Pipeline Optimization

COSC5557 – Practical Machine Learning

William Baumchen

12/4/2023

1 - Introduction

Given the cost of fitting machine learning models when computing large-scale or exhaustive optimization and considering the relative necessity of altering portions of the process, optimizing a pipeline – a formalized set of processes that typically import data, conduct preprocessing, conduct hyperparameter optimization, and report a final model – becomes an attractive methodology. In this report, the construction and optimization of a relatively simple pipeline will be conducted using the white wine quality dataset, and the result compared to a model of the same type trained using default hyperparameters.

2 – Dataset Description

The dataset used is the white wine quality dataset, containing 4,898 observations, with 11 features, all of which are real numbers taken from a continuous range. There is one 'target' feature, with seven possible classes. There are no missing values in the observations and target feature.

3 – Experimental Setup

In this exercise all computation was done utilizing MATLAB, more specifically the Statistics and Machine Learning Toolbox. Firstly, the dataset was split twice for nested resampling; a fifth of the original dataset was set aside for outer loop testing, and the remaining 80% of the data was split again, with 90% kept for model training and 10% for model evaluation. The pipeline was constructed in three steps – a feature normalization step, a feature reduction step utilizing principal component analysis, and a hyperparameter optimization step conducted using three different classification machine learning models. In this pipeline optimization, an exhaustive search was carried out. The first option was the presence of the normalization step. The second was the number of features removed using principal component analysis. Finally, hyperparameter optimization was carried out on each of the machine learning models using Bayesian optimization. The pipeline optimization target and the hyperparameter optimization target was the validation loss computed during the machine learning model hyperparameter optimization.

Normalization in data pre-processing is a term with a number of definitions. Here, the process refers to scaling the data such that it has a mean of zero and standard deviation of one, while retaining the shape properties of the original data set. In this exercise, the normalization of the white wine quality dataset was completed using the normalize function in MATLAB. In this first step, the pipeline optimization considered whether or not normalizing the dataset was useful.

The second component in the optimization was feature reduction via principal component analysis (PCA). Conducting PCA feature reduction on a dataset results in a transformed set of observations, with features that now correspond in decreasing effective variation in the target feature space of the model. This, more properly defined, is given as an orthogonal linear transformation that transforms the given data to a new coordinate system, such that the greatest variance by some scalar projection of the data comes to lie on the first coordinate (called the first principal component), the second greatest variance on the second coordinate, and so on [1]. Here, PCA was conducted using the pca function in MATLAB. In the second component of the pipeline the pipeline optimization optimized for the number of transformed features to be removed from the dataset used for the machine learning training, ranging from eleven features used to only one feature.

The third component was hyperparameter optimization carried out on three different classification machine learning models. These models were a binary decision tree classifier, a multiclass naive Bayes classifier, and a k nearest neighbors classifier. For the binary decision tree, the hyperparameter optimization optimized the minimum leaf size, the maximum number of decision splits, and the split criterion. The range of parameters used in the hyperparameter optimization was based on the default options set by the fitctree function in MATLAB and reported in Table 1.

Table 1 – Decision Tree Hyperparameter Ranges

Hyperparameter Optimization Ranges – Decision Tree	
MaxNumSplits	Log-scaled integers in [1,max(2,NumObservations-1)]
MinLeafSize	Log-scaled integers in [1,max(2,floor(NumObservations/2))]
SplitCriterion	Searches in [Gini's Diversity Index, Deviance, Twoing rule]

For the multiclass naive Bayes classifier, the hyperparameter optimization optimized the data distribution, the kernel smoother type, whether or not to standardize the data, and the kernel smoothing window width. The range of parameters used in the hyperparameter optimization was based on the default options set by the fitcnb function in MATLAB, and reported in Table 2.

Table 2 – Naïve Bayes Hyperparameter Ranges

Hyperparameter Optimization Ranges – Support Vector Machine	
DistributionNames	Searches between 'normal' and 'kernel'
Kernel	Searches among 'normal', 'box', 'epanechnikov', and 'triangle'
Standardize	Searches among ‘true’ and ‘false’
Width	Searches among real values, by default log-scaled in the range [1e-3,1e3]

For the k-nearest neighbor model, the hyperparameter optimization optimized the distance metric, choosing among a number of predefined metrics, the predefined distance weighting function, the Minkowski distance exponent, the number of nearest neighbors in the dataset to find for classifying each point when predicting, and whether or not to standardize the training data. The range of parameters used in the hyperparameter optimization was based on the default options set by the fitcknn function in MATLAB, and reported in Table 3.

Table 3 – KNN Hyperparameter Ranges

Hyperparameter Optimization Ranges – K-Nearest Neighbor	
Distance	Searches among 'cityblock', 'chebychev', 'correlation', 'cosine', 'euclidean', 'hamming', 'jaccard', 'mahalanobis', 'minkowski', 'seuclidean', and 'spearman'
DistanceWeight	Searches among 'equal', 'inverse', and 'squaredinverse'
Exponent	Positive real values in the range [0.5,3]
NumNeighbors	Searches among positive integer values, by default log-scaled in the range [1, max(2,round(NumObservations/2))]
Standardize	Searches among ‘true’ and ‘false’

For each of models used, kfold cross validation with 5 folds was used to avoid overfitting and bias. These folds were repartitioned with each hyperparameter optimization run. For each run, a budget of 50 iterations of the Bayesian optimization were set, a slight buffer on the default budget of 30 iterations per learner. The Bayesian optimization minimized the validation loss of the machine learning model in question, here defined as the MATLAB cross-validation classification error [2]. In addition, the time elapsed for each optimization was also recorded, but was not part of the optimization.

Once the exhaustive search over the pipeline optimization parameter space was concluded, a simple search for the minimum validation loss was conducted to find the optimal machine learning pipeline for use on the white wine quality dataset, with respect to a model evaluation conducted on the inner set of testing data. Finally, a final evaluation of the models validation loss was carried out on the outer set of testing data.

4 – Results

The pipeline optimization algorithm above was run according to the given conditions. Fig. 1 below shows the entirety of the optimization results, where the results are the minimum observed validation classification loss using the MATLAB classification loss function for each of the optimization evaluation points. As can be seen, here the normalized and unnormalized k nearest neighbors (knn) model performs the best overall. In the figure below, the ‘x-axis’ represents the number of features retained, with the 0 entry being the results without the use of PCA and feature reduction. For both knn models, the best performance occurred when PCA was not in use, and the ‘second-best’ performance occurred when 10 features were retained.

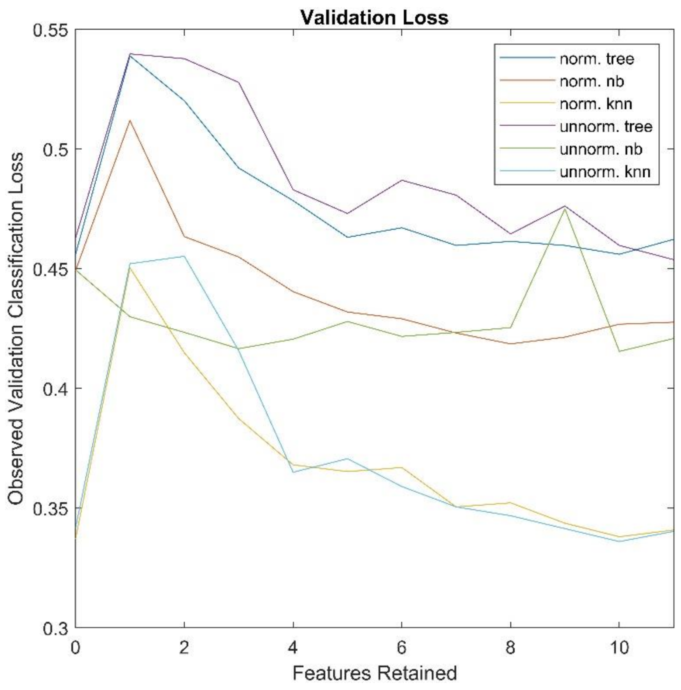

Figure 1 – Observed Validation Loss

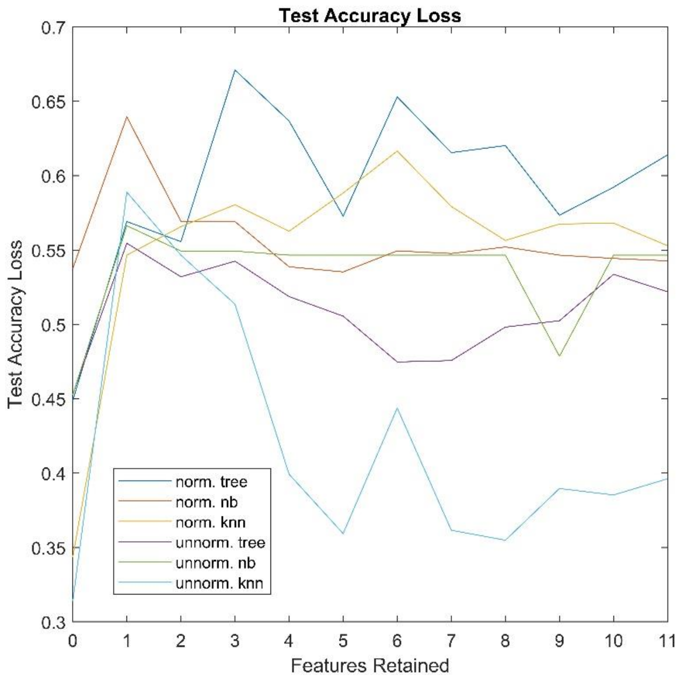

Figure 2 – Test Accuracy Loss

Fig. 2 shows the resulting classification loss of the different models for all possible pipeline configurations with respect to evaluation on the inner loop test data. As can be seen, the knn models once again had the most accurate scores. However, in this plot it is easy to see that the unnormalized knn model with PCA feature reduction was the best performing model, with and having 'second-best' performance with feature reduction, keeping only 8 and 5 features respectively.

As a result of this optimization, the knn model without feature reduction or normalization was kept, and its performance was evaluated using the outer-loop test data set. The resulting classification loss was 0.346466, which corresponds well with the evaluation completed using the inner loop testing data set, as seen in Fig. 2. In addition, it was decided to test the model's performance against a baseline knn model with no hyperparameter optimization, fitted using the fitcknn MATLAB function. This model was also tested against the outer loop testing data, and it was found that its classification loss was 0.452720. In conclusion, the pipeline optimization resulted in a knn model, with no normalization or feature reduction active, and with a resulting classification loss of 0.346466.

References

[1] Jolliffe, I. T. *Principal Component Analysis*. 2nd ed., Springer, 2002.

[2] The MathWorks, Inc. (n.d.). Classification Loss. Loss of k-nearest neighbor classifier - MATLAB. https://www.mathworks.com/help/stats/classificationknn.loss.html#btar64m-2_head