Hyperparameter Tuning and Model Evaluation

Rafael A. Okiishi Paulucci
CS 4641 – Machine Learning – Spring 2020
Georgia Institute of Technology

***Abstract*—We analyze methods for hyperparameter tuning and model evaluation on the Breast Cancer Wisconsin (Diagnostic) dataset, using an SVM classifier, a multi-level perceptron, and the XGBoost algorithm.**

***Index Terms*—machine learning, classification, decision trees, boosting, support vector machines, cancer**

I. Introduction to the dataset

According to the World Health Organization, breast cancer is the most common cancer in women, causing over half a million deaths worldwide a year as of 2011. 58% of deaths from this ailment happen in less developed countries, where medical professionals tend to be overburdened or have scarce access to diagnosis equipment for breast conditions. In this context, research in automatically identifying breast cancer is important for women's (and even a few men's) health all over the globe.

Machine Learning has the potential to assist in the triage and diagnosis of patients. However, there is a wide array of classification techniques, most of which are only effective when they are "tuned" to a certain task and data. In our experiments, we demonstrate strategies to tune certain parameters, known as hyperparameters, in three algorithms: a non-linear SVM classifier (scikit-learn's SVC), a multi-level perceptron (scikit-learn's MLPClassifier), and a decision tree booster (XGBoost). We then compare the performance of the three tuned algorithms.

We use the Breast Cancer Wisconsin (Diagnostic) dataset, available at https://www.kaggle.com/uciml/breast-cancer-wisconsin-data. The data for this set originates from images of fine-needle aspirates of breast mass – samples of suspected cancer cells extracted from breasts via a needle. According to the Kaggle page mentioned, 10 real-valued aspects were originally measured:

- radius (mean of distances from center to points on the perimeter)
- texture (standard deviation of gray-scale values)
- perimeter
- area
- smoothness (local variation in radius lengths)
- compactness (perimeter^2 / area -1.0)
- concavity (severity of concave portions of the contour)
- concave points (number of concave portions of the contour)
- symmetry
- fractal dimension ("coastline approximation" -1)

For each image, the authors calculated the mean, standard error and mean of the three largest values, bringing the amount of features to 30.

This dataset is amenable to comparison experiments due to its popularity in the machine learning community, no missing values, and a relatively small size (357 benign and 252 malignant samples), which enables faster training. Despite having less than 1000 samples, the data are non-trivial due to feature engineering and the medical research applied to its ground truth.

Our analysis consists of the following steps, for each algorithm, with the objective being to classify images as belonging to the benign or malignant categories.

1) Scale and split data into 75% training, 25% test
2) Using the training data, perform an exhaustive hyperparameter search with a stratified K-fold cross-validation through a hand-picked grid of hyperparameters
3) Discard trials with score $>$ 1 std. error lower than the one resulting from the "best" parameters so far
4) Use the remaining parameter combinations to classify the test data, and report the highest score

II. Description of the algorithms

(For brevity, the theoretical aspects of each algorithm are not discussed at length. Please refer to each library's documentation for more information.)

A. *C-Support Vector Classification (SVC)*

scikit-learn's SVC is a Support Vector Machine classifier that can use linear, polynomial, RBF of sigmoid kernels. In this experiment, we do not employ the linear kernel for two reasons. Firstly, every parameter added to the exhaustive grid search cross-validation (GSCV) increases the required training time considerably. Secondly, most successful results from contests such as the one on Kaggle (https://www.kaggle.com/uciml/breast-cancer-wisconsin-data/kernels) have employed linear SVM kernels only with principal component analysis (PCA), which is not the aim of this experiment.

B. *Multi-level Perceptron (MLP)*

scikit-learn's MLP is an implementation of a neural network with an adjustable amount of hidden layers, as well as support for stochastic gradient descent (SGD) and Newton (LBFGS) optimization. It is also able to use momentum to avoid some local minima/maxima, and to change its learning rate dynamically.

C. *Extreme Gradient Boosting (XGBoost)*

XGBoost is a library that uses boosted trees for supervised learning. It supports an API similar to the one from scikit-learn's classifiers, but tends to run faster than that library's random forest and AdaBoost algorithms, in part due to its support for CUDA-compliant graphics card acceleration. In addition to its standard booster, XGBoost includes DART, which combines a large number of additive regression trees with random dropouts to reduce overfitting.

III. Tuning hyperparameters

The following experiments were run on a desktop computer with an AMD Ryzen 3600 6-core processor, 16 GB DDR4 3200 MHz RAM, Sapphire RX 5700 XT Nitro+ GPU and a Crucial MX500 SATA SSD.

A. *Attempt 1: nested cross-validation and random search*

Initially, we attempted to perform nested cross-validation (CV) with 14 runs of the "outer" folds (for consistency validation) and a 4-"inner" fold CV for hyperparameter tuning itself. To counteract the high running time, we employed the Optuna framework to suggest pseudo-random parameters from given ranges, as opposed to running an exhaustive search. We also maintained a low amount of "inner" folds.

Additionally, we had attempted to combine scikit-learn's RandomForestClassifier and AdaBoostClassifier, but this resulted in an extremely long training time. Therefore, we chose XGBoost over scikit-learn's ensemble methods.

Since a slightly (but significantly) lower training score might be indicative of a more parsimonious model, we plotted graphs (Fig. 1) of the validation score and standard error for 14 runs of each algorithm with the conditions above. Notice that, with the exception of very few trials, there was no significant difference in the training score with nested and non-nested CV, corroborating the argument that nested CV is "overzealous" for most datasets (Wainer and Cauley, 2018). Therefore, this approach was not pursued further.

B. *Attempt 2: 14-fold cross-validation and grid search*

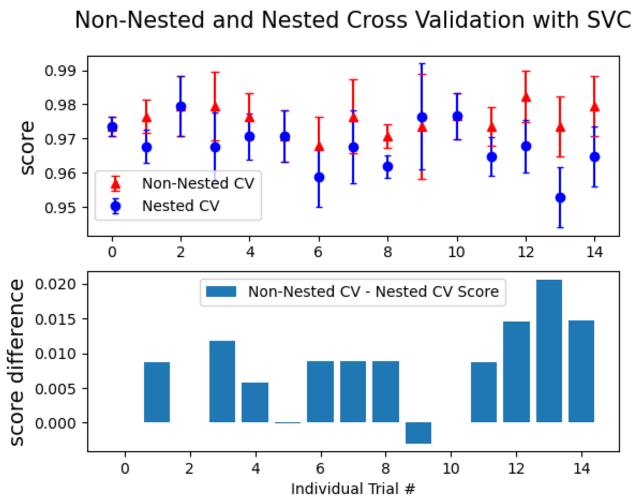
The procedure employed in the second attempt is outlined in section I. Here, we focus on the rationale for tuning some hyperparameters. Not every possible parameter was picked due to time constraints.

For SVC, we used a 1000 MB memory cache and tuned the following parameters:

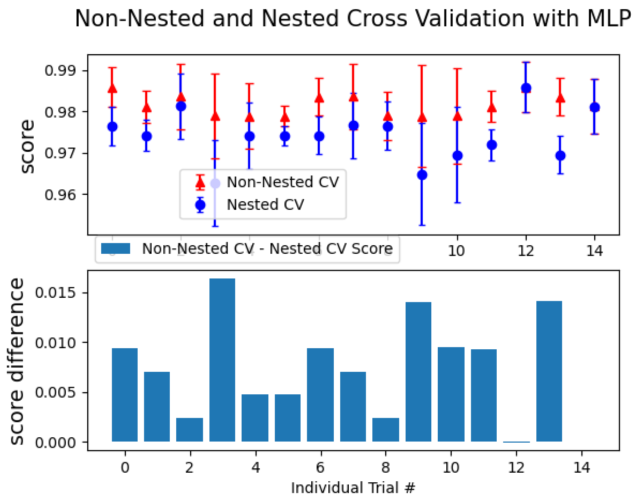
- C: inversely proportional to the strength of L2 regularization penalty
- Gamma (γ): kernel coefficient. Results in higher bias and lower variance when high
- Kernel: choice of polynomial, RBF or sigmoid kernel

For MLP, we used 5000 max. iterations (epochs) and tuned the following parameters:

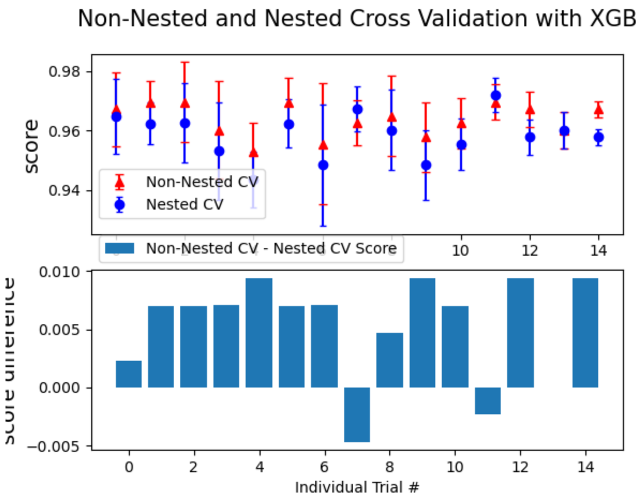
- Activation: the activation function for each node
- Solver: choice of LBFGS, SGD or Adam (SGD-based) solver



(a) Comparison of non-nested and nested CV on SVC



(b) Comparison of non-nested and nested CV on MLP



(c) Comparison of non-nested and nested CV on XGB

Figure 1: Comparison of nested and non-nested CV

- Momentum: helps prevent entrapment in local minima/maxima
- Learning rate (initial): causes convergence failure if too low, divergence if too high
- Learning rate (behavior): allows (or not) the solver to change the learning rate dynamically
- Alpha (α): strength of L2 regularization penalty

For XGBoost, we used 500 estimators for a binary class logistic distribution and tuned the following parameters:

- Booster: choice of GBTree or DART
- "Learning rate" (eta, η): step size. Introduces randomness
- Max. depth: maximum number of features in each tree. Can cause overfitting if too high
- Min. child weight: minimum sum of weights of observations for a child node. Can cause overfitting if too low

IV. Comparing algorithm performance

After performing the grid search for hyperparameters, we calculate standard errors for the training score from K-fold CV on each algorithm. We then discard trials with score > 1 std. error lower than the one resulting from the combination of hyperparameters that yields the highest training score.

This is done to make the model more parsimonious, as a slightly lower training score might indicate a less overfit (and thus more generalizable) model. On the other hand, the overall best model w.r.t. test score can be the same as the one with the highest training score, so we do not discard the top performer on training.

We then use the remaining parameter combinations above to fit each classifier to the full training data and classify the test data once, and report the highest score for each algorithm (see the Appendix for the full, raw output; and run the accompanying Jupyter notebook for the intermediate pandas DataFrames). Note we do not perform CV on the test data, due to its relatively small sample size, and since this would become a "degenerate case" of nested CV, which we are not interested in.

In our experiment, the best SVC score on the test dataset was 0.979021, with the following parameters:

```
{'C': 10, 'cache_size': 1000,
'gamma': 0.01, 'kernel': 'rbf',
'random_state': 0}
```

In our experiment, the best MLP score on the test dataset was 0.972028, with the following parameters:

```
{'activation': 'logistic',
'alpha': 0.0001,
'learning_rate': 'constant',
'learning_rate_init': 0.001,
'max_iter': 5000,
'momentum': 0.1,
'random_state': 0,
'solver': 'adam'}
```

In our experiment, the best XGBoost score on the test dataset was 0.986014, with the following parameters:

```
{'booster': 'gbtree',
'learning_rate': 0.1,
'max_depth': 3,
'min_child_weight': 1,
'n_estimators': 500,
'objective': 'binary:logistic',
'seed': 0}
```

V. Conclusion

There is only a minute variation between test scores, which could probably become more pronounced if the sample size were larger. In the experiment, XGBoost presented the highest test score. The biggest noticeable difference was between training times.

On the hardware employed for the experiments, SVC had a much faster training time (a few seconds for nested CV) than other classifiers, with a very small decrease in the test score.

scikit-learn's MLP training appeared to be somewhat slow (approx. 2 hours for nested CV) even on a relatively small dataset, and had a slightly larger decrease in the test score. It was not possible to test GPU-accelerated MLP libraries such as Keras, since support for AMD GPUs is inconsistent and highly experimental, even with backends like PlaidML.

Although XGBoost could also not be accelerated on an AMD GPU, it was faster for training than MLP (approx. 1 hour for nested CV) and presented the highest training score.

Considering scores and training time, as well as hardware support, we are inclined to choose XGBoost for the given dataset. This library's GPU-accelerated tree-building methods might allow it to be used effectively in larger but similar datasets, while maintaining a higher number of CV folds and test scores.

Acknowledgments

References

[1] Bergstra, James et al. "Algorithms for Hyper-Parameter Optimization." NIPS (2011). https://papers.nips.cc/paper/4443-algorithms-for-hyper-parameter-optimization.pdf

[2] Cawley, Gavin & Talbot, Nicola. "On Over-fitting in Model Selection and Subsequent Selection Bias in Performance Evaluation." Journal of Machine Learning Research (2011). 11. 2079-2107. http://jmlr.org/papers/volume11/cawley10a/cawley10a.pdf

[3] Wainer, Jacques & Cawley, Gavin. "Nested cross-validation when selecting classifiers is overzealous for most practical applications." (2018). https://arxiv.org/pdf/1809.09446.pdf

[4] World Health Organization. "Breast cancer: prevention and control" https://www.who.int/cancer/detection/breastcancer/en/index1.html