# Homework 3 Writeup

Tylman Michael
CSE 546 Machine Learning

2/18/2023

# 1 4-Fold Cross-Validation Optimal Parameters

## 1.1 Prelude

For this homework, I set out to better understand the GridSearchCV functionality of sklearn and how to utilize it for deeper analysis. My reasoning was that the burden of proof for professional/research applications of a GridSearchCV must be higher than the burden of proof for conclusions required for a class. If that's the case, and GridSearchCV was likely made for use in a professional/research setting like the rest of sklearn, then it must be possible to do quality analysis using this function. So, for every item listed in here, I am only doing operations on either the GridSearchCV, or a single estimator pulled from the GridSearchCV.

I will also list a single table with the best results (with the param lists cut) for every model for ease of reference in Table 1. The most important rows to recognize are the last few, where best_final_test is the accuracy of the best model on the test set, best_cv_test and best_cv_train are the average test accuracies of the best model across the CV splits for test and train respectively.

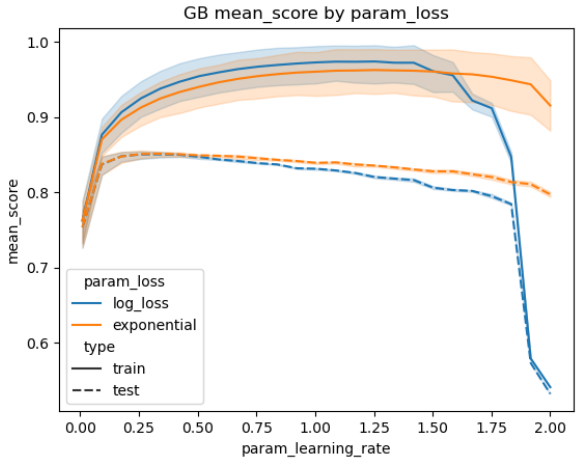
## 1.2 Gradient Boosted

For the Gradient Boosted (GB) classifier, I gridsearched combinations in the learning rate, loss function, and number of estimators. For the learning rate, I did a linspace of (.01, 2, 25). For the estimators I went from 100 to 700 in steps of 100. And for the losses I chose log loss or exponential options. The resulting performances is shown in Figure 1

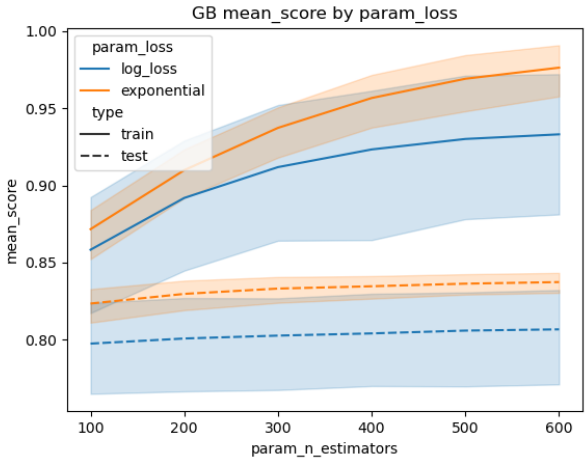
The shaded regions in the two plots show that learning rate had a much stronger influence on the performance of the model than the number of estimators. Interestingly, the training performance of the two loss functions actually inverts when we group by estimators or learning rate. In both cases; however, the exponential bests the log loss function in the test case.

| | gb | knn | rf | nb |
|---|---|---|---|---|
| Index | 47 | 33 | 249 | 8 |
| mean_fit_time | 208.3647250533104 | 0.4263809323310852 | 75.5146564245224 | 0.8436596393585205 |
| std_fit_time | 3.0957062399871327 | 0.0592272222690093 | 4.017492669496358 | 0.03875597162913965 |
| mean_score_time | 0.3202803134918213 | 13.931629002094269 | 5.75344318151474 | 0.09340763092041016 |
| std_score_time | 0.007948119349096923 | 0.13358312729265626 | 0.21164291916351288 | 0.011340124984171198 |
| params | {'learning_rate': 0.25875, 'loss': 'exponentia... | {'n_neighbors': 12, 'weights': 'uniform'} | {'max_depth': 70, 'max_features': 'log2', 'n_e... | {'alpha': 0.21161616161616165, 'fit_prior': True} |
| split0_test_score | 0.85504 | 0.65984 | 0.8448 | 0.82928 |
| split1_test_score | 0.85888 | 0.66496 | 0.84832 | 0.83392 |
| split2_test_score | 0.85296 | 0.66048 | 0.84208 | 0.82896 |
| split3_test_score | 0.85904 | 0.64512 | 0.84944 | 0.83776 |
| mean_test_score | 0.8564799999999999 | 0.6576 | 0.84616 | 0.83248 |
| std_test_score | 0.002587353860607377 | 0.007470475219154384 | 0.0029120439557121756 | 0.0036256861419598 61 |
| rank_test_score | 1 | 1 | 1 | 1 |
| split0_train_score | 0.94736 | 0.7285333333333334 | 0.9999466666666667 | 0.83824 |
| split1_train_score | 0.9471466666666667 | 0.7330666666666666 | 0.9999466666666667 | 0.8366933333333333 |
| split2_train_score | 0.9459733333333333 | 0.74576 | 1.0 | 0.83744 |
| split3_train_score | 0.9439466666666667 | 0.73072 | 0.9999466666666667 | 0.83616 |
| mean_train_score | 0.9461066666666667 | 0.7345200000000001 | 0.9999600000000001 | 0.8371333333333333 |
| std_train_score | 0.0013542361520634087 | 0.006684496158192375 | 2.3094010767592102e-05 | 0.0007841768508017322 |
| best_final_test | 0.86208 | 0.66304 | 0.84412 | 0.83236 |
| best_cv_test | 0.8564799999999999 | 0.6576 | 0.84616 | 0.83248 |
| best_cv_train | 0.9461066666666667 | 0.7345200000000001 | 0.9999600000000001 | 0.8371333333333333 |

Table 1: Best Results and MetaData




(a) Performance Grouped by learning Rate




(b) Performance Grouped by Estimators

Figure 1: GB Performance

## 1.3 KNN

For the KNN classifier, I gridsearched combinations in the Distance metric and the number of neighbors. The number of neighbors varied from 1 to 100, and the Distance metric could be 'uniform','distance' or a custom Radial Basis function. I did not also loop over the possible parameters for the rbf, since including it was a bonus in it's own right. The resulting performances is shown in Figure 1.

This plot shows that the distance and radial basis functions drastically increased training accuracy to be 100% at all values, but did not have a dramatic impact on test performance. This leads me to the conclusion that the non-uniform weighting increases the complexity of our models which opens us up to overfitting.

Interestingly, the uniform parameter showed a stairstep behavior where the even number neighbor counts outperformed the next and previous odd-number amount of neighbors. That's something that caught me by surprise, and I can't quite explain. This behavior worked in the favor of the uniform distribution, allowing it to peak higher than the other weights. So, the best parameter for the weight was the uniform option, which gave better test scores and showed less overfitting.

The best parameter for n_neighbors was less intersting. We can see that it peaks pretty early in the low 10s, with the best parameter being achieved at n_neighbors = 12, which gave us our final results shown in 1.

The best KNN model generalized well, with the accuracy on the test group being quite close to it's average test performance on the validation splits during training. It did perform noticeably higher on the training set, but not to a degree which would be indicative of oppressive over-fitting. All in all, I'd say the KNN model performed quite well.

## 1.4

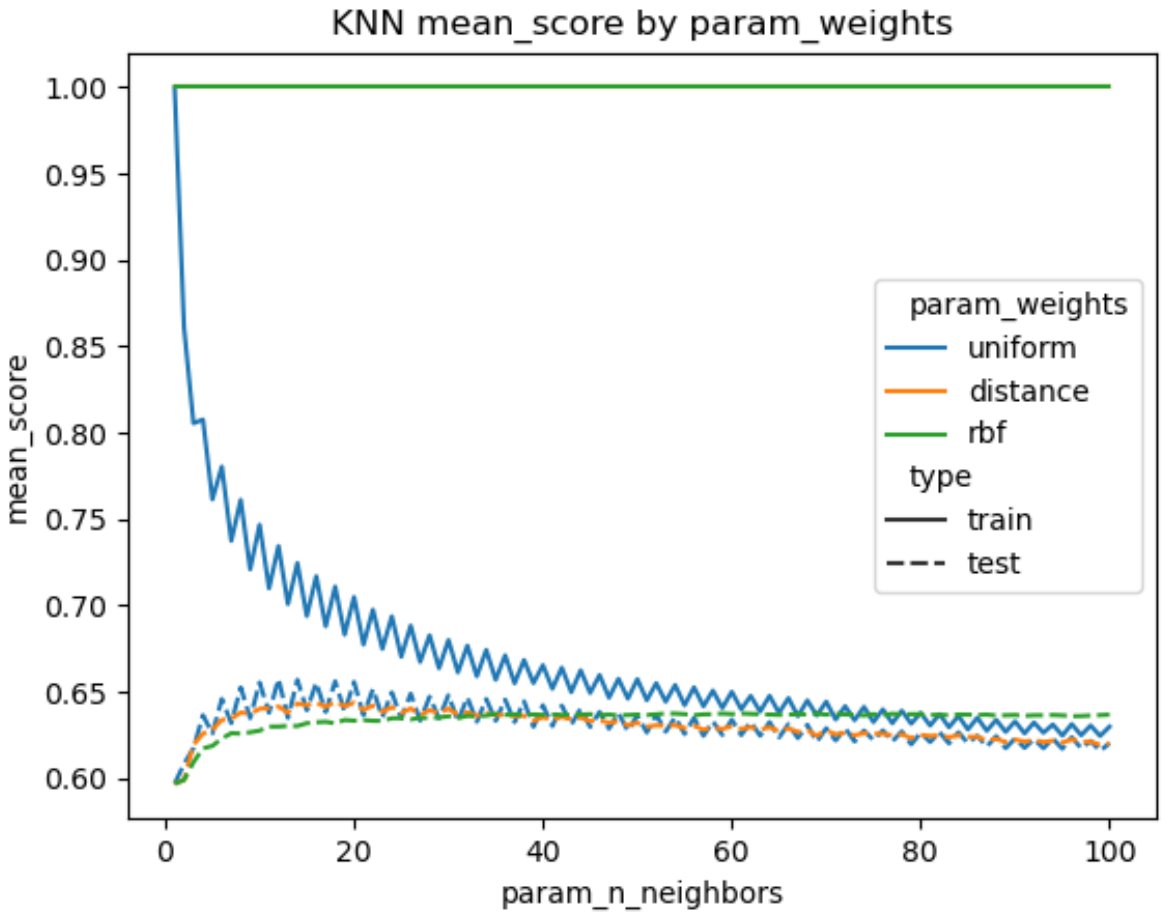



Figure 2: KNN Performance Grouped by Metric