Hyperparameter Optimization

Introduction:

For this exercise I experimented with both the GridSearchCV and BayesSearchCV model selectors on a couple of ML algorithms. I decided on k-nearest neighbors and random forest and compared the accuracy of the models from the default results, GridSearchCV optimized and BayesSearchCV optimized.

Dataset Description:

I used the wine quality dataset[1] for this and picked the white wine dataset because it has more entries than the red dataset. This data includes 11 features and a target value called "quality" that is between 0 and 10.

[1]: https://archive-beta.ics.uci.edu/dataset/186/wine+quality

Experimental Setup:

For the data setup, I went with the white wine as mentioned above, then split the test data to be 20% of the overall data and for the models I went with k nearest neighbors and random forest classifiers, knn was picked because I wanted to play around with the hyperparameters on this model more and random forest was picked because it was the most accurate in the previous exercise.

Accuracy this time is recorded using scikit-learn's accuracy_score function that exists for classification models.

For this experiment I used scikit-learn's GridSearchCV and BayesSearchCV model selectors which I gave a model, a list of hyperparameters, a scoring system and cross validation fold count to determine out of the list of parameters I give it, which ones are the best. BayesSearchCV I experimented with a little bit and reduced it's number of iterations because it was testing with hyperparameters it had already tested on.

I later noticed that the Bayesian optimization on random forest was running faster so I decided to time all of the models and output that too.

Also the Bayesian optimization is set to doing 5 cross validation folds and n_iter set to 15, which is the number of parameter settings sampled.

For the hyperparameters I optimized:

Random Forest:

bootstrap:

The default value for this is true, and I also tested false, this parameter either uses bootstrap samples for true or uses the whole dataset for each tree for false.

max_depth:

The default value for this is None, and I tested this with values 25, 50 and None, this parameter sets the max depth of the tree.

n_estimators:

The default value for this is 100, and I tested this with values 25, 50, 75, 100, 125 and 150, this parameter sets how many random trees the algorithm will generate.

min_samples_split:

The default value for this is 2, and I tested this with values 2, 4, 6, 8, this parameter determines the minimum number of samples before a node can split.

min_samples_leaf:

The default value for this is 1, and I tested this with 1, 3, 5, 7, this parameter determines the minimum number of samples in a leaf.

K-Nearest-Neighbors:

n_neighbors:

The default value for this is 5, and I tested this with values 3, 4, 5, 6, 10, this parameter sets the number of neighbors.

leaf_size:

The default value for this is 30, and I tested this with values 10, 30, 50, 100, 200, this parameter sets the leaf size passed into the balltree and kdtree algorithms.

algorithm:

The default value for this is ‘auto’ and I tested this with 'ball_tree', 'kd_tree', 'brute', 'auto', this parameter changes which algorithm is being used.

p:

The default value for this is 2, and I tested this with both 1 and 2, it changes which distance formula is used.

weights:

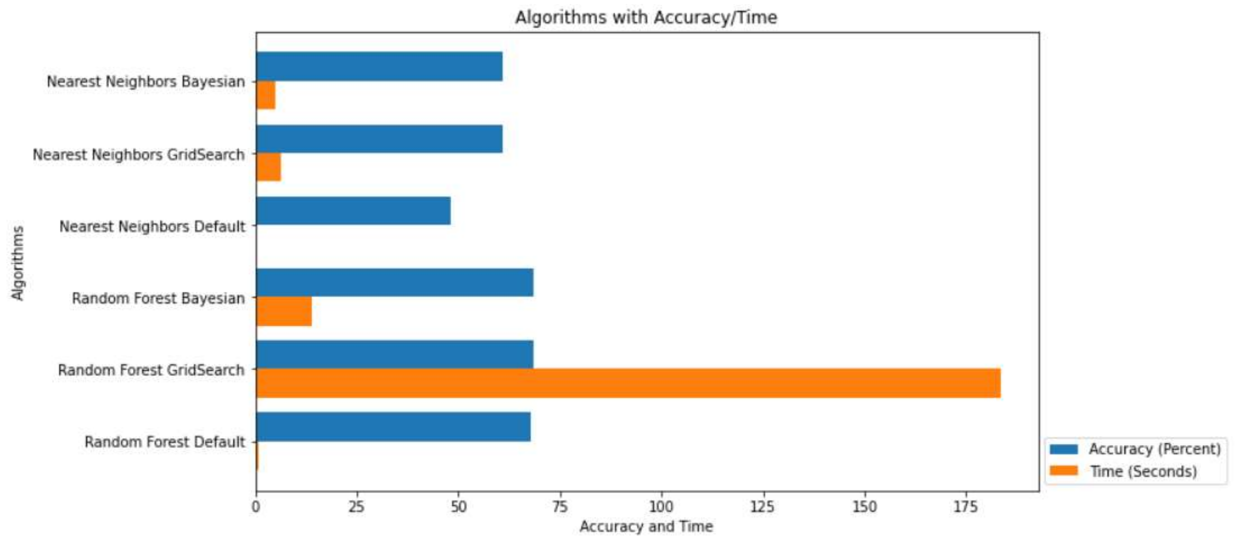
The default value for this is ‘uniform’ and I also tested ‘distance’, uniform weights weighs all points in each ‘neighborhood’ equally, where as distance weights points by the inverse of their distance.

I then would graph the results and make a list of the algorithms sorted by accuracy[2] then by time.

We also output the best hyperparameters found by each model selector.

[2]: code for that inspired by https://stackoverflow.com/questions/48053979/print-2-lists-side-by-side user SCB

Results:



```
Best Paremeters found by gridsearch random forest:  {'bootstrap': True, 'max_depth': 25, 'min_samples_leaf': 1, 'min_samp
les_split': 2, 'n_estimators': 150}

Best Paremeters found by baysesian random forest:  OrderedDict([('bootstrap', False), ('max_depth', 50), ('min_samples_l
eaf', 1), ('min_samples_split', 8), ('n_estimators', 50)])

Best Paremeters found by gridsearch knn:  {'algorithm': 'ball_tree', 'leaf_size': 10, 'n_neighbors': 10, 'p': 1, 'weights': 'distan
ce'}

Best Paremeters found by baysesian knn:  OrderedDict([('algorithm', 'kd_tree'), ('leaf_size', 200), ('n_neighbors', 10), ('p',
1), ('weights', 'distance')])


List of algorithms sorted best to worst:

Random Forest Bayesian: 68.67347% accuracy, 14.02394 seconds
Random Forest GridSearch: 68.57143% accuracy, 183.49651 seconds
Random Forest Default: 67.95918% accuracy, 0.88076 seconds
Nearest Neighbors Bayesian: 61.02041% accuracy, 4.92789 seconds
Nearest Neighbors GridSearch: 61.02041% accuracy, 6.23135 seconds
Nearest Neighbors Default: 48.06122% accuracy, 0.03653 seconds
Best Algorithm by accuracy and time is: Random Forest Bayesian
```

As we can see from the output Bayesian optimization outperformed the gridsearch in much less time for both algorithms tested. Both hyperparameter optimizations outperformed their default counterparts though and my conclusion is that for a more complex problem, Bayesian optimization would be better because it runs faster for similar or even better results.

For random forest the improvement was pretty small, this is likely due to the problem being too simple and the default values being good.

For nearest neighbor the hyperparameters were fairly different, but yielded the exact same accuracy.

Code:

In the repository under main.py.