David Strube - dstrube3@gatech.edu

CS 7641 - Machine Learning

Assignment 1 - Supervised Learning

2020-09-19

For this assignment, I decided to use scikit-learn because it has many easy-to-use tools for Machine Learning in Python. (More of my career has been using Java, but I've recently found Python to be very intuitive and powerful.) In particular, I used scikit-learn version 0.23.2 and Python version 3.7.3. (The details of all libraries I used and their versions can be found in the corresponding README.txt.) I didn't use any fancy, latest-and-greatest features of Python or scikit-learn, so as long as you have some recent version of Python 3 and scikit-learn, you should be able to run all my code. I used OpenML for the source of my datasets because scikit-learn has an efficient and simple interface for loading datasets from there. The name of my Jupyter Notebook for this assignment is: “A1 - Supervised Learning - dstrube3.ipynb”

Description of classification problems, and why they are interesting

Dataset 1 - *phoneme*[1] - deals with speech recognition. It seeks to make a distinction between oral sounds and nasal sounds. Nasal sounds are classified as 0; oral sounds are classified as 1. (Dua, D. and Graff, C. (2019).) This caught my attention because I am fascinated by the prospect of speech recognition by computers. It's interesting because it is a moderately sized dataset (over 5,400 instances) with 5 attributes with just 2 targets. The target name is Class; the attribute names are as follows: V1, V2, V3, V4, & V5. (Further details of the nature of this

dataset can be found at the dataset's URL.) As described here (and in the original source description), this sounds like it should be a simple problem for most Machine Learning algorithms to determine some patterns; and indeed between the two datasets, this one was relatively well behaved and a good dataset to begin with learning on my own (even more so than a dataset used in a step-by-step guided tutorial).

Dataset 2 - *credit-g*[2] - deals with credit reporting, particularly by German credit reporting, and whether the individuals in the reports are good or bad credit risks as reported by UCI. (Lee, John A. (2000).) This caught my attention because I am curious to know what metrics are used in the mysterious inner workings of financial decisions like whether an individual is seen as risky or not. This dataset is interesting because, as a moderately sized dataset with 1000 instances and 20 attributes and again just 2 targets, my first thought was that this dataset would also be ideal to use as a beginner trying to utilize various ML algorithms with the goal of finding and showing some simple, clear, and obvious patterns. However, this dataset proved to be trickier than expected. Some of the algorithms (especially Support Vector Machines) required a great deal of tweaking and retesting to find the right combination of hyperparameters just so the confusion matrix and learning curve graph would display properly. The target name is Class. (Further details of the nature of this dataset can be found at the dataset's URL.) Note, the order of the targets (i.e., 'bad', 'good') makes the confusion matrix a little perplexing: this order puts a favorable 'bad' score in the true positive corner and a favorable 'good' score in the true negative corner.

1: https://www.openml.org/d/1489

2: https://www.openml.org/d/31

Training and testing error rates obtained from various learning algorithms

For both datasets, I wrote functions that would demonstrate the use of Machine Learning algorithms. These functions would take in at least the training portions of the data and targets of the datasets (*X_train* and *y_train* respectively). Most of these functions would also take in a suggested cross validation value for the *cv* parameter of the *cross_val_score* function. This suggested *cv* value would be determined at the outset of each dataset's training and testing by the first Decision Tree function returning not only the classifier but also whatever *cv* value had the highest score, as well as a minimum score (*min_score*) used to set bounds for the learning curve graphs. The *cv* value would usually be a good starting point in training and testing (because we certainly have to start somewhere); however this rarely turned out to be the seemingly optimal *cv* value.

After calling each function to demonstrate a Machine Learning algorithm, I called functions to graph a confusion matrix and learning curve graphs. Modifying the confusion matrix code from scikit-learn's website was pretty straightforward.[3] I passed the training and testing data to the confusion matrix function, as well as the classifier / estimator, and a parameter indicating what kind of normalization the confusion matrix should perform. Modifying the code from scikit-learn's website on how to plot a learning curve was a bit less straightforward.[4] I removed the Bayesian Learning part from the graphs, as well as some of the function's parameters that seemed unnecessary.

3: https://scikit-learn.org/stable/auto_examples/classification/plot_digits_classification.html

4: https://scikit-learn.org/stable/auto_examples/model_selection/plot_learning_curve.html

The first Decision Tree function (*decisionTree*) did no pruning. It looked for a good value for the *cv* parameter of the *cross_val_score* function and which criterion parameter to pass to future *DecisionTreeClassifier* constructors. In the first Decision Tree function, the *cross_val_score* function was called taking in a *DecisionTreeClassifier* as its first parameter. The other parameters of the *cross_val_score* function were *X_train, y_train,* and *cv*. The only parameters that were modified between runs were: *cv* & *criterion*. The range of *cv* was from 2 to 100, because this range yielded the most interesting range of scores returned by *cross_val_score*. The graph displayed by the first Decision Tree function was based on the mean cross validation scores as compared to the cross validation size.

I made separate functions for decision trees with pre-pruning (*decisionTreePrePruning*) and post-pruning (*decisionTreePostPruning*). For the pre-pruning function, only the parameter *max_depth* was modified between runs. The range of the *max_depth* parameter was from 2 to 20, because this range yielded the most interesting range of scores returned by the *DecisionTreeClassifier*. Similarly, for the post-pruning function, only the parameter *ccp_alpha* was modified between runs, ranging from 0 to 10.

The Neural Networks function (*neuralNetworks*) used an *MLPClassifier* (a Multi-layer Perceptron classifier), first by cycling through the 3 possible values for the *solver* parameter (*lbfgs, sgd,* & *adam*). For the *sgd* solver, I also cycled through the *learning_rates* (*constant, invscaling,* & *adaptive)*. Then I searched through orders of magnitude for the *alpha* parameter (from 1e5 to 1e-5), then explored values for the *hidden_layer_sizes* parameter (from *(1,1),* to *(10,10)*), as well as the available values of the *activation* parameter (*identity*, *logistic*, *tanh*, &

relu). Since the *hidden_layer_sizes* is a two-dimensional parameter, the graph at the end of *neuralNetworks* was a 3D scatter plot.

The Boosting function (*boosting*) used an *AdaBoostClassifier.* Only the parameter *n_estimators* was modified for each run, ranging from 1 to 100.

The Support Vector Machines function (*supportVectorMachines*) used all 3 types of SVM classifiers available in scikit-learn: *SVC*, *NuSVC*, & *LinearSVC*. Since the SVM type of *SVC* was found to be optimal for both datasets, this function then explored various parameters of this classifier's constructor: *kernel, shrinking, probability, decision_function_shape*, and (in cases where *kernel = rbf, poly*, or *sigmoid) gamma.* Note, the *kernel* value of *'precomputed'* was skipped because it always resulted in this error: "X should be a square kernel matrix". Also note, the parameter *break_ties* wasn't used because it doesn't apply - the number of classes was not greater than 2 for both datasets.

The K-Nearest Neighbors function (kNearestNeighbors) used a KNeighborsClassifier and looked over a range of *k* from 1 to 30.

Analyses

Dataset 1

Decision trees with and without some form of pruning

For Dataset 1, the function for Decision trees with no pruning (*decisionTree*) returned the highest scoring *cv* at 46 using *criterion = entropy* (~0.874). However, looking at the graph of cross validation scores compared to cross validation sizes, a *cv* value of 20 using *criterion = gini* should be sufficient with a score of ~0.870.

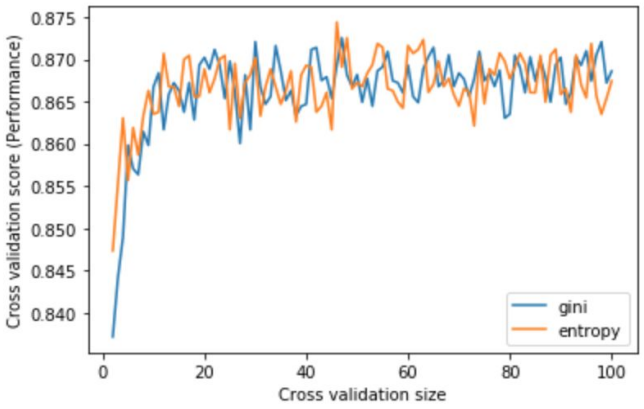

Figure1: Decision Tree with no pruning on Dataset 1

Using *cv* = 20 and *criterion* = *gini*, the function *decisionTreePrePruning* found its highest score of ~0.872 at *max_depth* = 20. However, looking at the graph of cross validation scores compared to cross validation sizes, an optimal *max_depth* value is reached at 8 with a score of ~0.852. Note, this is not as good as the afore-mentioned ~0.87 found in *decisionTree* with no pruning.

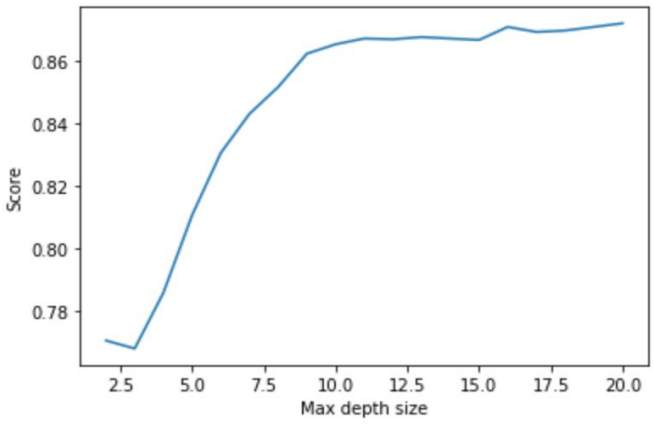

Figure 2: Decision Tree with pre-pruning on Dataset 1

The function *decisionTreePostPruning* found its highest score of ~0.87 using ccp_alpha = 0. This seems to be the same as using no post pruning at all, which appears to confirm the previous finding in the function *decisionTree*.

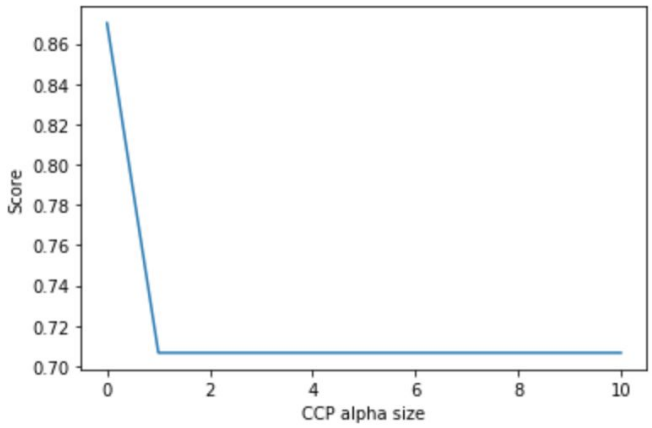

Figure 3: Decision Tree with post-pruning on Dataset 1

Neural Networks

The function *neuralNetworks* found solver *lbfgs* with a score of ~0.782. It then found an *alpha* order of magnitude = 10 with a score of ~0.823. It then looked for an optimal hidden layer setting and activation and found *hidden_layer_sizes* = (9,8) with *activation = tanh* with a score of ~0.855.

Boosting

The function *boosting* found a high scoring value for *n_estimators = 63* with a score of ~0.824. However, looking at the graph of scores compared to *n_estimators*, an *n_estimator* value of 21 should be sufficient with a score of ~0.814.

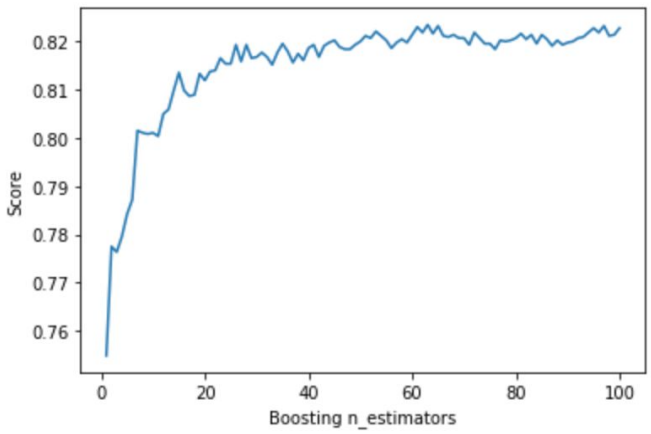

Figure 4: Boosting scores compared to n_estimators on Dataset 1

Support Vector Machines

The function *supportVectorMachines* found the best score using the SVM *SVC* (*kernel* = *rbf, shrinking*: True, *probability*: True, *decision_function_shape*: *ovo,* & *gamma: scale*) with a score of ~0.841.

K-Nearest Neighbors

The function *kNearestNeighbors* found the best score out of any Machine Learning algorithms for this dataset using $k = 1$ with a score of ~0.898. The graph of Score compared to K-Nearest Neighbors shows a downward trend as k increases.

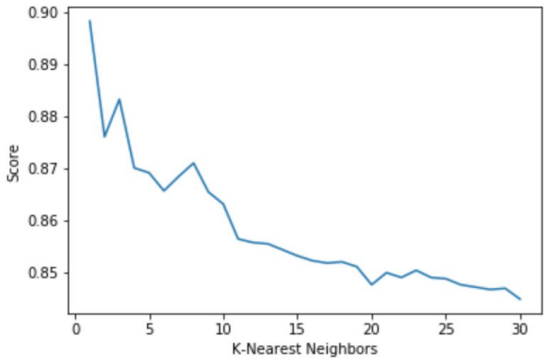

Figure 5: K-Nearest Neighbors on Dataset 1

Dataset 2

Decision trees with and without some form of pruning

For Dataset 2, the function for Decision trees with no pruning (*decisionTree*) started off with the range of *cv* being increased to [2, 201] because the default [2, 101] didn't resolve to any discernible pattern. In fact, increasing to a maximum *cv* value of 200 didn't yield much better results. Nonetheless, this function returned the highest scoring *cv* = 30 using *criterion* = *entropy* with a score of ~0.728. The function *decisionTreePrePruning* found its highest score - ~0.739 - at max_depth = 3. The function *decisionTreePostPruning* once again found its highest score - ~0.702 - at ccp_alpha = 0. So just like in Dataset 1, post-pruning didn't seem to help at all.

Neural Networks

The function *neuralNetworks* found solver *lbfgs* with a score of ~0.709. It then found an *alpha* order of magnitude: 100 with a score of ~0.709. It then looked for an optimal hidden layer setting and activation and found (7,6) with *activation* = *relu* with a score of ~0.713 in.

Boosting

The function *boosting* found its highest score of ~0.759 using n_estimators = 31.

Support Vector Machines

In the function *supportVectorMachines*, I added a parameter, *is_dataset_2*, to skip over exploring certain hyperparameters if the dataset was Dataset 2, because the function took too long in cases where *kernel = linear,* or when *kernel = poly* & *gamma = auto*. After this modification, the function *supportVectorMachines* found a best score using the SVM *SVC* (*kernel: rbf*, *shrinking*: True, *probability*: True, *decision_function_shape: ovo*, *gamma: scale*) with a score of ~0.708.

K-Nearest Neighbors

The function *kNearestNeighbors* found a highest score of ~0.701 with $k = 21$.

Conclusions

In some cases, the algorithms performed well with the models but in other cases it did not. In both cases, I did my best to find what hyperparameters could be tuned to find the best performance. In cases where this failed, it may be because I need to learn more about the algorithms, or the model may not be amenable to these algorithms.

For Dataset 1, the algorithm with the highest score was K-Nearest Neighbors. For this algorithm, the learning curve graph did not seem to converge as well as it did for other algorithms, like Neural Networks, Boosting, or Support Vector Machines. It is unclear whether

this because these other algorithms could have each had a better score, or because K-Nearest Neighbors learning curve could have been improved.

For Dataset 2, the algorithm with the highest score was Boosting. This algorithm's learning curve did seem to converge well, but then there were other algorithms whose score was not as good as Boosting but whose learning curve seemed to converge better; for example: Decision Tree (Post Pruning), Neural Networks, Support Vector Machines, and K-Nearest Neighbors. Again, tuning some of the hyperparameters differently in some of these algorithms may have given improved scores.

	Dataset 1	Dataset 2
Decision Tree (No Pruning)	0.8702	0.7284
Decision Tree (Pre Pruning)	0.8718	0.7386
Decision Tree (Post Pruning)	0.8702	0.7018
Neural Networks	0.8554	0.7125
Boosting	0.8137	**0.7586**
Support Vector Machines	0.8413	0.7075
K-Nearest Neighbors	**0.8982**	0.7013

Figure 6: Best scores in Datasets 1 & 2

In both datasets, the fastest algorithm was Decision Tree (Post Pruning) (Dataset 1: 5 seconds; Dataset 2: 1 second). The slowest was Neural Networks (Dataset 1: 1 hour, 22 minutes, and 3 seconds; Dataset 2: 15 minutes and 19 seconds).

References

Dua, D. and Graff, C. (2019). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml].

Irvine, CA: University of California, School of Information and Computer Science.

Lee, John A. (2000). The ELENA Project

[https://www.elen.ucl.ac.be/neural-nets/Research/Projects/ELENA/elena.htm].

Louvain-la-Neuve, Belgium: Université catholique de Louvain, Neural Network Group