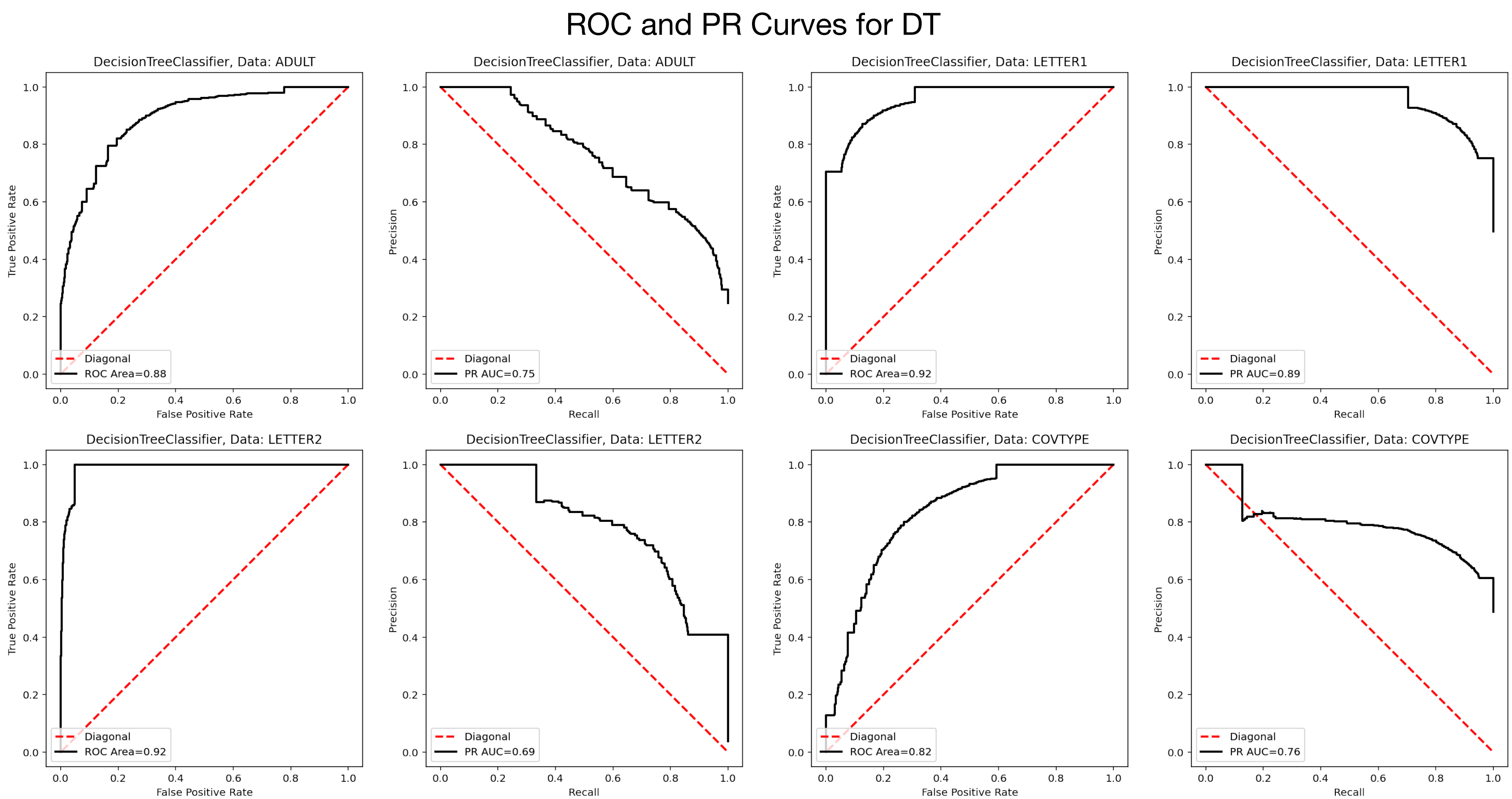
MODEL	ACC	APR	ROC	MEAN
LOGIT	0.82566	0.633612	0.856742	0.77200467
TREES	0.91844	0.896812	0.945291	0.920181
FOREST	**0.99996**	**1.000000**	**1.000000**	**0.99998667**
GBOOST	0.97645	0.979642	0.989608	0.9819
MLP_ADAM	0.93330	0.934849	0.970080	0.94607633
NB	0.75123	0.508511	0.708125	0.65595533

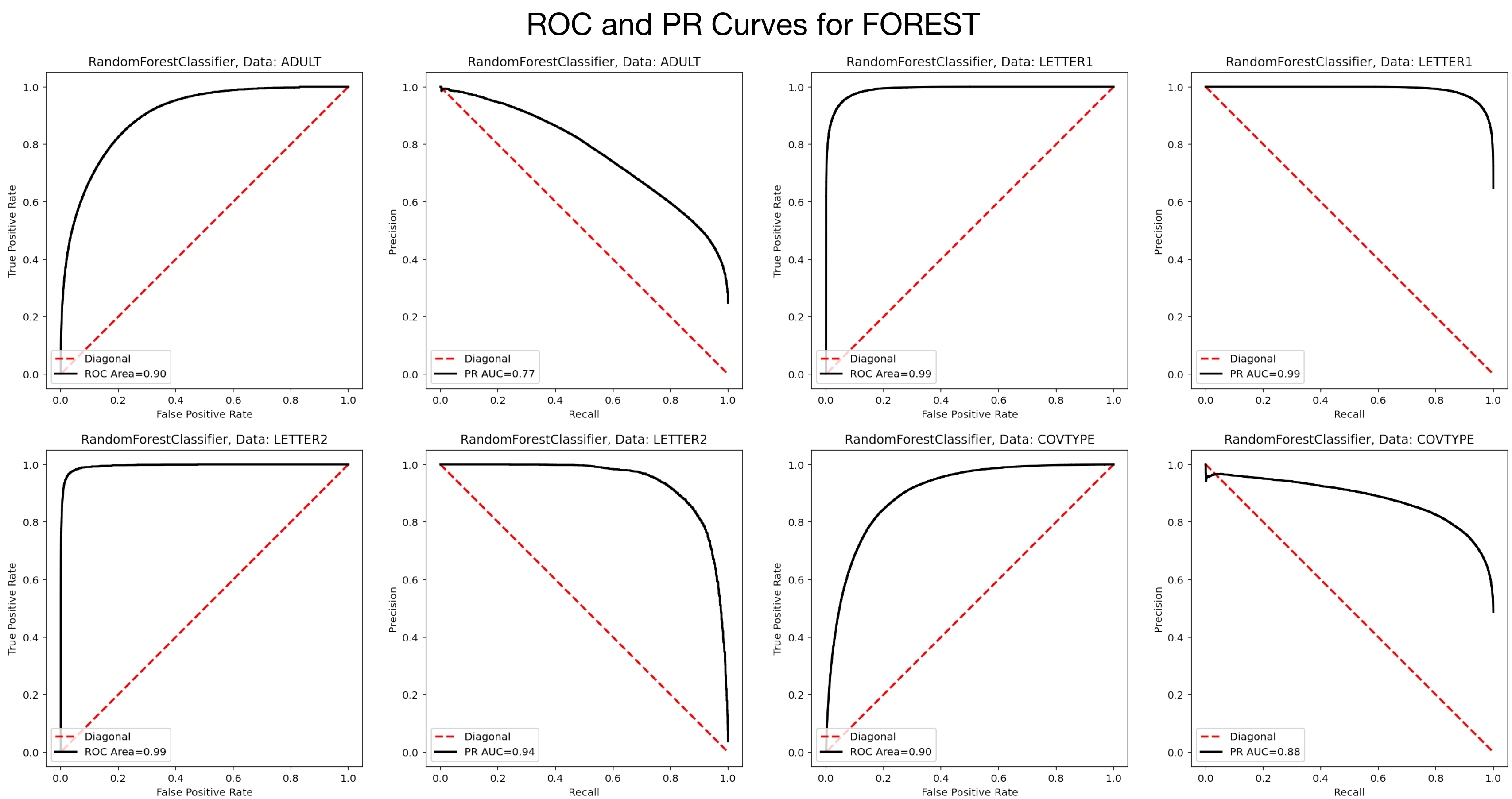
P-values for *Table 4*

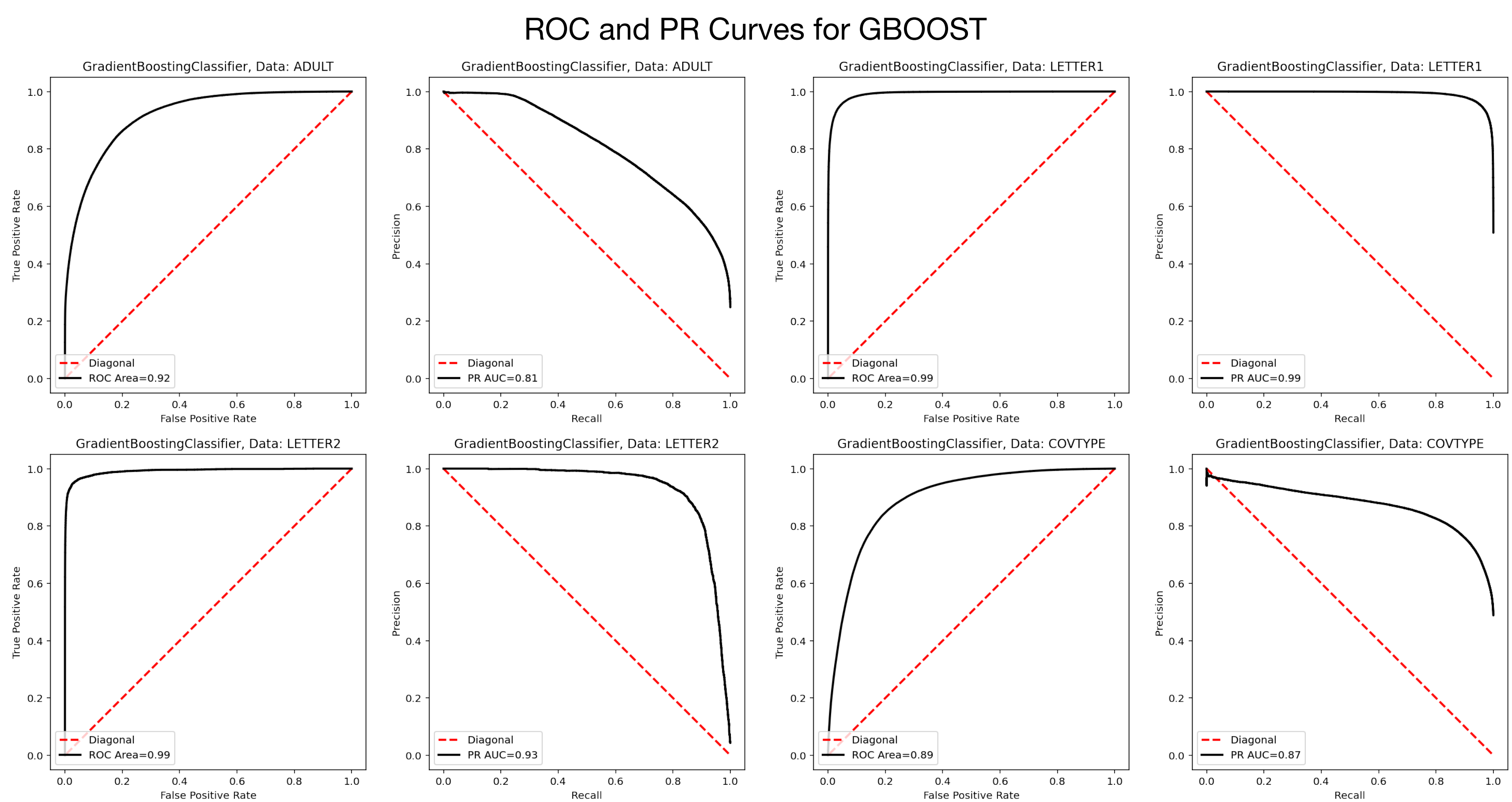
MODEL	ACC	APR	ROC
LOGIT	4.30E-10	3.07E-06	2.98E-19
TREES	2.83E-05	6.06E-07	1.99E-06
FOREST	1.0	1.0	1.0
GBOOST	1.68E-02	1.77E-02	2.15E-02
MLP_ADAM	6.83E-05	3.03E-03	5.91E-04
NB	9.88E-09	9.69E-10	1.27E-13

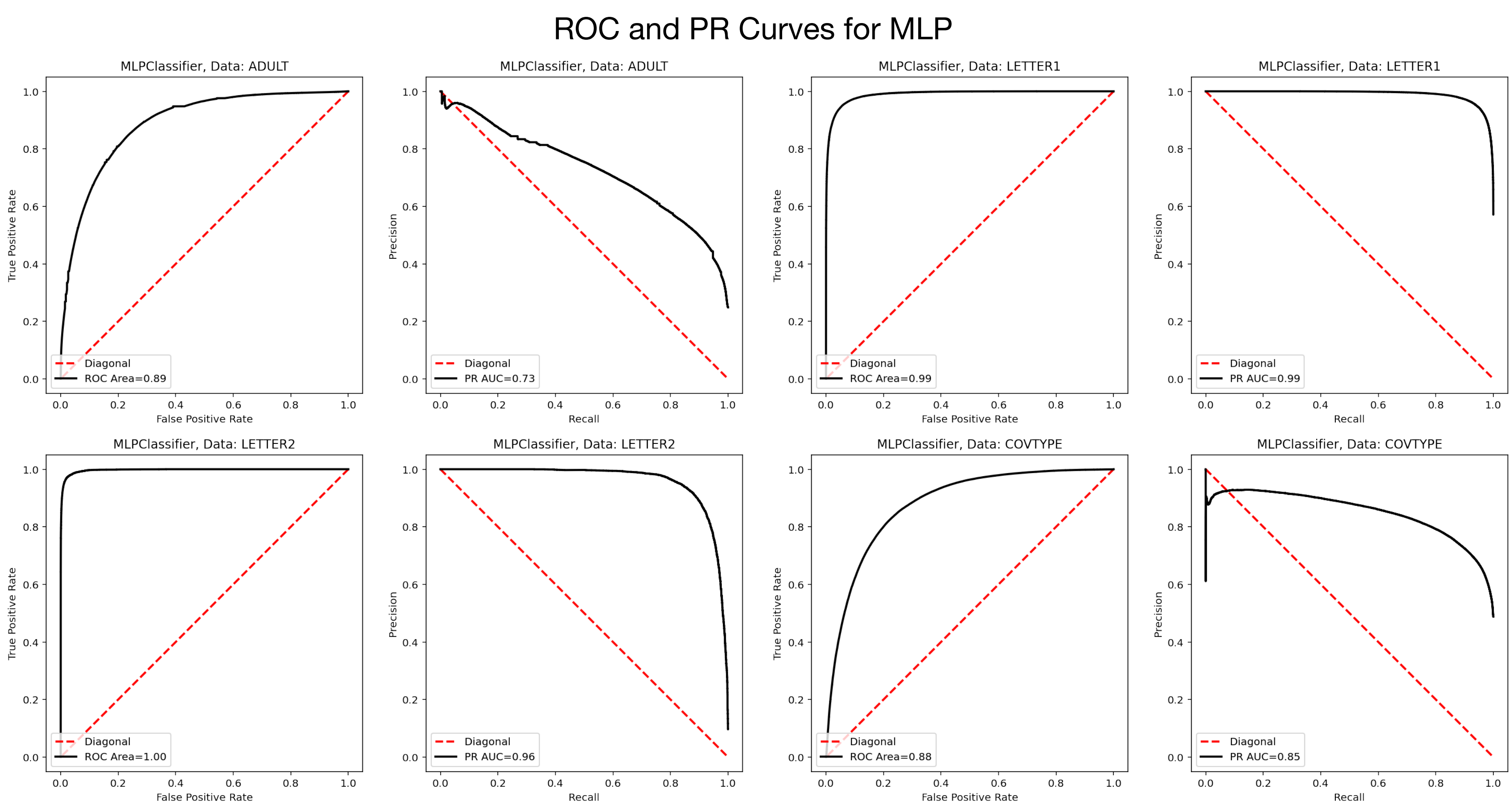
Raw test data: each mini table has 5 rows (one per trial) and 3 columns (ACC, APR, ROC)												
	ADULT			LETTER1			LETTER2			COVTYPE		
LOGIT	0.845930	0.751137	0.895458	0.722866	0.812257	0.810990	0.963333	0.120609	0.863018	0.755145	0.805384	0.829214
	0.846626	0.760971	0.899742	0.7264	0.808626	0.811039	0.961266	0.129506	0.868506	0.758513	0.797797	0.825379
	0.844264	0.753327	0.896392	0.726133	0.811839	0.813228	0.962466	0.122606	0.855275	0.755353	0.796598	0.825491
	0.845035	0.756307	0.899148	0.727	0.815077	0.812475	0.962866	0.126001	0.865461	0.759261	0.799202	0.824225
	0.842921	0.754277	0.898543	0.732666	0.816499	0.814953	0.961933	0.127564	0.868952	0.755103	0.800224	0.826181
TREES	0.848689	0.720483	0.888564	0.8872	0.919285	0.935495	0.979733	0.735674	0.945998	0.757557	0.765087	0.82031
	0.843394	0.726092	0.887863	0.883467	0.90086	0.917541	0.9798	0.677524	0.946919	0.752734	0.759939	0.810512
	0.842225	0.734187	0.880887	0.889533	0.9186	0.936984	0.985267	0.710369	0.935891	0.766787	0.778255	0.831084
	0.840087	0.729362	0.884261	0.890067	0.909806	0.925014	0.985067	0.714158	0.929901	0.760052	0.772402	0.816605
	0.839938	0.715234	0.877794	0.880867	0.903732	0.931858	0.981	0.730317	0.950268	0.763877	0.76891	0.813494
FOREST	0.849286	0.775516	0.901979	0.943267	0.990215	0.989852	0.989933	0.943158	0.994931	0.989933	0.943158	0.994931
	0.845731	0.76848	0.901455	0.943333	0.98943	0.989075	0.9846	0.942913	0.996178	0.9846	0.942913	0.996178
	0.846626	0.770349	0.898839	0.949	0.992283	0.992149	0.989	0.943758	0.996489	0.989	0.943758	0.996489
	0.846825	0.771289	0.896636	0.944	0.990041	0.989861	0.987667	0.932388	0.992746	0.987667	0.932388	0.992746
	0.847894	0.774702	0.900875	0.9458	0.991173	0.990859	0.989267	0.96014	0.997609	0.989267	0.96014	0.997609
GBOOST	0.863483	0.812984	0.915965	0.951667	0.991293	0.991549	0.990467	0.93451	0.994928	0.822045	0.882584	0.901599
	0.8604	0.811109	0.916242	0.9546	0.991766	0.991678	0.991267	0.938804	0.993377	0.817804	0.879001	0.898991
	0.859455	0.807542	0.915535	0.9584	0.993208	0.9934	0.991067	0.943328	0.992075	0.825828	0.886026	0.90065
	0.861668	0.810376	0.915894	0.9516	0.991491	0.990816	0.9914	0.935222	0.992002	0.821005	0.882722	0.900694
	0.862215	0.808667	0.91444	0.957133	0.992937	0.992366	0.991667	0.963394	0.996656	0.823999	0.882935	0.901195
MLP	0.829297	0.736217	0.886896	0.944333	0.989131	0.989454	0.9924	0.952738	0.995948	0.804624	0.859977	0.880731
	0.844612	0.686086	0.891674	0.9456	0.989927	0.989392	0.9914	0.953962	0.997259	0.797264	0.846477	0.875997
	0.83243	0.726612	0.886984	0.946	0.989089	0.988775	0.9924	0.958694	0.997453	0.80163	0.849867	0.877888
	0.833648	0.741592	0.883633	0.948333	0.990061	0.990287	0.993533	0.967743	0.997827	0.797306	0.847377	0.879204
	0.841952	0.739644	0.888417	0.9464	0.989448	0.989699	0.9928	0.962543	0.997357	0.800881	0.848892	0.878879
NB	0.794341	0.670979	0.862656	0.559267	0.580243	0.577878	0.963133	0.048524	0.618547	0.691115	0.717526	0.757941
	0.798966	0.672797	0.863726	0.561467	0.577406	0.583256	0.961267	0.052104	0.62179	0.684961	0.698577	0.750045
	0.796803	0.675708	0.864627	0.563733	0.583046	0.588274	0.962467	0.051065	0.622542	0.673735	0.712648	0.751214
	0.794317	0.675698	0.863268	0.558	0.579082	0.581946	0.962667	0.050235	0.620336	0.675772	0.707372	0.747155
	0.798692	0.673045	0.861006	0.5538	0.574705	0.576023	0.961933	0.04751	0.603373	0.675939	0.703254	0.749454


ROC and PR Curves for LOGIT
LogisticRegression, Data: ADULT
True Positive Rate
False Positive Rate
Diagonal
ROC Area=0.90
LogisticRegression, Data: ADULT
Precision
Recall
Diagonal
PR AUC=0.76
LogisticRegression, Data: LETTER1
True Positive Rate
False Positive Rate
Diagonal
ROC Area=0.81
LogisticRegression, Data: LETTER1
Precision
Recall
Diagonal
PR AUC=0.81
LogisticRegression, Data: LETTER2
True Positive Rate
False Positive Rate
Diagonal
ROC Area=0.70
LogisticRegression, Data: LETTER2
Precision
Recall
Diagonal
PR AUC=0.07
LogisticRegression, Data: COVTYPE
True Positive Rate
False Positive Rate
Diagonal
ROC Area=0.83
LogisticRegression, Data: COVTYPE
Precision
Recall
Diagonal
PR AUC=0.80

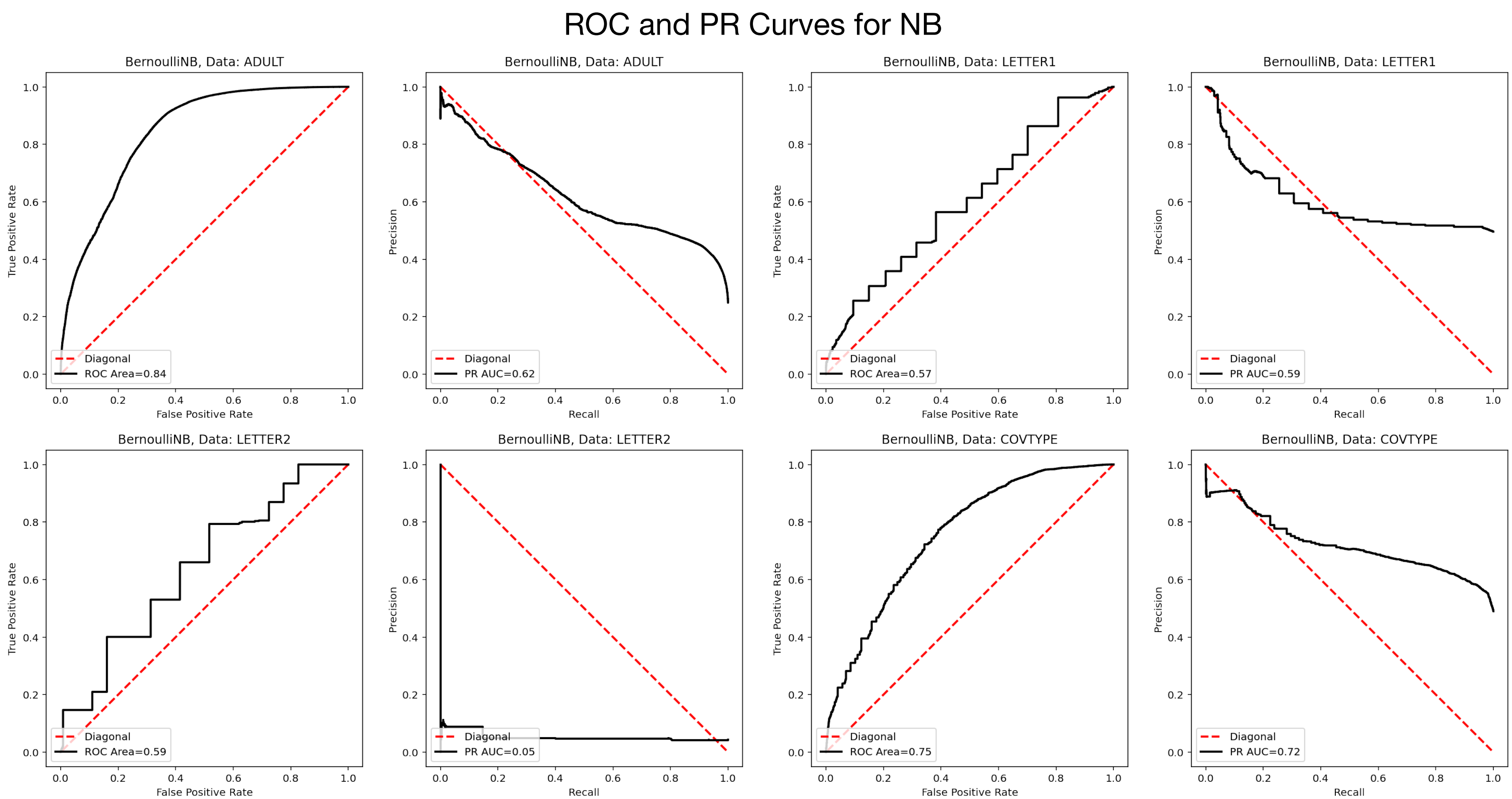

ROC and PR Curves for DT
DecisionTreeClassifier, Data: ADULT
True Positive Rate
False Positive Rate
Diagonal
ROC Area=0.88
DecisionTreeClassifier, Data: ADULT
Precision
Recall
Diagonal
PR AUC=0.75
DecisionTreeClassifier, Data: LETTER1
True Positive Rate
False Positive Rate
Diagonal
ROC Area=0.92
DecisionTreeClassifier, Data: LETTER1
Precision
Recall
Diagonal
PR AUC=0.89
DecisionTreeClassifier, Data: LETTER2
True Positive Rate
False Positive Rate
Diagonal
ROC Area=0.92
DecisionTreeClassifier, Data: LETTER2
Precision
Recall
Diagonal
PR AUC=0.69
DecisionTreeClassifier, Data: COVTYPE
True Positive Rate
False Positive Rate
Diagonal
ROC Area=0.82
DecisionTreeClassifier, Data: COVTYPE
Precision
Recall
Diagonal
PR AUC=0.76


ROC and PR Curves for FOREST
RandomForestClassifier, Data: ADULT
True Positive Rate
False Positive Rate
Diagonal
ROC Area=0.90
RandomForestClassifier, Data: ADULT
Precision
Recall
Diagonal
PR AUC=0.77
RandomForestClassifier, Data: LETTER1
True Positive Rate
False Positive Rate
Diagonal
ROC Area=0.99
RandomForestClassifier, Data: LETTER1
Precision
Recall
Diagonal
PR AUC=0.99
RandomForestClassifier, Data: LETTER2
True Positive Rate
False Positive Rate
Diagonal
ROC Area=0.99
RandomForestClassifier, Data: LETTER2
Precision
Recall
Diagonal
PR AUC=0.94
RandomForestClassifier, Data: COVTYPE
True Positive Rate
False Positive Rate
Diagonal
ROC Area=0.90
RandomForestClassifier, Data: COVTYPE
Precision
Recall
Diagonal
PR AUC=0.88


ROC and PR Curves for GBOOST
GradientBoostingClassifier, Data: ADULT
True Positive Rate
False Positive Rate
Diagonal
ROC Area=0.92
GradientBoostingClassifier, Data: ADULT
Precision
Recall
Diagonal
PR AUC=0.81
GradientBoostingClassifier, Data: LETTER1
True Positive Rate
False Positive Rate
Diagonal
ROC Area=0.99
GradientBoostingClassifier, Data: LETTER1
Precision
Recall
Diagonal
PR AUC=0.99
GradientBoostingClassifier, Data: LETTER2
True Positive Rate
False Positive Rate
Diagonal
ROC Area=0.99
GradientBoostingClassifier, Data: LETTER2
Precision
Recall
Diagonal
PR AUC=0.93
GradientBoostingClassifier, Data: COVTYPE
True Positive Rate
False Positive Rate
Diagonal
ROC Area=0.89
GradientBoostingClassifier, Data: COVTYPE
Precision
Recall
Diagonal
PR AUC=0.87



ROC and PR Curves for MLP
MLPClassifier, Data: ADULT
ROC Area=0.89
PR AUC=0.73
MLPClassifier, Data: LETTER1
ROC Area=0.99
PR AUC=0.99
MLPClassifier, Data: LETTER2
ROC Area=1.00
PR AUC=0.96
MLPClassifier, Data: COVTYPE
ROC Area=0.88
PR AUC=0.85
Diagonal
True Positive Rate
False Positive Rate
Precision
Recall


ROC and PR Curves for NB
BernoulliNB, Data: ADULT
True Positive Rate
False Positive Rate
Diagonal
ROC Area=0.84
BernoulliNB, Data: ADULT
Precision
Recall
Diagonal
PR AUC=0.62
BernoulliNB, Data: LETTER1
True Positive Rate
False Positive Rate
Diagonal
ROC Area=0.57
BernoulliNB, Data: LETTER1
Precision
Recall
Diagonal
PR AUC=0.59
BernoulliNB, Data: LETTER2
True Positive Rate
False Positive Rate
Diagonal
ROC Area=0.59
BernoulliNB, Data: LETTER2
Precision
Recall
Diagonal
PR AUC=0.05
BernoulliNB, Data: COVTYPE
True Positive Rate
False Positive Rate
Diagonal
ROC Area=0.75
BernoulliNB, Data: COVTYPE
Precision
Recall
Diagonal
PR AUC=0.72

In [ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

Adult data set cleaning:

In [ ]:
```
import numpy as np
import pandas as pd
%config InlineBackend.figure_format = 'retina'
pd.options.mode.chained_assignment = None
```

In [ ]:
```
adult_data_UCI = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/d
ata/UCI-adult.csv')
```

In [ ]:
```
def binarize(income):
    if (income == '<=50K'): return 0
    else: return 1
```

In [ ]:
```
adult_data_UCI['income'] = adult_data_UCI['income'].apply(binarize)
adult_data_UCI.rename(columns={'income':'y'}, inplace=True)
```

In [ ]:
```
adult_data_UCI.columns
```

In [ ]:
```
# del adult_data_UCI['fnlwgt']
adult_data_UCI.shape
```

In [ ]:
```
adult_data_UCI = adult_data_UCI.replace('?', np.nan).dropna(axis = 0,
how = 'any')
adult_data_UCI.shape
```

In [ ]:
```
adult_data_UCI = pd.get_dummies(adult_data_UCI)

y = adult_data_UCI.pop('y')
adult_data_UCI['y'] = y
adult_data_UCI = adult_data_UCI.reset_index().drop('index', axis = 1)
# adult_data_UCI.to_csv('/content/drive/MyDrive/Colab Notebooks/data/c
leaned/adult.csv')
```

In [ ]:
```
adult_data_UCI.shape
```

In [ ]:
```
print(adult_data_UCI.y.value_counts())
```

In [ ]:
```
adult_data_UCI.describe()
```

Letter data set cleaning:

In [ ]:
```
letter_p1_data = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/d
ata/UCI-Letter.csv', header=None)
letter_p2_data = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/d
ata/UCI-Letter.csv', header=None)
letter_p1_data.drop([0], inplace=True)
letter_p2_data.drop([0], inplace=True)
```

In [ ]:
```
def option1(letter):
    if (letter <= 'M'): return 1
    else: return 0
def option2(letter):
    if (letter == 'O'): return 1
    else: return 0
letter_p1_data[0] = letter_p2_data[0].apply(option1)
letter_p2_data[0] = letter_p2_data[0].apply(option2)
```

In [ ]:
```
letter_p1_data.rename(columns={0:'y'}, inplace=True)
letter_p2_data.rename(columns={0:'y'}, inplace=True)
y1 = letter_p1_data.pop('y')
letter_p1_data['y'] = y1
y2 = letter_p2_data.pop('y')
letter_p2_data['y'] = y2
```

In [ ]:
```
letter_p1_data.to_csv('/content/drive/MyDrive/Colab Notebooks/data/cle
aned/letter_p1.csv')
letter_p2_data.to_csv('/content/drive/MyDrive/Colab Notebooks/data/cle
aned/letter_p2.csv')
```

Covtype data set cleaning:

In [ ]:
```
covtype = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/data/cov
type/covtype.data', header = None)
covtype.shape
```

In [ ]:
```
covtype[54].value_counts()
```

In [ ]:
```
def binarize_cov(cov_class):
    if (cov_class == 2): return 1
    else: return 0
```

In [ ]:
```
# only once
covtype[54] = covtype[54].apply(binarize_cov)
```

In [ ]:
```
covtype[54].value_counts()
```

In [ ]:
```
# COV TYPE has been converted to a binary problem by treating the larg
est class as the positive and the rest as negative.

covtype = covtype.reset_index().drop('index', axis = 1)
covtype[54].value_counts()
```

In [ ]:
```
covtype.rename(columns={54:'y'}, inplace=True)
covtype['y'].value_counts()
```

In [ ]:
```
covtype.shape
```

In [ ]:
```
covtype5 = covtype.sample(frac=0.05, random_state=1)
covtype5.shape
```

In [ ]:
```
covtype5.y.value_counts()
```

In [ ]:
```
covtype.to_csv('/content/drive/MyDrive/Colab Notebooks/data/cleaned/covtype.csv')
covtype5.to_csv('/content/drive/MyDrive/Colab Notebooks/data/cleaned/covtype5.csv')
```

In [ ]:

In [ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

Packages

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import time
import warnings
import importlib
from fastprogress import master_bar, progress_bar

from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.naive_bayes import BernoulliNB
from sklearn.metrics import roc_curve, roc_auc_score, classification_r
eport
from sklearn.metrics import confusion_matrix, accuracy_score, f1_scor
e, precision_score
from sklearn.metrics import recall_score, balanced_accuracy_score
from sklearn.metrics import average_precision_score, log_loss
from sklearn.metrics import precision_recall_curve, auc, average_preci
sion_score
from sklearn.neural_network import MLPClassifier
from sklearn import ensemble, preprocessing
from sklearn.preprocessing import StandardScaler
from scipy.stats import ttest_rel, ttest_ind

%config InlineBackend.figure_format = 'retina'
pd.options.mode.chained_assignment = None
pd.options.display.max_columns = None
warnings.filterwarnings('ignore')
warnings.filterwarnings("ignore", category=FutureWarning)
warnings.filterwarnings("ignore", category=DeprecationWarning)
```

Utility functions:

In [ ]:
```
# instantiate a model using the parameters and import source
def get_model(model_name:str, import_module:str, model_params:dict):
    model_class = getattr(importlib.import_module(import_module), mode
l_name)
    model = model_class(**model_params) # Instantiates the model
    return model

# returns the p_values for the each value except the best for each col
umn
def p_stats(raw_data, raw_mean, rows, cols, per_data=False):
    # get index of algo with highest performance for each metric or da
taset (in each column)
    best_algo_in_cols = raw_mean.to_numpy().argmax(axis=0)
    algo_raw = []

    for alg in raw_data:
        if per_data:
            # due to the way data is store
            algo_raw.append(np.split(alg.flatten(), cols))
        else:
            algo_raw.append(np.split(alg.flatten('F'), cols))

    p_val_test = np.ones_like(raw_mean)

    # for each metric, get the best performing algo first
    for col in range(cols):
        idx = best_algo_in_cols[col]
        # get raw data for best algo and metric(col)
        best_raw = algo_raw[idx][col]
        # run t-test between this algo(idx) and other algos
        for id_alg in range(rows):
            if (id_alg == idx):
                continue
            else:
                # run t-test between this and the best
                this_raw = algo_raw[id_alg][col]
                t_stat, p_stat = ttest_ind(best_raw, this_raw, nan_pol
icy='omit')
                p_val_test[id_alg][col] = p_stat
    return p_val_test

def plot_roc_pr(real_y, prob_y, algo_name, data_name):
    fig, axes = plt.subplots(1,2, figsize=(12,10))
    # for roc_auc_curve
    # roc_score = roc_auc_score(real_y, prob_y)

    fpr, tpr, _ = roc_curve(real_y, prob_y)
    lab1 = 'ROC Area=%.2f' % (auc(fpr, tpr))
    axes[0].plot([1, 0], [1, 0], color='red', lw=2, linestyle='--', la
bel='Diagonal')
    axes[0].step(fpr, tpr, label=lab1, lw=2, color='black')
    axes[0].set_title((algo_name)+ ", Data: " + str(data_name))
    axes[0].set_xlabel('False Positive Rate')
    axes[0].set_ylabel('True Positive Rate')
    axes[0].legend(loc='lower left')
```

```
    # set same size for two subplots
    asp0 = np.diff(axes[0].get_xlim())[0] / np.diff(axes[0].get_ylim
())[0]
    axes[0].set_aspect(asp0)
    # for precision-recall
    precision, recall, _ = precision_recall_curve(real_y, prob_y)
    ave_PR = average_precision_score(real_y, prob_y)
    lab2 = 'PR AUC=%.2f' % (ave_PR)
    # add diagonal line
    axes[1].plot([0.0, 1.0], [1.0, 0.0], color='red', lw=2, linestyle=
'--', label='Diagonal')
    axes[1].step(recall, precision, label=lab2, lw=2, color='black')
    axes[1].set_title((algo_name)+ ", Data: " + str(data_name))
    axes[1].set_xlabel('Recall')
    axes[1].set_ylabel('Precision')
    axes[1].legend(loc='lower left')
    asp1 = np.diff(axes[1].get_xlim())[0] / np.diff(axes[1].get_ylim
())[0]
    axes[1].set_aspect(asp1)
    plt.show()
    fig.tight_layout()
```

Load cleaned data:

In [ ]:

```
ADULT = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/data/clean
ed/adult.csv').drop('Unnamed: 0', axis=1)
LETTER1 = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/data/cle
aned/letter_p1.csv').drop('Unnamed: 0', axis=1)
LETTER2 = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/data/cle
aned/letter_p2.csv').drop('Unnamed: 0', axis=1)
COVTYPE = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/data/cle
aned/covtype5.csv').drop('Unnamed: 0', axis=1)
```

Algorithms and their parameters:

In [ ]:
```
LOGIT = {'name' : 'LogisticRegression()',
            'name_str' : 'LogisticRegression',
            'module' : 'sklearn.linear_model',
            'hyperparameters' : {
                 'solver': ['liblinear'],
                 'C' : [10**i for i in range(-8, 4)],
                 }
            }

TREES = {'name' : 'DecisionTreeClassifier()',
          'name_str' : 'DecisionTreeClassifier',
          'module' : 'sklearn.tree',
          'hyperparameters' : {
              'criterion': ['gini', 'entropy'],
              'max_depth': [i for i in range(1, 26, 3)],
              'min_samples_split': [i for i in range(1, 10, 2)],
              'min_samples_leaf': [1,3,5,7],
              }
         }

FOREST = {'name' : 'RandomForestClassifier()',
          'name_str' : 'RandomForestClassifier',
          'module' : 'sklearn.ensemble',
          'hyperparameters' : {
              'n_estimators' : [2**i for i in range(1, 11)],
              'max_features' : [1,2,4,6,8,12,16,20],
              }
          }

GBOOST = {'name' : 'GradientBoostingClassifier()',
             'name_str' : 'GradientBoostingClassifier',
             'module' : 'sklearn.ensemble',
             'hyperparameters' : {
                 'n_estimators' : [2,4,8,16,32,64,128,256,512,1024,204
8],
                 'max_depth': [i for i in range(1, 10, 2)],
                 }
              }

MLP_ADAM = {'name' : 'MLPClassifier()',
       'name_str' : 'MLPClassifier',
       'module' : 'sklearn.neural_network',
       'hyperparameters' : {
           'hidden_layer_sizes' : [1,2,4,8,32,128,256],
           }
            }

NB = {'name' : 'BernoulliNB()',
             'name_str' : 'BernoulliNB',
             'module' : 'sklearn.naive_bayes',
             'hyperparameters' : {
                 'alpha' : [10**i for i in range(-4, 4)]
                 }}
```

Function and variables for training classifiers

In [ ]:
```
TRIALS = 5
CV_NUM = 5

# add all datasets/algos to a list
data_list = [ADULT, LETTER1, LETTER2, COVTYPE]
algorithms = [LOGIT, TREES, FOREST, GBOOST, MLP_ADAM, NB]

# string names of variables used for printing
dataset_names =  ['ADULT', 'LETTER1', 'LETTER2', 'COVTYPE']
algo_names = ['LOGIT', 'TREES', 'FOREST', 'GBOOST', 'MLP_ADAM', 'NB']
metric_names = ['ACC','APR','ROC']
scorings = {'ACC':'accuracy', 'APR': 'average_precision','ROC' : 'roc_
auc'}

# metrics used
scorers = [accuracy_score, average_precision_score, roc_auc_score]

# scores of each algorithm by data set (averaged over all metrics)
algo_data_df = pd.DataFrame(0.0, index=algo_names, columns=dataset_nam
es)

# scores for each algorithm by metric (average over all data sets)
algo_metric_df_train = pd.DataFrame(0.0, index=algo_names, columns=met
ric_names)
algo_metric_df_test = pd.DataFrame(0.0, index=algo_names, columns=metr
ic_names)

# for each algo, store raw train/test scores for each dataset
# number_of_dataset by trails by number_of_metrics
raw_train = np.zeros((len(algorithms), len(dataset_names), TRIALS, len
(metric_names)))
raw_test = np.zeros((len(algorithms), len(dataset_names), TRIALS, len
(metric_names)))


# work on one algorithm
def learn(idx_algo, algo):
    start = time.time()
    print('\n
-------------------------------------------------------------\n')
    print(f'Started training {algo["name"]}')

    # for this algo, store training metrics averaged across all datase
ts after the loop over data_list
    metric_across_data_train = np.zeros((len(dataset_names), len(metri
c_names)))

    # for this algo, store testing metrics averaged across all dataset
s after the loop over data_list
    metric_across_data_test = np.zeros((len(dataset_names), len(metric
_names)))

    # for this algo, store raw training metrics
    # for each dataset, there is 3 metrics. For each metric, there is
5 trials
```

```
    raw_metric_data_train = np.zeros((len(dataset_names), TRIALS, len
(metric_names)))

    # for this algo, store raw training metrics
    raw_metric_data_test = np.zeros((len(dataset_names), TRIALS, len(m
etric_names)))

    # loop over all datasets
    for idx_data, data in enumerate(progress_bar(data_list)):
        start_data = time.time()
        print(f'Started on {dataset_names[idx_data]} dataset')

        # for each algo/data combo, store testing metrics averaged acr
oss 5 trials
        metric_across_trials_test = np.zeros((TRIALS, len(metric_name
s)))
        # for each algo/data combo, store training metrics averaged ac
ross 5 trials
        metric_across_trials_train = np.zeros((TRIALS, len(metric_name
s)))

        # data for precision-recall curve
        auc_real_y = []
        auc_prob_y = []

        # loop over all trials
        for trial in progress_bar(range(TRIALS)):
            start_trial = time.time()
            print("Started trial: ", trial+1)

            # pick 5000 samples for training
            X = data.drop('y', axis=1)
            Y = data['y']
            X_train, X_test, Y_train, Y_test = train_test_split(X, Y,
train_size=5000, random_state=trial)

            # only scale data for MLP's or LOGREG
            if (algo['name_str'] in ['MLPClassifier', 'LogisticRegress
ion']):
                scaler = StandardScaler()
                X_train = pd.DataFrame(scaler.fit_transform(X_train),
columns = X.columns)
                X_test = pd.DataFrame(scaler.transform(X_test), column
s = X.columns)

            clf = eval(algo['name'])
            param = algo['hyperparameters']

            # Get each parameter that has best performance on validati
on set

            CV = GridSearchCV(clf, param, cv=CV_NUM, n_jobs=-1, scorin
g=scorings, refit='ACC')
            CV.fit(X_train, Y_train)
            result_cv = CV.cv_results_
```

```
            print("\nBest param for ACC:", CV.best_params_)

            # get parameters for best models for each metric
            param_list = []
            for metric in metric_names:
                best_id = pd.Series(result_cv['rank_test_'+str(metri
c)]).idxmin()
                param_list.append(result_cv['params'][best_id])

            # Train n models using the 5000 samples and each of the n
best parameters
            # and test on test set
            clf_name = algo['name_str']
            module = algo['module']
            train_metrics = []
            test_metrics = []

            for i in range(len(param_list)):
                clf_best = get_model(clf_name, module, param_list[i])
                clf_best.fit(X_train,Y_train)

                # for roc_auc and PR curves
                pred_proba = clf_best.predict_proba(X_test)
                auc_real_y.append(Y_test)
                auc_prob_y.append(pred_proba[:,1])

                X_train_pred = clf_best.predict(X_train)
                X_test_pred = clf_best.predict(X_test)
                X_train_pred_prob = clf_best.predict_proba(X_train)
                X_test_pred_prob = clf_best.predict_proba(X_test)

                # get appropriate scoring function
                scorer = scorers[i]

                # if metric is average precision, need y-score or prob
a
                if (i==1):
                    train_metrics.append(scorer(Y_train, X_train_pred_
prob[:, 1]))
                    test_metrics.append(scorer(Y_test, X_test_pred_pro
b[:, 1]))
                # if scorer is roc_auc, need proba
                elif (i==2):
                    train_metrics.append(scorer(Y_train, X_train_pred_
prob[:, 1]))
                    test_metrics.append(scorer(Y_test, X_test_pred_pro
b[:, 1]))
                else:
                    train_metrics.append(scorer(Y_train, X_train_pre
d))
                    test_metrics.append(scorer(Y_test, X_test_pred))


            # update the row in metric_across_trials_train
            metric_across_trials_train[trial] = train_metrics
```

```
            # update the row in metric_across_trials
            metric_across_trials_test[trial] = test_metrics

            finish_trial = time.time()
            print(f'Ended trial {trial+1} in {(finish_trial - start_tr
ial):.3f} seconds')
            print('\n')

        # plot ROC and PR curves
        auc_real_y = np.concatenate(auc_real_y)
        auc_prob_y = np.concatenate(auc_prob_y)
        plot_roc_pr(auc_real_y, auc_prob_y, algo['name_str'], dataset_
names[idx_data])

        # add 5 trails by 3 metrics data to raw list
        raw_metric_data_train[idx_data] = metric_across_trials_train
        raw_metric_data_test[idx_data] = metric_across_trials_test

        # mean of metrics across trials
        mean_across_trials_train = np.mean(metric_across_trials_train,
axis=0)
        mean_across_trials_test = np.mean(metric_across_trials_test, a
xis=0)

        # update algo-data combo with mean of mean_across_trials
        mean_algo_data = np.mean(mean_across_trials_test)
        algo_data_df.iat[idx_algo, idx_data] = mean_algo_data

        # update metric_across_data
        metric_across_data_train[idx_data] = mean_across_trials_train
        metric_across_data_test[idx_data] = mean_across_trials_test

        finish_data = time.time()
        print(f'Ended {dataset_names[idx_data]} in {(finish_data - sta
rt_data):.3f} seconds')

    # update raw_train and raw_test
    raw_train[idx_algo] = raw_metric_data_train
    raw_test[idx_algo] = raw_metric_data_test

    # mean of metrics across data
    mean_across_data_train = np.mean(metric_across_data_train, axis=0)
    mean_across_data_test = np.mean(metric_across_data_test, axis=0)
    algo_metric_df_train.iloc[idx_algo] = mean_across_data_train
    algo_metric_df_test.iloc[idx_algo] = mean_across_data_test

    finish   = time.time()
    print(f'Ended {algo["name"]} in {(finish - start):.3f} seconds')
    print('\n
-------------------------------------------------------------\n')
```

Training Per Algorithm

Logit

In [ ]:
```
# algorithms = [LOGIT, TREES, FOREST, GBOOST, MLP_ADAM, NB]
learn(0, algorithms[0])
```

Decision Trees

In [ ]:
```
# algorithms = [LOGIT, TREES, FOREST, GBOOST, MLP_ADAM, NB]
learn(1, algorithms[1])
```

Random Forest

In [ ]:
```
# algorithms = [LOGIT, TREES, FOREST, GBOOST, MLP_ADAM, NB]
learn(2, algorithms[2])
```

Gradient Boosting Classifier

In [ ]:
```
# algorithms = [LOGIT, TREES, FOREST, GBOOST, MLP_ADAM, NB]
learn(3, algorithms[3])
```

MLP with ADAM

In [ ]:
```
# algorithms = [LOGIT, TREES, FOREST, GBOOST, MLP_ADAM, NB]
learn(4, algorithms[4])
```

Naive Bayes

In [ ]:
```
# algorithms = [LOGIT, TREES, FOREST, GBOOST, MLP_ADAM, NB]
learn(5, algorithms[5])
```

Tables and Output

In [ ]:

```
p_val_test_metric = p_stats(raw_test, algo_metric_df_test, len(algo_na
mes), len(metric_names), per_data=False)
print('p_val_test_metric: for Table 2')
print(p_val_test_metric);print()

p_val_train_metric =  p_stats(raw_train, algo_metric_df_train, len(alg
o_names), len(metric_names), per_data=False)
print('p_val_train_metric:')
print(p_val_train_metric);print()

p_val_test_data = p_stats(raw_test, algo_data_df, len(algo_names), len
(data_list), per_data=True)
print('p_val_test_data: for Table 3')
print(p_val_test_data);print()
```