|  |
| --- |
| Name: Samrat Bikram Shah  ID:  Email: |

**Random Search Optimisation and Meta Learning**

**Introduction:**

Machine Learning and Artificial Intelligence is a major part of technology that requires a lot of complex calculations, proper structure, and the correct set of parameters to work correctly. The machine does most of the calculation part itself however choosing the correct model/s and parameters and tuning it for the best possible accuracy is all on us. This process is called optimization. There are many ways to perform the optimization task. Some of the popular ones that will be used here are Random Search, Meta Learning and Adaptive Boosting. Usually, in machine learning, hyperparameters are adjusted. Hyperparameters are the parameters that are adjusted for the learning process. Hyperparameters are finalized based on how efficient and good a result it produces on the corresponding model as compared to other values of the same type.

In this assignment, it is expected to use an optimal classification model and find a set of parameters within one or more optimization strategies. It also expects an explanation of the pros and cons of using certain techniques and how it affects the results. For this, multiple sets of experiments are conducted on different models with various hyperparameter optimization techniques performed and the results are compared. While Random Search helps in basic optimization of hyperparameters, Adaboosting is used for the highest possible accuracy with a minimum amount of data. It combines multiple weak classifiers into a strong one and is a well-optimized model. It is used to obtain the best possible accuracy with the lowest errors and is used with many other algorithms to improve performance and reduce overfitting. It is a type of **Meta learning** algorithm. Meta Learning is basically learning to learn. It trains a model using the least number of data as possible. Hence, while Random search helps to optimize the hyperparameters, AdaBoosting helps to optimize the structure and performance of the model itself.

**Designing a solution:**

The solution is designed on a similar setup and model as the previous assignment. The initial experiments are conducted with a **Multilayered Perceptron model**. The training and testing set of data are applied **Principal Component Analysis** which makes the data easier to train and more efficient. Though the model and the approach are similar, **random search** is applied in the following experiments in the MLP classifier for hyperparameter optimization.

Instead of a hit and trial method as in the previous assignment, to find the optimal hyperparameters this time the optimization algorithm will itself find the best possible set of values from where we can adjust our model.

Random Search sets up a grid of hyperparameter values and selects random combinations to train the model and score. (Koehrsen, 2018) It is known to find a good combination of hyperparameters in very few iterations which saves a lot of time. However, as the selection of parameters is completely random, there is a lot of luck involved in this. **K-Fold** is then applied with random search with **cv=5** to achieve 5 fold cross-validation to our dataset and to compare the results with previous assessments. Even though this many hyperparameter optimization and validations are performed, the data still suffers from some overfitting.

Optimization techniques besides Random search should be applied to focus not only on the hyperparameters but on the **model** and **structure** itself. For this, **Meta-Learning** can be used. **Adaboosting** is a type of Meta Learning. It uses multiple weak classifiers to make a strong one. In AdaBoosting, the weights of the training examples that were incorrectly predicted by the boosted model in the previous step are increased, meanwhile, the weights for those that were predicted correctly are decreased. (1.11. Ensemble methods — scikit-learn 0.23.1 documentation, 2011)

**Experiments:**

**First Experiment:**

Firstly, to start the experiment, the **MLP Classifier** is used with **random search** hyperparameter optimization. As in the previous assignments, the Principle Component Analysis is applied on the training data to remove redundancy and for efficiency. In this case, the hyperparameters that are **randomized** are the **learning rate initialization**, **number of hidden neurons**, **batch size** and **learning rate type.**

By running this test **five** times the following results could be achieved.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Experiment** | **hidden\_layer** | **learning\_rate** | **learning\_rate\_init** | **batch\_size** | **Accuracy** |
| 1 | 2936 | adaptive | 0.0091 | 128 | 94% |
| 2 | 2348 | constant | 0.0106 | 32 | 95% |
| 3 | 2442 | adaptive | 0.0086 | 32 | 99% |
| 4 | 2772 | constant | 0.0084 | 32 | 98% |
| 5 | 2802 | constant | 0.0042 | 128 | 96% |

From looking at the results obtained from random search, we can see very varying results.

Training of the models is done using gradient descent where the errors are estimated and weights are updated. Our model is trained using **stochastic gradient descent** which chooses random batches of data for training.

The number of training examples used in one iteration is the **batch size** hyperparameter of the model. The results of batch sizes we obtain are either **32** or **128** which means **32** or **128** training examples are chosen in one iteration for the best possible results. Batch sizes are usually set smaller than the training dataset because they give the best stability and general performance with a certain computational power. It performs better in the same or less amount of training time than large batch sizes. It affects how quickly a model learns.

So, in this case choosing aslow batch size as possible is optimal which is **32** in this case. It can also be seen **batch size=32** having a better performance and chosen more times by the random search than the 128 batch size.

**Next** is the number of **hidden neurons** or **hidden layers**.

The number of hidden neurons generally should be at least ⅔ of the input size adding the size of the output layer. Increasing hidden neurons increases more logical operations performed however it takes way more computational power and time for even a single epoch. Judging the number of hidden layers chosen they are all around the same mark so basing off on performance **Experiment 3 and 4 did the best.**

Moving on to the **learning rate type** and the **learning rate init** hyperparameters, learning rate init is the initial learning rate that is used while learning rate type is the way the weights are updated.

In case of **learning rate init**, the lower the learning rate the less number of steps are taken while gradient descent which aims to reach the minima of the function though it might take more time however if the learning rate is too high the gradient is descended at a faster pace but at the risk of skipping the minima. So the learning rate must be as low as possible taking time into account. In the experiments above **3 and 4** seems like the **best options** with a balance of **low learning rate** and a **good performance**. Lastly, for the **learning rate type**, in ‘**adaptive’** learning the learning rate is adjusted if needed while ‘**constant’ learning** keeps it the same throughout the whole training.  
  
By matching all the criteria we can conclude, **Experiment 3** to have the **best** possible results in all hyperparameters and even in accuracy.

**Best result: 99%**

**Second Experiment:**

**K-Fold cross validation and Random Search**

For these next sets of experiments, we will use K-Fold cross validation in conjunction with random search and look for the best possible set of hyperparameters using K-Fold. The dataset currently used are splitted into testing and training and are fixed and unchanged. Hence, the data overfits and is unable to recognize newer patterns. Hence, K-Fold cross validation is used which allows the dataset to be **randomly divided** into both training and testing dataset making it easier to recognize newer patterns.

In this case, where the dataset is divided into **5 folds** of training and testing dataset, the following hyperparameter values are obtained.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Experiment** | **hidden\_layer** | **learning\_rate** | **learning\_rate\_init** | **batch\_size** | **Accuracy (Avg)** |
| 1 | 2115 | adaptive | 0.0104 | 128 | 88.8% |
| 2 | 1726 | adaptive | 0.0049 | 32 | 90.9% |
| 3 | 2994 | constant | 0.0042 | 128 | 90.1% |
| 4 | 2561 | adaptive | 0.0021 | 32 | 91.1% |
| 5 | 2541 | constant | 0.0042 | 32 | 91.8% |

The results in the previous assessment **without** any hyperparameter optimization ranged from **79% to 81%** with **k = 5 values**. The difference in result is significant when hyperparameters are optimized with an improvement **over 10%** in performance. The way the hyperparameters work are similar in this case as well.

So, **Experiment 4** seems like the **best** option while using k fold cross validation and is more effective while predicting newer models.

**Best result: 91.1%**

**Advanced Experiments:**

These sets of experiments will be conducted using Meta Learning and Adaboosting which is known to combine multiple weak classifiers into a strong one. The bias and variance of such models are usually low. While in this case the model also learns from its mistakes from previous models. (Chen, 2020)

**Meta Learning and Adaboosting**

In these experiments, Adaboost will be used in conjunction with random search. Here, the parameters that are adjusted and optimized are **learning\_rate** and **n\_esitmators**.

For this a base\_estimator will be used. Base estimator is the classifier that is used as a **weak model** for classifying. The two base estimators that will be used are **Decision Tree** and **Support Vector Classifier(SVC)**.

Firstly, Decision Tree is used in the adaboosting model where **Max\_depth** is also altered but since the value of max\_depth usually ranges from 1 2 and 3, before the model starts to **overfit**. Hence,the hyperparameters of the Adaboost model will be optimized using Random search based on the max depths.

**Using max\_depth =1**

|  |  |  |  |
| --- | --- | --- | --- |
| **Experiments** | **n\_estimators** | **learning\_rate** | **Accuracy (Avg)** |
| 1 | 2827 | 0.0848 | 72.3% |
| 2 | 2752 | 0.0040 | 63.5% |
| 3 | 1917 | 0.0076 | 61.3% |

Even though the accuracy of the model is very low, since we know the proper hyperparameter estimation from random search, it saves us a lot of time in finding the proper hyperparameter values for better accuracy which in this case is **experiment 1** from above.

Moving on, we will be using the value of max\_depth as 2 and performing the same experiment as follows:

**Using max\_depth = 2**

|  |  |  |  |
| --- | --- | --- | --- |
| **Experiments** | **n\_estimators** | **learning\_rate** | **Accuracy (Avg)** |
| 1 | 1218 | 0.0072 | 79.5% |
| 2 | 1377 | 0.0076 | 82.4% |
| 3 | 600 | 0.0103 | 71.2% |

From this, we can observe a general trend of increase in accuracy as the max\_depth **increases** and n\_estimator **increases.**

Hence, lastly using the value of max\_depth as 3 we get

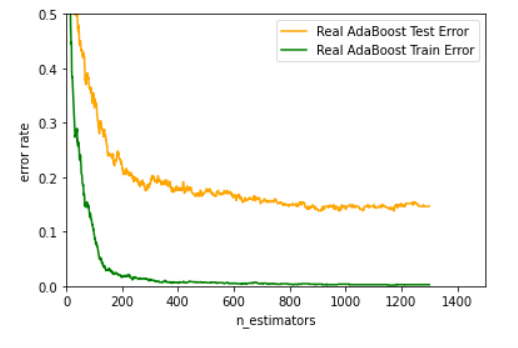
**Using max\_depth = 3**

|  |  |  |  |
| --- | --- | --- | --- |
| **Experiments** | **n\_estimators** | **learning\_rate** | **Accuracy (Avg)** |
| 1 | 497 | 0.0046 | 82.4% |
| 2 | 353 | 0.0070 | 82.7% |
| 3 | 428 | 0.0101 | 84.5% |

Looking at the data obtained from random search and the corresponding accuracy, it does give very low accuracy as compared to previous models however it can be observed as the **max\_depth** and **n\_estimators increase**, there is a **spike** in accuracy.

As for the learning rate, it is the same as the learning rate for MLP Classifiers. It is the steps taken for gradient descent which should generally be low so it can successfully reach the minimum point. Because if it is high, it will descend quicker but might skip the minimum point completely. In this case, the learning rate is the weight of weak learners.

Likewise, the other hyperparameter **n\_estimator** is the number of weak learners that are trained in each iteration. If it’s increased a lot there is a chance for the data to overfit as we can observe from the graph. The difference between test and train data keeps increasing as the graph progresses for **max\_depth =3.**



Lastly, the **max\_depth** is only applicable for the Decision Tree classifier which is the depth of the tree i.e n\_estimators on one level of the decision stump which is responsible for making predictions. It is generally more **effective** to increase the max\_depth however since nodes required for the decision trees increase having a higher max depth can cause **overfitting**. Meanwhile, having too low of max\_depth is known to cause underfitting.

So, we can observe very good accuracy in the experiments conducted with **max\_depth=3.**

From all this, we can conclude the optimal model being the experiments where max\_depth = 3 with the best one being experiment 3 with 428 n\_estimators and 0.01 learning rate.

**Final Experiment:**

As much optimization as it may be, the accuracy of the classification is still very low compared to the previous model so **changing** the **base estimator** to another classification model might be a good idea. Our goal is not only to optimize the hyperparameters but also the structure and model to obtain the best performance and results. In our case, we have been using DecisionTrees as the base estimator which uses trees, depth and stump to classify our data however **Support Vector Classifier (SVC)** can also be used.

SVC works with **hyperplanes** to classify which are multidimensional planes rather than trees. Hyperplanes are boundaries that help to make decisions to classify the data. Data falling on either side of the hyperplane can be classified into various classes. (Gandhi, 2018) So, a boost in performance can be expected in this model in conjunction with Adaboost.

Since using random search directly on SVC consumed a huge amount of time even for a single test run, we will be using the hyperparameter values from the experiments above with the best performance.

With **SVC**, the following results can be obtained

|  |  |
| --- | --- |
| **Experiments** | **Accuracy** |
| **1.** | 97.33% |
| **2.** | 99.47% |
| **3.** | 98.67% |
| **4.** | 97.07% |
| **5.** | 97.33% |
| **Average Accuracy** | **97.97%** |

From the experiments run we can easily observe the drastic change in the accuracy of the model. It is not only high but consistent throughout the 5 runs too. This shows the potential of **hyperplanes** as a weak learner for classification when used in conjunction with Adaboosting with correct hyperparameter values.

Hence, the **final** average accuracy of the model is **97.97%** with the highest accuracy being **99.47%.**

**Conclusion:**

In conclusion, Random Search and Meta Learning proved to be a very effective optimization technique being able to cope not only with the hyperparameters but also the structure improving the performance of the model throughout the series of experiments conducted. Firstly, with MLP, then with K-Fold and lastly combining Meta Learning and Random Search in Adaboosting. The performance of the model has increased considerably throughout even as compared to last assessment also tackling problems such as overfitting or bias. With MLP it secured the best accuracy of **99%** and **91.1%** when K-Fold cross-validation was used. Similarly, the accuracy in the final stages of Adaboosting reached up to **99.47%** with the best performance overall which also remained consistent in the high 90s throughout.

**References:**  
  
Koehrsen, W., 2018. *Intro To Model Tuning: Grid And Random Search*. [online] Kaggle.com. Available at: <https://www.kaggle.com/willkoehrsen/intro-to-model-tuning-grid-and-random-search> [Accessed 26 May 2020].

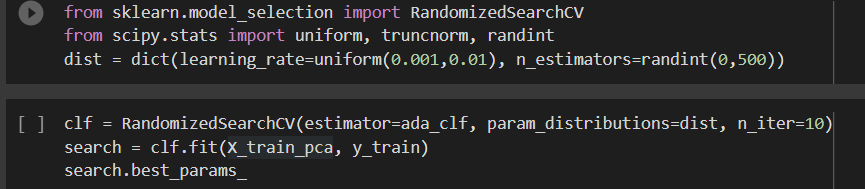
Scikit-learn.org. 2011. *1.11. Ensemble Methods — Scikit-Learn 0.23.1 Documentation*. [online] Available at: <https://scikit-learn.org/stable/modules/ensemble.html> [Accessed 21 June 2020].

Chen, L., 2020. *Basic Ensemble Learning (Random Forest, Adaboost, Gradient Boosting)- Step By Step Explained*. [online] Medium. Available at: <https://towardsdatascience.com/basic-ensemble-learning-random-forest-adaboost-gradient-boosting-step-by-step-explained-95d49d1e2725> [Accessed 8 July 2020].

Gandhi, R., 2018. *Support Vector Machine — Introduction To Machine Learning Algorithms*. [online] Medium. Available at: <https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47> [Accessed 8 July 2020].

**Appendix:**

Random Search code snippets



Meta Learning and Adaboost code snippets

