Nicholas Schenone - A13599911 - COGS 118A Final Project

1 - Abstract

In this paper, I performed an empirical evaluation of supervised learning on relatively high dimensional data. I evaluated performance on the accuracy and the F1 score on the test set. On lower dimensional data, many of the models performed similarly. But as the datasets became larger (both in terms of size and dimensionality), certain models performed better than others. My findings were not consistent with those of the paper in that the top performing models were Logarithmic Regression and Multi-Layer Perceptron. In fact, the Random Forest was one of the lower performing models that I tested.

2 - Introduction

In the last several years, machine learning (and deep learning in particular) have become a hot button item. From image analysis, to natural language processing, to medical imaging technologies, machine learning has certainly left its mark on the world - and it's only just beginning. Its popularity is only rivaled by its depth and complexity. As a newcomer to the field, how does one get started? This paper seeks to assess the performance of 7 difference machine learning model types across 3 different binary classification datasets. This will give the reader a general idea of how powerful different models are depending on the context and data being used to train.

3 - Methods

3.1 - Learning algorithms

This section summarizes the 7 different learning algorithms and the list of hyperparameters that were tuned, tested, and evaluated in this study. All of the following models were implemented using Scikit-Learn, a machine learning framework for Python. Due to the large number of hyperparameters, all tuning was done using a random grid search using the same Scikit-Learn library.

Support Vector Machine (SVM): I varied the kernel {linear, rbf}, C {1, 10, 100, 1000, 10000}, and gamma {1e-6, 1e-5, 1e-4, 1e-3, 1e-2} parameters while trying to find the best model.

Logarithmic Regression (LR): I varied the C {1, 10, 100, 1000, 10000} and penalty {l1, l2} parameters while trying to find the best model.

Decision Tree (DT): I varied the criterion {gini, entropy} and max depth of the tree {4,6,8,12} parameters while trying to find the best model.

Perceptron (PRC): This was specifically a single layer perceptron and thus a linear model, as Multi-Layer Perceptrons were tested as well. I varied the penalty {None, l1, l2, elastic net}, alpha {0.001, 0.0001, 0.00001}, maximum iteration {500, 1000, 2000}, tolerance {1e-4, 1e-3, 1e-2}, and early stopping {True, False} parameters while trying to find the best model.

Multi-Layer Perceptron (ANN): Unlike the single layer Perceptron, the ANN had multiple layers stacked on top of each other. I varied the hidden layer sizes {(100,), (50,), (200,), (25,)}, activation function {identity, logistic, tanh, relu}, solver {lbfgs, sgd, adam}, maximum iteration {200, 100, 300}, tolerance {1e-4, 1e-3, 1e-5}, and early stopping {True, False} while trying to find the best model.

K-Nearest Neighbor (KNN): I used a uniform weighted KNN. I varied the size of n {1, 3, 5, 9, 15, 25, 50, 75, 100} while trying to find the best model.

Random Forest (RF): Unlike all the previous models, this is an ensemble model. This means that the prediction is made using a combination of predictions from individual Decision Tree models. I varied bootstrapping {True, False}, the max depth of the trees {10, 20, 30, 40, 50, 60, 70, 80, 90, 100, None}, the minimum number of samples required to be at a leaf node {1, 2, 4}, the minimum number of samples required to split an internal node {2, 5, 10}, and the number of trees in the forest {200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 2000} while trying to find the best model.

3.2 - Performance metric:
To evaluate the performance of a model, I used two criteria: accuracy score and F1 score. These two scores were compared and ranked on the testing set.

3.3 - Datasets (and description of problem):
We compare the methods on 3 binary classification problems whose dimensionality ranges from 303 to 48842. The datasets are summarized in the following table:

Problem	# Attr	Train size (80/20)	Test size (80/20)	Train size (20/80)	Test size (20/80)	% Poz
Heart	13	242	61	60	243	54.5%
Mushroom	22/118	6499	1625	1624	6500	51.8%
Adult	13/111	39073	9769	9768	39074	7.9%

The Heart Disease dataset is from Kaggle. It uses categories such as age, sex, resting blood pressure, maximim heart rate achieved, and more to predict whether a particular subject has heart diesase or not. The dataset is mostly comprised of older individuals as the median age is 63. Additionally, it is mainly male. However, the dataset itself is fairly balanced in terms of the target.

The Mushroom dataset is from the UCI Machine Learning repository. It includes descriptions of hypothetical samples corresponding to 23 species of gilled mushrooms in the Agaricus and Lepiota Family. Each species is identified as definitely edible, definitely poisonous, or of unknown edibility and not recommended. This latter class was combined with the poisonous one, so the two categories are edible and not edible. The dataset itself is also fairly balanced in terms of the target.

The Adult dataset (also known as the Census Income dataset) is also from the UCI Machine Learning repository. The dataset predicts whether a person's income exceeds $50K/yr based on census data. Predictors include age, working class, education level, marital status, and more. This is the largest dataset in this test, but is not very balanced in terms of the target.

4 - Experiments

Because of the size of the data (especially the adult dataset), the sizable number of models, and large number of hyperparameters, computational efficiency was a concern. I did not want to grid search for hyperparameters for every trial of every model of every dataset of every data split. To do so would have made for an incredibly long training process. To put it into perspective, one hyperparameter tuning of the adult dataset on the random forest took over 45 minutes. However, most of the tunings took between 0 and 5 minutes.

For the actual experimentation, I broke it up into two parts: hyperparameter tuning and the main training loop. This way, I could solve for the near-optimal hyperparameters in one step and have them saved on disk, and then load them from the disk while training. Due to the large number of hyperparameters, I used a random grid search with a large candidate pool to get near-optimal results. The near-optimal hyperparameters are displayed per model per dataset in the tables below:

Model	Trial 1 - Heart	Trial 2 - Heart	Trial 3 - Heart
SVM	{"kernel": "rbf", "gamma": 0.01, "C":	{"kernel": "rbf", "gamma": 1e-05, "C":	{"kernel": "linear", "gamma": 0.001, "C":

	10000}	1000}	1}
LR	{"penalty": "l1", "C": 1}	{"penalty": "l2", "C": 1}	{"penalty": "l1", "C": 10}
DT	{"max_depth": 4, "criterion": "gini"}	{"max_depth": 6, "criterion": "gini"}	{"max_depth": 8, "criterion": "gini"}
PRC	{"tol": 0.01, "penalty": "l1", "max_iter": 1000, "early_stopping": true, "alpha": 0.001}	{"tol": 0.01, "penalty": "elasticnet", "max_iter": 2000, "early_stopping": false, "alpha": 1e-05}	{"tol": 0.001, "penalty": null, "max_iter": 2000, "early_stopping": true, "alpha": 0.0001}
ANN	{"tol": 1e-05, "solver": "adam", "max_iter": 200, "hidden_layer_sizes": [25], "early_stopping": false, "activation": "logistic"}	{"tol": 1e-05, "solver": "sgd", "max_iter": 200, "hidden_layer_sizes": [50], "early_stopping": false, "activation": "tanh"}	{"tol": 0.0001, "solver": "adam", "max_iter": 200, "hidden_layer_sizes": [50], "early_stopping": false, "activation": "tanh"}
KNN	{"n_neighbors": 1}	{"n_neighbors": 3}	{"n_neighbors": 5}
RF	{"n_estimators": 600, "min_samples_split": 2, "min_samples_leaf": 4, "max_features": "sqrt", "max_depth": null, "bootstrap": false}	{"n_estimators": 1400, "min_samples_split": 10, "min_samples_leaf": 4, "max_features": "sqrt", "max_depth": 20, "bootstrap": true}	{"n_estimators": 1400, "min_samples_split": 2, "min_samples_leaf": 2, "max_features": "auto", "max_depth": 70, "bootstrap": false}

Model	Trial 1 - Mushroom	Trial 2 - Mushroom	Trial 3 - Mushroom
SVM	{"kernel": "rbf", "gamma": 1e-06, "C": 10}	{"kernel": "linear", "gamma": 1e-06, "C": 10}	{"kernel": "rbf", "gamma": 0.01, "C": 10000}
LR	{"penalty": "l1", "C": 1}	{"penalty": "l2", "C": 1}	{"penalty": "l1", "C": 10}

DT	{"max_depth": 4, "criterion": "gini"}	{"max_depth": 6, "criterion": "gini"}	{"max_depth": 8, "criterion": "gini"}
PRC	{"tol": 0.0001, "penalty": "l1", "max_iter": 1000, "early_stopping": false, "alpha": 1e-05}	{"tol": 0.0001, "penalty": null, "max_iter": 1000, "early_stopping": true, "alpha": 1e-05}	{"tol": 0.001, "penalty": "l1", "max_iter": 500, "early_stopping": true, "alpha": 1e-05}
ANN	{"tol": 1e-05, "solver": "lbfgs", "max_iter": 300, "hidden_layer_sizes": [200], "early_stopping": false, "activation": "identity"}	{"tol": 0.001, "solver": "adam", "max_iter": 200, "hidden_layer_sizes": [50], "early_stopping": false, "activation": "logistic"}	{"tol": 0.0001, "solver": "sgd", "max_iter": 200, "hidden_layer_sizes": [50], "early_stopping": false, "activation": "logistic"}
KNN	{"n_neighbors": 1}	{"n_neighbors": 3}	{"n_neighbors": 5}
RF	{"n_estimators": 1200, "min_samples_split": 10, "min_samples_leaf": 4, "max_features": "sqrt", "max_depth": 30, "bootstrap": true}	{"n_estimators": 600, "min_samples_split": 10, "min_samples_leaf": 4, "max_features": "sqrt", "max_depth": null, "bootstrap": false}	{"n_estimators": 1600, "min_samples_split": 2, "min_samples_leaf": 1, "max_features": "sqrt", "max_depth": 90, "bootstrap": true}

Model	Trial 1 - Adult	Trial 2 - Adult	Trial 3 - Adult
SVM	{"kernel": "rbf", "gamma": 1e-06, "C": 1000}	{"kernel": "linear", "gamma": 0.0001, "C": 1}	{"kernel": "rbf", "gamma": 0.001, "C": 100}
LR	{"penalty": "l1", "C": 1}	{"penalty": "l2", "C": 1}	{"penalty": "l1", "C": 10}
DT	{"max_depth": 4, "criterion": "gini"}	{"max_depth": 6, "criterion": "gini"}	{"max_depth": 8, "criterion": "gini"}
PRC	{"tol": 0.001, "penalty": "l2", "max_iter": 500, "early_stopping":	{"tol": 0.001, "penalty": "l1", "max_iter": 500, "early_stopping":	{"tol": 0.0001, "penalty": null, "max_iter": 500, "early_stopping":

	true, "alpha": 1e-05}	false, "alpha": 1e-05}	true, "alpha": 0.0001}
ANN	{"tol": 1e-05, "solver": "sgd", "max_iter": 200, "hidden_layer_sizes": [50], "early_stopping": false, "activation": "tanh"}	{"tol": 0.0001, "solver": "sgd", "max_iter": 100, "hidden_layer_sizes": [200], "early_stopping": true, "activation": "identity"}	{"tol": 0.001, "solver": "adam", "max_iter": 300, "hidden_layer_sizes": [200], "early_stopping": true, "activation": "identity"}
KNN	{"n_neighbors": 1}	{"n_neighbors": 3}	{"n_neighbors": 5}
RF	{"n_estimators": 1000, "min_samples_split": 10, "min_samples_leaf": 1, "max_features": "auto", "max_depth": 10, "bootstrap": true}	{"n_estimators": 1600, "min_samples_split": 2, "min_samples_leaf": 2, "max_features": "auto", "max_depth": 90, "bootstrap": true}	{"n_estimators": 200, "min_samples_split": 10, "min_samples_leaf": 2, "max_features": "sqrt", "max_depth": null, "bootstrap": false}

Once the hyperparameters were obtained and saved on disk, the main training loop looking something like the following pseudo code:

```
For each dataset (Heart, Mushroom, Adult):
    For each data split (80/20, 20/80):
        Prepare data splits (X_train, y_train, etc.)
            For each trial i in range(1 - 3):
                For each model (SVM, LR, DT, PRC, ANN, KNN, RF):
                    Load ith set of best hyperparameters
                    Train model
                    Evaluate model (accuracy / f1 score) on training set
                    Evaluate model (accuracy / f1 score) on testing set
                    Save model evaluation in large json file
```

5 - Conclusions

5.1 - Model Evaluation Results

Following the training of the models, I evaluated them on two metrics: testing accuracy and testing F1 score. The results are as follows, separated by the data split (80/20, 20/80):

5.1.1 - Model Evaluation (80/20 split)

80/20 Accuracy	Adult Dataset (80/20)		Mushroom Dataset (80/20)		Heart Dataset (80/20)		Average (80/20)	
	Train Acc	Test Acc	Train Acc	Test Acc	Train Acc	Test Acc	Train Acc	Test Acc
Multi-Layer Perceptron	99.99 ± 0.0%	99.96 ± 0.03%	99.98 ± 0.02%	100.0 ± 0.0%	89.12 ± 0.39%	71.04 ± 0.77%	96.37%	90.33%
Logarithmic Regression	100.0 ± 0.0%	100.0 ± 0.0%	100.0 ± 0.0%	100.0 ± 0.0%	88.84 ± 0.0%	70.49 ± 0.0%	96.28%	90.16%
Decision Tree	100.0 ± 0.0%	100.0 ± 0.0%	100.0 ± 0.0%	100.0 ± 0.0%	96.42 ± 3.46%	69.95 ± 2.04%	98.81%	89.98%
Perceptron	99.81 ± 0.19%	99.8 ± 0.19%	100.0 ± 0.0%	100.0 ± 0.0%	82.64 ± 1.55%	69.95 ± 2.04%	94.15%	89.91%
Support Vector Machine	100.0 ± 0.0%	99.99 ± 0.02%	97.06 ± 4.16%	97.19 ± 3.97%	92.29 ± 5.46%	69.95 ± 0.77%	96.45%	89.04%
Random Forest	98.01 ± 2.75%	97.69 ± 2.56%	100.0 ± 0.0%	100.0 ± 0.0%	96.14 ± 2.81%	68.85 ± 1.34%	98.05%	88.85%
K-Nearest Neighbor	98.37 ± 1.19%	95.93 ± 0.18%	100.0 ± 0.0%	100.0 ± 0.0%	92.98 ± 4.97%	70.49 ± 1.34%	97.12%	88.81%

80/20 F1	Adult Dataset (80/20)		Mushroom Dataset (80/20)		Heart Dataset (80/20)		Average (80/20)	
	Train F1	Test F1	Train F1	Test F1	Train F1	Test F1	Train F1	Test F1
Multi-Layer Perceptron	99.98 ± 0.01%	99.87 ± 0.11%	99.98 ± 0.02%	100.0 ± 0.0%	88.89 ± 0.39%	69.09 ± 0.7%	96.29%	89.65%
Decision Tree	100.0 ±	100.0 ±	100.0 ±	100.0 ±	96.33 ±	68.09 ±	98.78%	89.36%

	0.0%	0.0%	0.0%	0.0%	3.56%	2.15%		
Logarithmic Regression	100.0 ± 0.0%	100.0 ± 0.0%	100.0 ± 0.0%	100.0 ± 0.0%	88.6 ± 0.0%	68.01 ± 0.0%	96.2%	89.34%
Perceptron	99.35 ± 0.65%	99.31 ± 0.66%	100.0 ± 0.0%	100.0 ± 0.0%	82.35 ± 1.69%	68.35 ± 2.56%	93.9%	89.22%
Support Vector Machine	100.0 ± 0.0%	99.95 ± 0.07%	97.02 ± 4.22%	97.14 ± 4.04%	92.09 ± 5.61%	68.02 ± 1.2%	96.37%	88.37%
K-Nearest Neighbor	93.82 ± 4.58%	84.89 ± 0.95%	100.0 ± 0.0%	100.0 ± 0.0%	92.88 ± 5.04%	69.23 ± 1.4%	95.57%	84.71%
Random Forest	89.49 ± 14.67%	88.08 ± 14.29%	100.0 ± 0.0%	100.0 ± 0.0%	96.08 ± 2.85%	65.86 ± 1.76%	95.19%	84.65%

5.1.2 - Model Evaluation (20/80 split)

20/80 Accuracy	Adult Dataset (20/80)		Mushroom Dataset (20/80)		Heart Dataset (20/80)		Average (20/80)	
	Train Acc	Test Acc	Train Acc	Test Acc	Train Acc	Test Acc	Train Acc	Test Acc
Logarithmic Regression	100.0 ± 0.0%	100.0 ± 0.0%	100.0 ± 0.0%	99.97 ± 0.04%	100.0 ± 0.0%	82.03 ± 0.19%	100.0%	94.0%
Multi-Layer Perceptron	99.98 ± 0.01%	99.87 ± 0.09%	99.92 ± 0.12%	99.78 ± 0.21%	96.67 ± 2.72%	80.93 ± 1.18%	98.86%	93.53%
Perceptron	99.85 ± 0.12%	99.74 ± 0.09%	100.0 ± 0.0%	99.93 ± 0.03%	98.89 ± 0.79%	78.6 ± 2.67%	99.58%	92.76%
Random Forest	97.87 ± 2.64%	97.12 ± 2.48%	100.0 ± 0.0%	100.0 ± 0.0%	98.33 ± 2.36%	78.05 ± 1.27%	98.73%	91.72%
Decision Tree	100.0 ± 0.0%	100.0 ± 0.0%	100.0 ± 0.0%	100.0 ± 0.0%	99.44 ± 0.79%	73.66 ± 1.46%	99.81%	91.22%
K-Nearest Neighbor	97.51 ± 1.83%	94.16 ± 0.15%	99.96 ± 0.03%	99.9 ± 0.01%	96.11 ± 2.83%	78.05 ± 0.19%	97.86%	90.7%
Support Vector Machine	100.0 ± 0.0%	99.97 ± 0.05%	83.62 ± 23.16%	83.93 ± 22.56%	97.22 ± 3.93%	80.38 ± 1.03%	93.61%	88.09%

20/80 F1	Adult Dataset (20/80)		Mushroom Dataset (20/80)		Heart Dataset (20/80)		Average (20/80)	
	Train F1	Test F1	Train F1	Test F1	Train F1	Test F1	Train F1	Test F1
Logarithmic Regression	100.0 ± 0.0%	100.0 ± 0.0%	100.0 ± 0.0%	99.97 ± 0.04%	100.0 ± 0.0%	81.87 ± 0.18%	100.0%	93.95%
Multi-Layer Perceptron	99.94 ± 0.03%	99.56 ± 0.32%	99.92 ± 0.12%	99.78 ± 0.22%	96.36 ± 2.99%	80.49 ± 1.33%	98.74%	93.27%
Perceptron	99.48 ± 0.4%	99.11 ± 0.32%	100.0 ± 0.0%	99.93 ± 0.03%	98.83 ± 0.82%	78.52 ± 2.66%	99.44%	92.52%
Decision Tree	100.0 ± 0.0%	100.0 ± 0.0%	100.0 ± 0.0%	100.0 ± 0.0%	99.41 ± 0.84%	73.4 ± 1.47%	99.8%	91.13%
Random Forest	88.99 ± 14.26%	85.31 ± 14.9%	100.0 ± 0.0%	100.0 ± 0.0%	98.22 ± 2.51%	77.41 ± 1.19%	95.74%	87.57%
Support Vector Machine	100.0 ± 0.0%	99.88 ± 0.16%	77.9 ± 31.25%	78.0 ± 30.95%	96.91 ± 4.37%	79.97 ± 1.37%	91.61%	85.95%
K-Nearest Neighbor	89.98 ± 7.46%	77.99 ± 1.54%	99.96 ± 0.03%	99.9 ± 0.01%	95.75 ± 3.11%	77.28 ± 0.35%	95.23%	85.06%

5.2 - Rank Classifiers via Accuracy

Based on the metric of average testing accuracy across data splits, Logarithmic Regression and the Multi-Layer Perceptron were the best performing models. The results are displayed in the following table:

	Avg Test Acc 80/20	Avg Test Acc 20/80	Avg Test Acc
Logarithmic Regression	90.16%	94.0%	92.080%
Multi-Layer Perceptron	90.33%	93.53%	91.930%

Perceptron	89.91%	92.76%	91.335%
Decision Tree	89.98%	91.22%	90.600%
Random Forest	88.85%	91.72%	90.285%
K-Nearest Neighbor	88.81%	90.7%	89.755%
Support Vector Machine	89.04%	88.09%	88.565%

5.3 - Rank classifiers via F1 score

Based on the metric of average testing F1 score across data splits, Logarithmic Regression and the Multi-Layer Perceptron were still the best performing models. The results are displayed in the following table:

	Avg Test F1 80/20	Avg Test F1 20/80	Avg Test F1
Logarithmic Regression	89.34%	93.95%	91.645%
Multi-Layer Perceptron	89.65%	93.27%	91.460%
Perceptron	89.22%	92.52%	90.870%
Decision Tree	89.36%	91.13%	90.245%
Support Vector Machine	88.37%	85.95%	87.160%
Random Forest	84.65%	87.57%	86.110%
K-Nearest Neighbor	84.71%	85.06%	84.885%

5.4 - Overall Results

By evaluating all of the classifiers on the metrics of testing accuracy and f1 score, I found that on these datasets, the top performing models are Logarithmic Regression, Multi-Layer Perceptron, Perceptron, and Decision Tree in that order. This did not agree with the reference paper in that

Random Forests were not the highest performing model. In fact, with my datasets, hyperparameters, and implementation, they were one of the lower performing models.

6 - Bonus Points

I evaluated 7 classifiers instead of 3.

7 - References

Detrano, R., Janosi, A., Steinbrunn, W., Pfisterer, M., Schmid, J., Sandhu, S., Guppy, K., Lee, S., & Froelicher, V. (1989). International application of a new probability algorithm for the diagnosis of coronary artery disease. American Journal of Cardiology, 64,304–310.

Koehrsen, W. (2018, January 10). Hyperparameter Tuning the Random Forest in Python. Retrieved March 17, 2020, from https://towardsdatascience.com/hyperparameter-tuning-the-random-forest-in-python-using-scikit-learn-28d2aa77dd74

KPLauritzen. (2017, December 13). Choosing top k models using GridSearchCV in scikit-learn. Retrieved from https://stackoverflow.com/questions/47793569/choosing-top-k-models-using-gridsearchcv-in-scikit-learn

Lichman, M. (2013). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

The Audubon Society Field Guide to North American Mushrooms (1981). G. H. Lincoff (Pres.), New York: Alfred A. Knopf