12-2020

Random Search Plus: A more effective random search for machine learning hyperparameters optimization

Bohan Li
bli43@vols.utk.edu

To the Graduate Council:

I am submitting herewith a thesis written by Bohan Li entitled "Random Search Plus: A more effective random search for machine learning hyperparameters optimization." I have examined the final electronic copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Computer Science.

Bruce J.MacLennan, Major Professor

We have read this thesis and recommend its acceptance:

Bruce J.MacLennan, Audris Mockus, Amir Sadovnik

Accepted for the Council:
Dixie L. Thompson

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

Random Search Plus: A more effective random search for machine learning hyperparameter optimization

A Thesis Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

Bohan Li

December 2020

Acknowledgments

I would like to thank my graduate committee for their support and help in this thesis. I want to thank Dr. Bruce MacLennan for his support and encouragement in my research, for his teaching me machine learning and for his review of my paper. I would like to thank Dr. Amir Sadovnik for his teaching me deep learning and reinforcement learning, for his suggestions about focused grid search which I never want to know before. I want to thank Dr. Audris Mockus for his teaching me digital archaeology, for his suggestions about sobol sequences and Lipshitz functions which broadened my horizons and also updated my knowledge a lot. I want to thank my graduate discussion group for their insights and patience: Allen McBride, Casey Miller. Your great ideas and thinking about research inspired me a lot.

Finally, I would like to thank my dear family for their support on my study, for everything they have done for me in the past two years. I also want to thank my friends and classmates who have given me time to listen to my opinions and given me help when I encountered difficulties in writing the thesis.

Abstract

Machine learning hyperparameter optimization has always been the key to improve model performance. There are many methods of hyperparameter optimization. The popular methods include grid search, random search, manual search, Bayesian optimization, population-based optimization, etc. Random search occupies less computations than the grid search, but at the same time there is a penalty for accuracy. However, this paper proposes a more effective random search method based on the traditional random search and hyperparameter space separation. This method is named random search plus. This thesis empirically proves that random search plus is more effective than random search. There are some case studies to do a comparison between them, which consists of four different machine learning algorithms including K-NN, K-means, Neural Networks and Support Vector Machine as optimization objects with three different size datasets including Iris flower, Pima Indians diabetes and MNIST handwritten dataset. Compared to traditional random search, random search plus can find a better hyperparameters or do an equivalent optimization as random search but with less time in most cases. With a certain hyperparameter space separation strategy, it can only need 10% time of random search to do an equivalent optimization or it can increase both the accuracy of supervised learning and the silhouette coefficient of a supervised learning by 5%-30% in a same runtime as random search. The distribution of the best hyperparameters searched by the two methods in the hyperparameters space shows that random search plus is more global than random search. The thesis also discusses about some future works like the feasibility of using genetic algorithm to improve the local optimization ability of random search plus, space division of non-integer hyperparameters, etc.

Table of Contents

1 Introduction **1**

1.1 Hyperparameter Optimization Methods 2

1.1.1 Comparison between GS, RS, RS+ 5

1.1.2 Other Methods and Comparisons 7

1.2 Machine Learning Methods 8

1.2.1 K-nearest Neighbor Algorithm 8

1.2.2 K-Means 9

1.2.3 Neural Network 11

1.2.4 Support Vector Machine 12

2 Definitions and Assumptions **14**

2.1 Overview 14

2.2 Definitions 14

2.3 Assumptions 15

3 Implementations and Methods **17**

3.1 Overview 17

3.2 Space Separation and Search 17

3.3 Core Algorithm 20

3.4 Extreme Cases Discussion 22

3.5 Experiment Descriptions 25

3.5.1 General information 25

3.5.2 Experiment 1 27

3.5.3 Experiment 2 . . . 28
3.5.4 Extra Experiment: . . . 29

4 Results **31**
4.1 Overview . . . 31
4.2 Experiment 1 . . . 31
4.2.1 K-NN . . . 31
4.2.2 K-Means . . . 37
4.2.3 Neural Network . . . 42
4.2.4 Support Vector Machine . . . 47
4.3 Experiment 2 . . . 52
4.3.1 K-NN . . . 52
4.3.2 K-Means . . . 57
4.3.3 Neural Network . . . 62
4.3.4 Support Vector Machine . . . 67
4.4 Extra experiment . . . 72
4.4.1 Random search plus vs grid search (neural network) . . . 72

5 Conclusions and Future Work **76**
5.1 Conclusions . . . 76
5.2 Future Work . . . 77
5.2.1 Experiments Improvement . . . 77
5.2.2 Algorithm Improvement . . . 77

Bibliography **80**

Appendices **84**

Vita **85**

List of Figures

1.1 Workflow of Grid Search 3
1.2 The Relationship between GS, RS, RS+ 6
1.3 K-NN 10
1.4 SVM 13

3.1 2-D Case 19
3.2 3-D Case 21
3.3 Core Algorithms 23
3.4 Hyperparameter Settings and Datasets 26

4.1 Accuracy Distributions for K-NN in Experiment 1 33
4.2 Average Accuracy for K-NN in Experiment 1 34
4.3 Total Runtime for 100 Trials 35
4.4 Hyperparameters Returned for Each Run of Each Method in K-NN's Hyperparameter Space 36
4.5 Silhouette Coefficient Distributions for K-means in Experiment 1 38
4.6 Average Silhouette Coefficient for K-means in Experiment 1 39
4.7 Total Runtime for 100 Trials for K-means in Experiment 1 40
4.8 Hyperparameters Returned for Each Run of Each Method in K-means Hyperparameter Space 41
4.9 Accuracy Distributions for Neural Network in Experiment 1 43
4.10 Average Accuracy for Neural Network in Experiment 1 44
4.11 Total Runtime for 1000 Trials for Neural Network in Experiment 1 45

4.12 Hyperparameters Returned for Each Run of Each Method in Neural Network Hyperparameter Space 46
4.13 Accuracy Distributions for SVM in Experiment 1 48
4.14 Average Accuracy for SVM in Experiment 1 49
4.15 Total Runtime for 100 Trials for SVM in Experiment 1 50
4.16 Hyperparameters Searched for Each Run of Each Method in for SVM in Experiment 1 51
4.17 Accuracy Distributions for K-NN in Experiment 2 53
4.18 Average Accuracy for K-NN in Experiment 2 54
4.19 Runtime for Each Run for K-NN in Experiment 2 55
4.20 Best Parameters Returned for Each Run for K-NN in Experiment 2 56
4.21 Silhouette Coefficient Distributions for K-means in Experiment 2 58
4.22 Average Silhouette Coefficient for K-means in Experiment 2 59
4.23 Runtime for Each Run for Each Run for K-means in Experiment 2 60
4.24 Best Parameters Returned for Each Run for K-means in Experiment 2 . . . 61
4.25 Accuracy Distributions for Neural Network in Experiment 2 63
4.26 Average accuracy for Neural Network in Experiment 2 64
4.27 Runtime for Each Run for Neural Network in Experiment 2 65
4.28 Best Parameters Returned for Neural Network in Experiment 2 66
4.29 Accuracy Distributions for SVM in Experiment 2 68
4.30 Average Accuracy for SVM in Experiment 2 69
4.31 Runtime for Each Run for SVM in Experiment 2 70
4.32 Best Parameters Returned for Each Run for SVM in Experiment 2 71
4.33 Accuracy by Random Search Plus and Grid Search 73
4.34 Runtime by Random Search Plus and Grid Search 74
4.35 Best Parameters Returned by Random Search Plus and Grid Search 75

5.1 Code Improvement 78

Chapter 1

Introduction

Machine learning is the study of computer algorithms that can optimize its performance by itself through example data. Machine learning was first proposed by Arthur Samuel in 1952 and before long, the perceptron, the first major breakthrough, was introduced into machine learning. With decades of development, nowadays many machine learning algorithms have emerged and matured. The common machine learning algorithms are: k-nearest neighbors algorithm, k-means, support vector machine and neural network.

The current application areas of machine learning are very wide. These areas include data mining, computer vision, natural language processing, and so forth.

The birth of a machine learning algorithm's program usually goes through the following processes:

1.The collection, processing and division of the dataset.

2.Choice an appropriate machine learning algorithm according to the dataset

3.Set the parameters of the model

4.Training model

5.Model testing and validation.

At the step 1, a machine learning data set often contains two parts: features and class (or labels). The datasets are often divided into two types: labeled and unlabeled. When faced with unlabeled data sets, we will choose unsupervised learning. Otherwise, we will choose supervised learning.

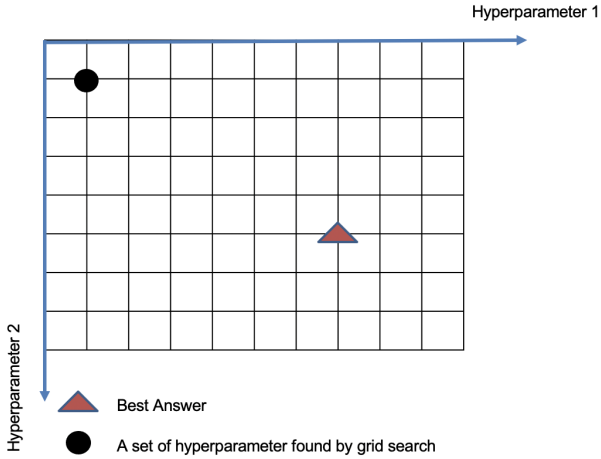
An **unsupervised learning** is a machine learning algorithm that doesn't require training

datasets to have own label or class in its training process[8]. Similarly, a machine learning algorithm that requires training datasets to have their own label or class in training process is a **supervised** learning[8].

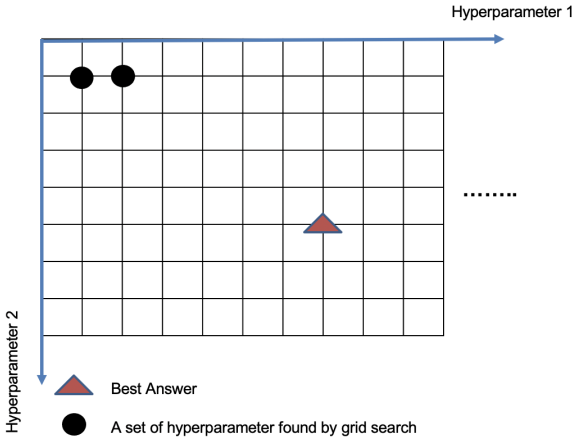
Hyperparameter in machine learning models is a parameter that is manually given before training and will not be changed during the training process. Different choices of hyperparameter can produce different models for a same machine learning algorithm and also the performance of this machine learning program will vary according to different models. Some combinations of hyperparameter of the machine learning algorithm will generate a set of models with extraordinary performance but also some combinations will case the machine learning program to have really low performance and even to be unable to work in actual use. Therefore, a process to filter or skip a large quantity of models and only leave the best model is necessary to make machine learning algorithm work in practice and the process can be viewed as an optimization for machine learning hyperparameter.

1.1 Hyperparameter Optimization Methods

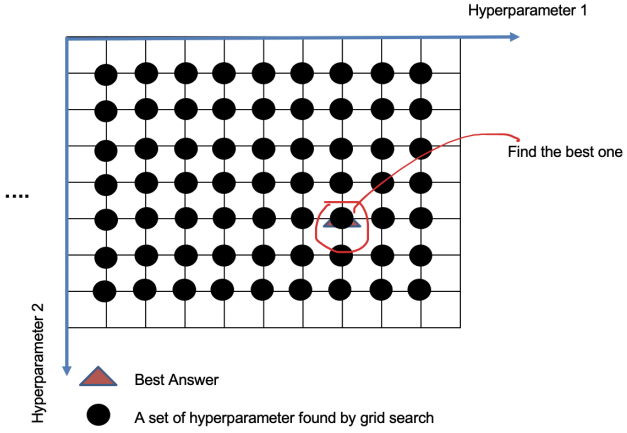
Machine learning hyper-parameter optimization is an approach to find out a combination of hyperparameters for a machine learning model or algorithm where such a combination of hyperparameters can improve the performance of the machine learning model or algorithm[16]. Not only the choosing an appropriate model based on the data but also an excellent optimization plays an essential role in the process for a machine learning program from the coding work to its application. Currently, there are many popular and different methods to do a hyperparameter optimization for machine learning algorithms. The most popular and most widespread methods include grid search(See figure 1.1), random search, Bayesian optimization, evolutionary optimization, population-based optimization and manual optimization. However, compared with manual optimization, these optimization methods either have to face a huge computing ability challenge or are less accurate than manual optimization methods[5].



(a) 1st step of grid search



(b) 2st step of grid search



(c) 63 st step of grid search

Figure 1.1: Workflow of Grid Search

Before discussing hyperparameter optimization in this thesis, there is a major premise that needs to be emphasized. This is that all the parameters discussed in this thesis are discrete. If we encounter continuous hyperparameters, we will discretize them.

Grid search(GS) is a way to list all combinations of hyperparameter, then search each of them in order and finally find the best combination for a certain machine learning algorithm(See figure 1.1). It is the most basic method to do a machine learning tuning. Grid search can find the best solution of hyperparameter optimization as it will search and try all combinations of hyperparameter. However, most machine learning algorithms have a very expensive training process, which usually has very strict requirements for computing ability and also consumes a lot of computing resources.

Random search(RS) actually is a very general concept. In tradition, it can be described as a set of numerical optimization methods that can solve a problem without any optimized gradient of the problem[13]. In machine learning area and especially in hyperparameter optimization, it can be defined as an optimization algorithm that samples the search space by randomness or probability[17]. There are a lot of papers have proven that random search is a better method than the grid search as the dimension of hyperparameter increases[17][3][5].

RandomizedSearchCV is a common method for machine learning hyperparameters optimization in Python-sklearn module[12]. It is based on an article[2] proposed by James Bergstra and Yoshua Bengio but in addition, RandomizedSearchCV use cross-validation to verify the correctness of the results rather than basic validation. The process of how RandomizedSearchCV works in scikit-learn module can be briefly described as:

1. For hyperparameter whose search range is distribution, randomly sample according to the given distribution.

2. For hyperparameter whose search range is list, sample with medium probability in the given list.

3. Traverse the n_iter group sampling results obtained in steps a and b but if the given search range is all list, do not return to sampling n_iter times.

Random search plus (RS+) is a new method proposed by this paper. It can be defined as a method to separate the search space or hyperparameter space(See Definitions)

and randomly sample each hyperparameter subspace(See Definitions), then return the best solution from those samples.

1.1.1 Comparison between GS, RS, RS+

In general, the main difference between grid search, random search and random search plus is their different sampling methods. Different sampling methods lead to different sampling efficiencies of them and eventually lead to the different expected value of their sampling and also their different running time.

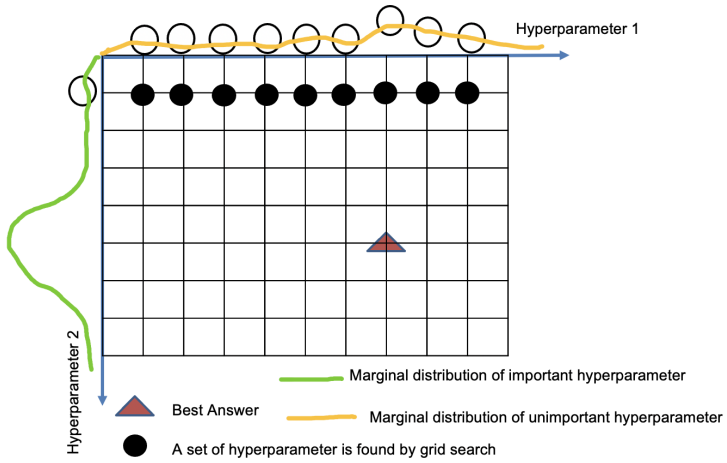
A grid search is a exhaustive search method, which means that it will sample all points in a search space. This way makes the result perfect while it will cost a lot of time compared to others. Random search with sampling points in a whole search space with probability or randomness[17]. Random search can work better than grid search. Here will be an example of sampling points in a search space which can vividly show what the difference between them is, what the relationship between them is, why random search works better than grid search and why random search plus works better than both of them.

Assume before we do a hyperparameter optimization, we don't have any knowledge about the different influence of parameters on the model's performance. Grid search wastes time on unimportant parameter(Figure1.2-a). Random search is better then grid search with same times of sampling(same points) because it explores the important parameter(Figure1.2-b). However, random search plus get the almost same result to random search's but with less samplings(less points) because after the space separation or division, the two sub areas or hyperparamter subspace(See Definitions) at bottom(c) almost cover the good points. Random search plus must sampling a point for each sub area(hyperparamter subspace), so the probability of getting good one for random search plus is larger than for random search.

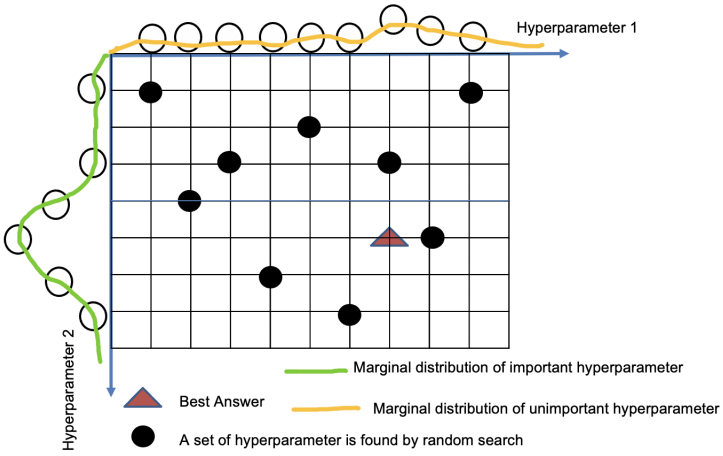
Summary about the difference between the three:

1. Random search and random search plus both are random search methods just with different way of sampling.

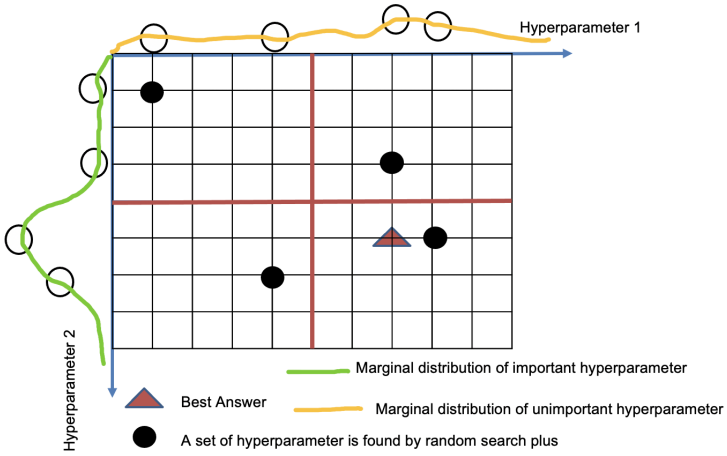
2. A purely random search is neither systematic nor exhaustive, but it might miss good solutions by under-sampling some regions of the hyperparameter space(See definition).



(a) Grid search



(b) Random search



(c) Random search plus

Figure 1.2: The Relationship between GS, RS, RS+

2. Such chance of missing good solutions for random search can be decreased by doing more sampling but the way of random search plus to sample is more efficient than random search. This is mainly because random search plus divides the hyperparameter space into regions or cells and visits them systematically, so that no cell is neglected. Then it does random sampling in each cell, which is less expensive than searching it exhaustively. Therefore it has a lower probability of missing a good solution than a pure random search.
3. Grid search is a systematic and exhaustive search. It will find the best hyperparameter values, but it's expensive.
4. On one hand, if the entire hyperparameter space is one cell, then random search plus reduces to random search. On the other hand, if the cells are so small that there is only one grid point per cell, then random search plus reduces to something similar to grid search but it's not exactly grid search, since random search plus samples the cells randomly.
In the follow chapters, more concepts and details about random search plus will be discussed.

1.1.2 Other Methods and Comparisons

Grid search and random search are model-free methods of hyperparameters optimizations which can optimize the machine learning algorithm without any knowledge about them[10]. Random search plus is a random search method with a better way of sampling so it is also a model-free method of hyperparameters optimization. The general difference we have discussed in last subsection and we will also talk about more in the reset of this paper, so in here, we will talk about some popular others methods of hyperparameters optimization.
Population-based methods generally are a series of search algorithms inspired by nature population. The core idea of population-based methods is that it view a lot of possible solutions as a population and each possible solution as individual, then information will be exchanged between individuals to create new individuals or new possible solution and eventually find out the optimal solutions. The most common population-based methods are genetic algorithms, evolutionary algorithms, evolutionary strategies and particle swarm optimization[1]. Population-based methods are also model-free methods. Compare to the random search method(both random search and random search plus), population-based methods can optimize not only the hyperparameter but also optimize the parameters like

the genetic algorithm optimizing the weights in neural network[9]. Random search methods can not optimize the parameters like this because random search sample sets of parameters before the machine learning model is created. The population-based methods focus on keeping model better and there is usually a process of exchanging information and creating new individuals, which lead to that the optimization will take a long time. However, random search methods pay attentions on how to get a good model in a short time even with some performance penalty. Also most population-based methods cannot support an ability of searching global optimal [6]while random search methods will not fall into a local optimal solution.

Bayesian optimization currently is a sate-of-the-art method of hyperparameter optimization, which recently obtains a big breakthrough in deep neural networks for image classification[4], speech recognition and neural language modeling.

Bayesian optimization is an iterative algorithm based on an acquisition function with posterior probability to do a black-box optimization by employing Gaussian processes to model the target function[14]. Compare to the random search method, Bayesian optimization will change the strategy of sampling points according to the past sampling points or experiences while random search methods don't have such a learning process. For more information about bayesian optimization please see[14].

1.2 Machine Learning Methods

In the research of this paper, mainstream machine learning algorithms will be selected as the experimental targets. These algorithms are k-nearest neighbor algorithm, k-means, support vector machine and neural network. So in this this section, this paper will generally discuss how they work before everything starts.

1.2.1 K-nearest Neighbor Algorithm

K-nearest neighbor algorithm is a basic classification and regression method. The general idea of K-NN is that for each input data, calculate the distance between it and each training data, order all training data by the distance, then find which class of data

appears most often among the top k data and finally let the input data belong to this class. As shown in the figure 1.3, this is a good example to explain how K-NN works. There are two different types of sample data, represented by small blue squares and small red triangles, and the data marked by the green circle in the middle of the picture is the data to be classified.

Step 1: The first step is normalization. Normalization is a way to map the values of numeric features in datasets to range of $(0, 1)$:

Step 2: Then calculate the distance between the point that will be classified and other points. The common ways to calculate the distance can be described as follows:

Assuming that each data x that has n numerical features is x_k=($x_k^1, x_k^2, ...x_k^n$). There are two data x_i=$(x_i^1, x_i^2, ...x_i^n)$ and x_j=$(x_j^1, x_j^2, ...x_k^n)$.

The distance $L_p(x_i^l, x_j^l) = (\sum_{l=1}^{n} |x_i^l - x_j^l|^p)^{\frac{1}{p}}$.

When $p = 1$, $L_1(x_i^l, x_j^l) = \sum_{l=1}^{n} |x_i^l - x_j^l|$, it is Manhattan.

When $p = 2$, $L_1(x_i^l, x_j^l) = \sum_{l=1}^{2} (|x_i^l - x_j^l|^2)^{\frac{1}{2}}$, it is Euclidean.

When $p = \infty$, $L_1(x_i^l, x_j^l) = \max_l |x_i^l - x_j^l|$, it is maximum distance of each coordinate.

Step 3: Choose an integer value for k and find the k points closest to the input data by the query algorithm.

Step 4: Calculate the proportion of each class in k neighboring points. Let the class with the largest proportion as the input data class.

1.2.2 K-Means

K-means is a basic but extensive unsupervised clustering algorithm. The general idea of k-means can be described as follows:

Assume we have clusters $(C_1..C_k)$ with centroids $(u_1..u_k)$.u is the mean vector of i th cluster and it can be defined as:

$$E = \sum_i^k \sum_{x \in C_i} = \|x - u_i\|_2^2$$

The best clustering strategy will be obtained when the object function E is being minimized. However, such an optimization is a NP problem, so a heuristic iterative approach is an excellent solution to it:

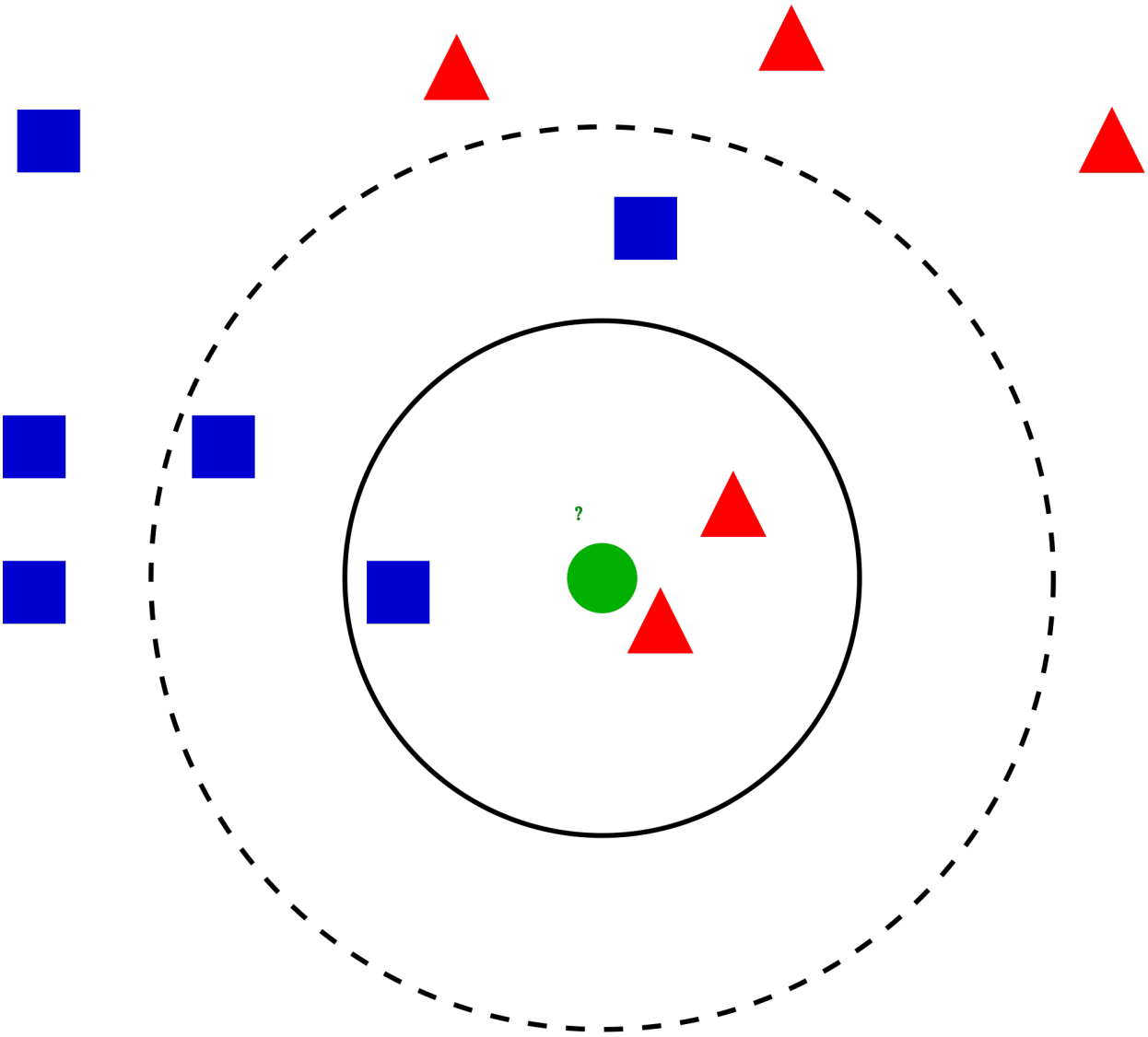


Figure 1.3: K-NN

Step1: Choose an integer value for k.

Step2: Initialize k centroids$(u_1, u_2, ..u_k)$.

Step3: For each sample in the data set, calculate the distance of it to k cluster centers and let the sample belong to the cluster with the smallest distance.

Step4: For each cluster u_j, recalculate its cluster center $\frac{1}{\|c_i|} \sum_{x \in C_i} x$.

Step5: Repeat step 3 and 4, until a certain suspension condition is reached.

1.2.3 Neural Network

Neural network is a family of algorithms that is used to do classification or recognition in a way inspired by biological brain operating. The basic neural network includes three main parts: input layer, hidden layer, output layer. There are neurons in each layer.

A neuron in neural network play an essential role of filtering invalid information and extracting valid information from the data set. Each neuron is a computational node. There is an activation function in each neuron(See Figure1.3-c), which enable input signals to be converted into some certain output signals. In each neuron, different information will get into the neuron with different weights. Weight can be viewed as the synapses in biology and it can explain how important the information from another neuron is.

The workflow of a neural network consists of two main parts: forward propagation, back propagation.

Forward propagation: In this process, the data will propagate through the hidden layer to the output layer. For each neuron in the same hidden layer, the output data is the output of activation function where independent variable is the input data. Then the input data of the next layer will be the output data of the previous layer multiplied by the corresponding weight, which is the weight of the connection between the neurons of two adjacent hidden layers. Last hidden layer will connect to the output layer. At the end, we will get the error between the output(predictive value) of output layer and true value. For more details and information, please see[18].

Backpropagation: In this process, after we get the error, we will modify the weight according to the derivative of the weight by the error, which will be obtained by chain derivation. For more details and information, please see[18].

A forward propagation and a back propagation are called a training, after many times of training, the weights and errors usually converge. After that, the neural network can make predictions from the new input data.

1.2.4 Support Vector Machine

Support vector machine narrowly speaking, is a binary classifier with the kernel model which is the linear classifier with largest interval in the feature space. In machine learning, it is a supervised learning and it generally can be viewed as a set of hyperplanes in a high- or infinite-dimensional space, which is used to solve some classification and regression problems[15].

The basic idea of support vector machine learning is to correctly divide the training data set and the separation hyperplane with the largest geometric interval[11]. As shown in the figure 1.4, the formula:

$$wx + b = 0$$

is the separating hyperplane. For a linearly separable data set, there are infinitely many such hyperplanes (ie perceptrons), but the separating hyperplane with the largest geometric interval is the only one[11]. Such the separating hyperplane is also called support vector and the core work of support vector is to find such a support vector. For more details about support vector machine please see[5].

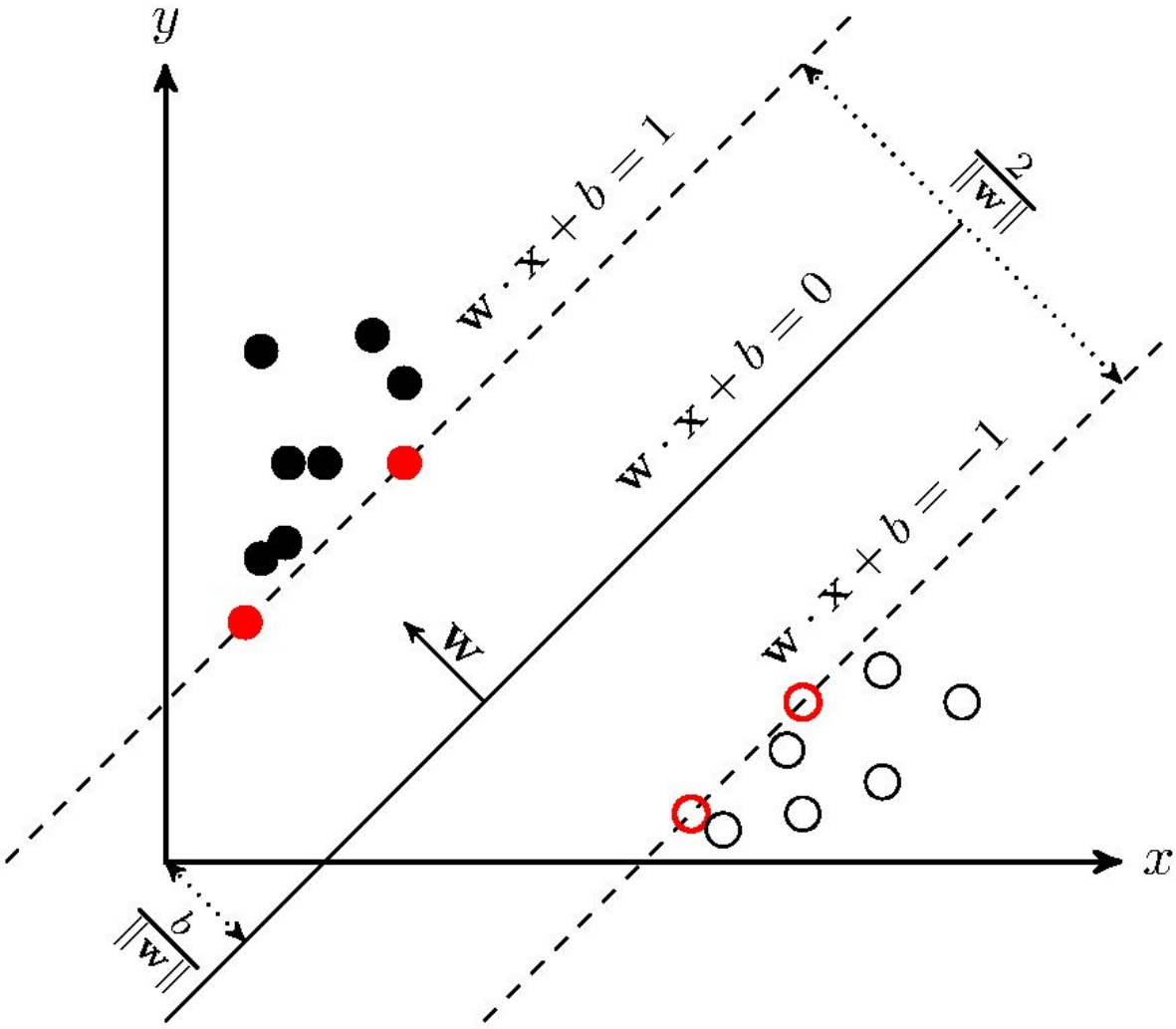


Figure 1.4: SVM

Chapter 2

Definitions and Assumptions

2.1 Overview

In this chapter, we will propose a mathematical theoretical basis for faster and more effective random search. These mathematical theories include assumptions, definitions for new concepts and mathematical models of random search plus.

2.2 Definitions

Hyperparameter vector

Each hyperparameter h_i that will be optimized has a discrete and finite range:

$$h_i : (h_i^{(0)},\, h_i^{(1)},\, h_i^{(2)}, ..h_i^{(n-1)},\, h_i^{(n)}) \quad h_i^{(0)}<h_i^{(1)}<h_i^{(2)}..h_i^{(n-1)}<h_i^{(n)} \text{ and } n \in \mathbb{N}^0$$

It also represents an ordered set.

Hyperparameter space:

A set of hyperparameter vectors:

$$\mathbb{H} : (h_1, h_2, h_3, ...h_n)$$

A point in hyperparameter space:

$$h^{(i)} = (h_1^{(k_1)},\, h_2^{(k_2)},\, h_3^{(k_3)}, ..h_n^{(k_n)})$$

$h_i^{(k)}$ means the k element in the hyperparameter vector h_i

Hyperparameter subspace:

A set of hyperparameter vectors:

$$\mathbb{S} : (s_1, s_2, s_3, ...s_n) \quad \text{For each } s_i \subseteq \mathrm{h}_i$$

Hyperparameter space separation:

Divide a entire hyperparameter space $\mathbb{H} : (h_1, h_2, h_3, ...h_n)$ into some $(\mathbb{S}^{(1)}, \mathbb{S}^{(2)}, \mathbb{S}^{(3)}, ...\mathbb{S}^{(n)})$. For each subspace $\mathbb{S}^{(i)}$has same dimension as $\mathbb{H}$:

$$\mathbb{S}^{(i)} : (s_1^{(i)}, s_2^{(i)}, s_3^{(i)}, ...s_n^{(i)})$$

For any two different subspace:

$$\mathbb{S}^{(m)} \cap \mathbb{S}^{(n)} = \emptyset \quad (m \neq n)$$

For all subspace:

$$\mathbb{S}^{(1)} \cup \mathbb{S}^{(2)} .. \mathbb{S}^{(n-1)} \cup \mathbb{S}^{(n)} = \mathbb{H}$$

Random search plus:

A random search method. It is based on hyperparameter space separation; after hyperparameter space is divided into a lot of hyperparameter subspace. In each hyperparameter subspace, we randomly sample a certain number of points. Such a method is called random search plus.

2.3 Assumptions

Assumption 1:

All data x in dataset comes from ground truth and they are discrete and finite.[2].

Assumption 2:

A machine learning model M_{h^i} is unique according to a certain hyperparameter combination h^i.

Assumption 3:

All hyperparameters are discrete and finite.

Assumption 4:

A machine learning model M_{hi} can be viewed as a functional that maps training data set $x^{(train)}$ to a function that minimizes a kind of loss function $L(x, F)$[2].

Assumption 5:

The machine learning hyperparameter optimization can be viewed as:

$$\Phi(h) \approx \underset{h \in (h^{(0)}, h^{(1)}, h^{(2)}, .. h^{(n)})}{\arg\min} L(x^{(validation)}, M_h(x^{(train)}))$$

So here, the hyperparameter optimization is to choose a good point h in hyperparameter space or find a hyperparameter combination in the set of all possible hyperparameter combinations, which can make the value of $L(x^{(validation)}, M_h(x^{(train)}))$ approximately equal to its minimum[2].

Chapter 3

Implementations and Methods

3.1 Overview

This chapter will mainly discuss the core algorithm of hyperparameter space separation and search, the theoretical basis of designing experiments, what is the object the experiment will test, what will be the comparison experiment, materials for doing experiments and details about pseudocode.

3.2 Space Separation and Search

As we discussed in Chapter 2, the core of random search plus is to divide the hyperparameter space into several subspaces and then to use random search for each of them.
Let's start with an easy example to show how it works in general.
Assume there is a 2-D hyperparameter space: $\mathbb{H} : (h_1, h_2)$ and there are N_1, N_2 elements in h_1, h_2. Obviously, in the two-dimensional case, if we get the value of the diagonal, then we can easily know the range of points that will be randomly generated. At the same time, the entire two-dimensional space can be regarded as a rectangle, this large rectangle can be composed of several small and identical rectangles which are similar to the large rectangle. Therefore, if we can traverse all such small rectangles, then we can reach any part of the two-dimensional hyperparameter space, which also means that we can sample everywhere of this 2-D space.

In order to better traverse every part of the space, we need to define a unit hyperparameter subspace first:

Because of similarity, the length of sides c_1, c_2 of the unit hyperparameter subspace must be:

$$\frac{N_1}{c_1} = \frac{N_2}{c_2} = g,\ g \text{ is the common divisor for } N_1 \text{ and } N_2$$

So the unit hyperparameter subspace $\mathbb{S}^{unit}$can be:

$$\mathbb{S}^{unit} : (h_1, h_2)$$

And for h_1, h_2:

$$h_1 = (h_1^{(0)}, h_1^{(1)}, ..h_1^{(c_1)})$$
$$h_2 = (h_2^{(0)}, h_2^{(1)}, ..h_2^{(c_1)})$$

(See figure 3.1 a)

Once unit hyperparameter subspace is determined, we can move it by adding and subtracting the parameter vector, and then eventually traverse all the hyperparameter subspaces.

The general process is as follows:

Step 1: Determine the unit hyperparameter subspace: $\mathbb{S}^{unit} : ((h_1^{(0)}, h_1^{(c_1)}), (h_2^{(0)}, h_2^{(c_1)}))$.

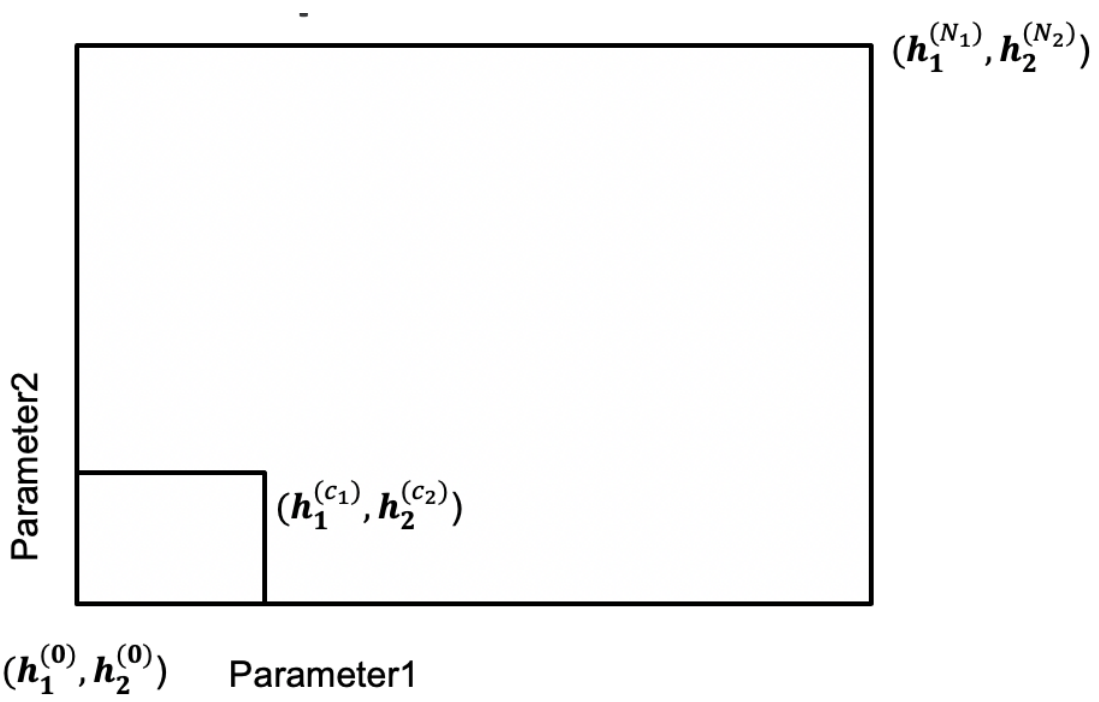
Step 2: For each dimension or hyperparameter vector of the unit hyperparameter subspace $\mathbb{S}^{unit}$, add the constant c_i into index of all elements in this until the index of this largest one equal to the N_i:

$$h_i = (h_i^{(0)}, h_i^{(c_1)}) \quad \rightharpoonup \quad (\mathrm{h}_i^{(N_1 - c_1)}, h_i^{(N_1)}) \text{ (See figure 3.1 b)}$$

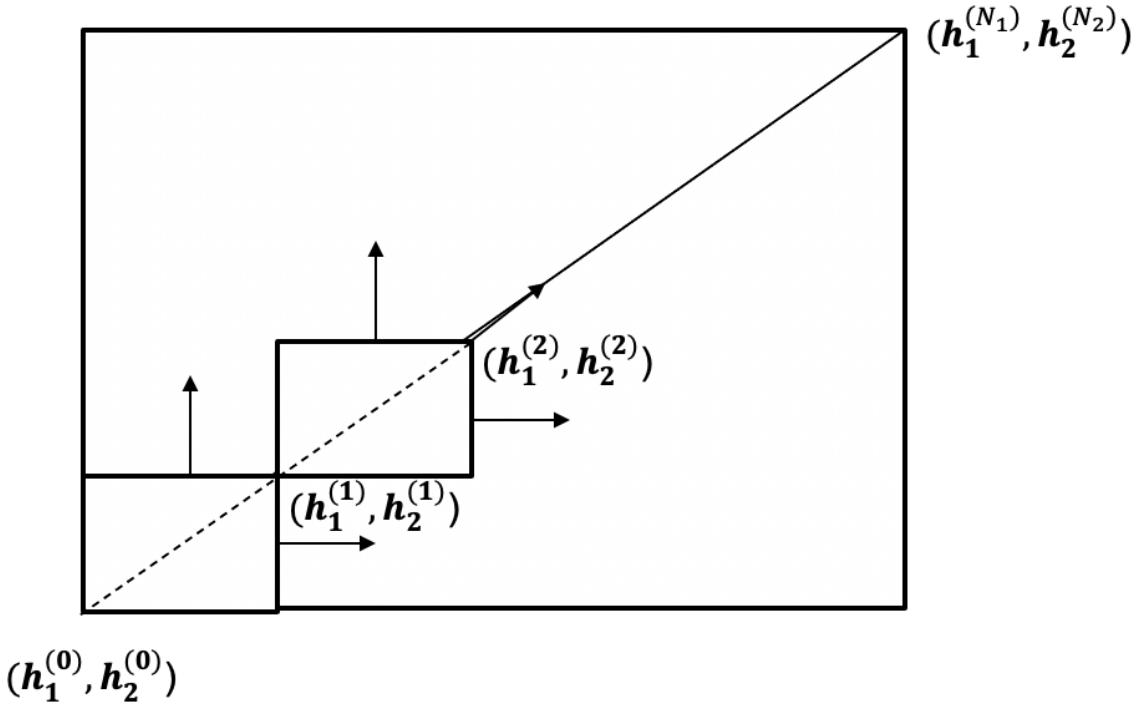
Step 3: After that, move the hyperparameter unit subspace to next hyperparameter subspace follow the diagonal and repeat step 2 until the the hyperparameter unit subspace move into the hyperparameter subspace at the end of diagonal then every subspace will be traveled. (See figure 3.1 b)

3-D case: When the dimension of hyperparameters space is three, it is similar to the 2-D case. The hyperparameters space will be $\mathbb{H} : (h_1, h_2, h_3)$ and there are N_1, N_2, N_3 elements in h_1, h_2, h_3. The hyperparameter unit subspace is a cube:

$$\frac{N_1}{c_1} = \frac{N_2}{c_2} = \frac{N_3}{c_3} = g,\ g \text{ is the common divisor for } N_1,\ N_2 \text{ and } N_3$$



(a) 2-D space



(b) 2-D space searching

Figure 3.1: 2-D Case

So the unit hyperparameter subspace $\mathbb{S}^{unit}$ can be:

$$\mathbb{S}^{unit} : (h_1, h_2, h_3)$$

And for h_1, h_2, h_3:

$$h_1 = (h_1^{(0)}, h_1^{(c_1)})$$
$$h_2 = (h_2^{(0)}, h_2^{(c_1)})$$
$$h_3 = (h_3^{(0)}, h_3^{(c_1)})$$

(See figure 3.2)

Travel to every hyperparameter subspace:

Step 1: For the first dimension to the last dimension (for $i = 1$ to 3) do:

$$h_i = (h_i^{(0)}, h_i^{(c_i)}) \rightharpoonup \quad (\mathrm{h}_i^{(N_i - c_i)}, h_i^{(N_i)})$$

Step 2: Move unit hyperparameter subspace to next hyperparameter subspace at diagonal:

$$\mathbb{S}^{unit} \rightharpoonup \quad \mathrm{S}_j^{diagonal}$$

Step 3: Repeat step 1, step 2 until:

$$\mathbb{S}^{unit} \rightharpoonup \quad \mathrm{S}_g^{diagonal}$$

(The number of hyperparameter subspace at diagonal is g).

3.3 Core Algorithm

The above conclusions can be generalized to any dimensional hyperparameter space. Assume there is n dimension hyperparamter space $\mathbb{H} : (h_1, h_2, h_3, ...h_n)$, $(N_1, N_2, ..N_n)$ is the elements' number in each hyperparameter vector h_i.

1. Find the greatest common divisor g of the $(N_1, N_2, ..N_n)$ and the factors of $g(g_1, g_2, ...g_k)$ and choose one of them like g_k.

2. Hyperparameter unit subspace:

$$\frac{(N_1, N_2, ..N_n)}{g_k} = (c_1, c_2, c_3, ..c_n)$$
$$\mathbb{S}^u = (h_1, h_2, ...h_n)$$
$$h_i = (0, c_i)$$

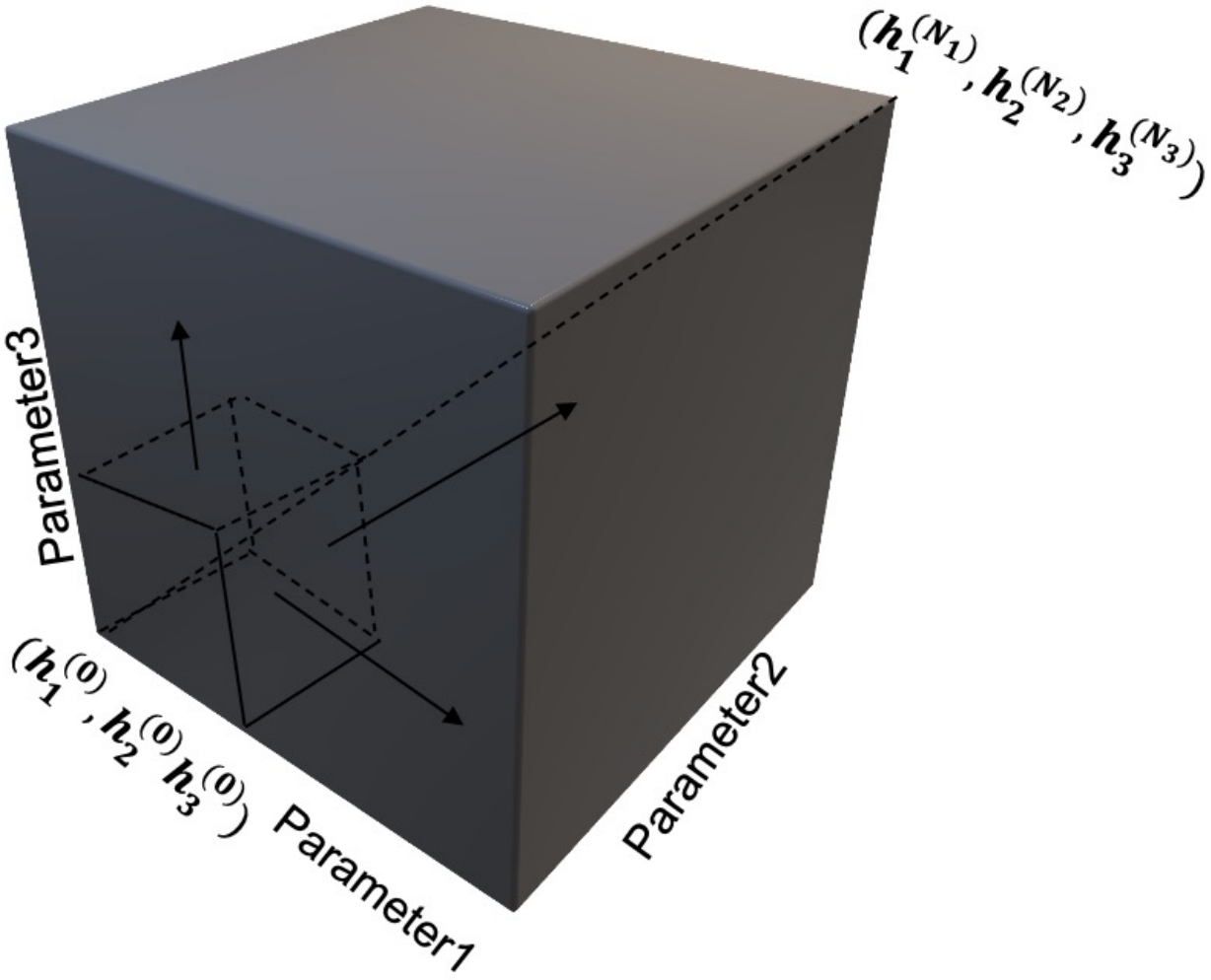


Figure 3.2: 3-D Case

3. Initialize an empty list variable: Map=[ ]

4. Define a move function, which is used to save all hyperparameter subspaces (See figure 3.3 a).

5. For each hyperparameter subspace stored in Map, create a machine learning method with a set of hyperparameter which is obtained by randomly sampling in the hyperparameter subspace stored in map (See figure 3.3 b).

6. A list Score = [] to store each model's accuracy or other evaluation indexes.

7. Return the best one and its parameters.

As is shown in the pseudocode, there is a hyperparameter for random search plus. That is g_k because there could be a lot of choices for g from $g(g_1, g_2, ...g_k)$. So before random search plus works, it should be given a value of k for telling random search plus how to divide the search space.

3.4 Extreme Cases Discussion

Case 1: All hyperparameters' numbers are relatively prime. If the random search plus meet a set hyperparameters with their numbers are relatively prime. For example, the numbers of n hyperparameters are (1, 2, 3, 5, 7,..), the solution is to add 1 for each odd number to make the relatively prime numbers' set to be a even numbers' set (2, 2, 4, 6, 8,...) then the hyperparameter space can be separated, but the index of the number added before, let the algorithm just jump this hyperparameter subspace. For example:

$$(1, 2, 3, 5) \rightharpoonup (2, 2, 4, 6)$$

hyperparameter subspace is:

$$((0, 1), (0, 1), (0, 2), (0, 3)), ((1, 2), (0, 1), (0, 2), (0, 3)), ((0, 1), (2, 4), (0, 2), (0, 3))...$$

Algorithm will skip $((1, 2), (0, 1), (0, 1), (0, 1))$ because the 2 is 1+1 which is an invalid hyperparameter subspace or subregion. It will just go to the next one $((0, 1), (1, 2), (0, 1), (0, 1))$.

Case 2: Millions of hyperparameters. Assume n hyperparameters $(h_1, h_2, ...h_n)$ and the number of choice for each h_i are $(N_1, N_2, ...N_n)$. As it is discussed in extreme case 1, there are a least one common factor 2 for these numbers. Then the number of hyperparameter

Algorithm 1 Move

Input: $\mathbb{S}^u(h_1, h_2...h_n)$;
Output: Map

Global g
if $g > 0$ **then**
    for $j = 1$ to n **do**
        $S = \mathbb{S}^u$
        for $i = 1$ to g **do**
            $h_j = h_j + ic_i$
            $h_j = h_j - uc_i$
            Map.append($\mathbb{S}^u$)
        end for
        $\mathbb{S}^u = S$
    end for
    Map.append($\mathbb{S}^u$)
    for $j = 1$ to n **do**
        $h_j = h_j + ic_i$
        $h_j = h_j - uc_i$
    end for
    $g = g - 1$
    Move($\mathbb{S}^u$)
end if
return Map

(a) Move

Algorithm 2 Model_Score

Input: Map
Output: Score

for $i = 0$ to $g^2 - 1$ **do**
    Parameters = Random(Map[i])
    Model = Create_Model(Machine_learning_method, Parameters)
    Model.fit()
    Score.Append(Model.Validation())
end for
return Score

(b) Model Validation

Figure 3.3: Core Algorithms

subspaces or subregions will be:

$$2^n$$

Compared to random search plus, the total trials of grid search is $N_1 N_2 ... N_n$. It is easy to know that any $N_i \geq 2$ or separation would not happened. In this case:

$$2^n \leq N_1 N_2 ... N_n$$

So whatever n is, random search plus is at least faster than grid search. The reduced numbers of searching will be:

$$\begin{gathered} 2^n \leq (2 \times c_1) N_2 ... N_n \\ 2^n \leq (2 \times c_1)(2 \times c_2) ... N_n \\ ..... \\ 2^n \leq (2 \times c_1)(2 \times c_2) ... (2 \times c_n) \\ 2^n \leq 2^n \times c_1 c_2 c_4 .. c_n \end{gathered}$$

When $c_1 = c_2 = c_4 .. c_n = 1$, random search plus will equal to grid search. However, if there is at least one $c_i > 1$, the random search plus will save the total search number at least:

$$2^{n+1} - 2^n = 2^n$$

Because c_i is a integer. The save number of times of searching will be also larger with the increasing dimension. But random search plus also have to face to the problem of high dimension because of dimension increasing.

Finally, compared to random search, whether random search plus is better or not is still uncertain. Because it is not clear if the random search needs to sample more with dimension increasing. If so, probably the random plus will behavior better as with same samples, random plus has more chance of getting a good point, but for more rigorous inference, there need to be more discussions in future.

3.5 Experiment Descriptions

3.5.1 General information

Experiment subjects:

Random search, Random search plus, Random search_cv, Grid search

Optimized machine learning method:

K-NN, K-means, Neural network and Support vector machine.

Platform:

Scikit-learn

Optimized Hyperparameters:

See (Figure 3.3 a)

Datasets:

See (Figure 3.3 b)

Max Iterations: For k-means, neural network and support vector machine, the max iteration of training process will be 300. Once the training iteration is over 300, they will stop the training.

Experiment 1:

Compare random search to random search plus. In this experiment, random search plus with its different parameters k will be compared with random search. All methods have same trials of randomly samplings. The goal of this experiment is to compare the expectation value of accuracy or silhouette coefficient with same trials to verify the hypothesis and proofs which are discussed on the chapter 2.

Experiment 2:

Compare random search_cv to random search plus.

In this experiment, random search plus with its different parameters k will be compared to random search_cv from scikit-learn. They are all random-search based but with different ways of sampling. Random search_cv will use the default numbers of sampling(10 each run) while the numbers of sampling of random search plus for each run will depends on the different parameters k.

Machine learning methods	K-NN	K-means	Neural network	Support vector machine
Hyperparameters(optimized)	N neighbors(1,30) Leaf size(1,30)	N_clusters(2,30) N_init(1,30)	Hidden layers(1,30) Neurons(1,20) Learning rate(0,1)	Penalty C: $(10^{-10},10^{10})$ Gamma: $(10^{-10},10^{10})$
Hyperparameters(default)	Weights = 'uniform' P= 2（euclidean metric）	Initial = 'K-means ++' Distance = 'euclidean metric'	Activation function ='relu' Solver = 'adam'	Kernal function='rbf'

(a) Hyperparameter Settings

Datasets Name	Classes	Features	Samples
Iris flower	3	5	150
Pima indians diabetes	2	9	768
MNIST hand-written	10	748	600000

(b) Datasets

Figure 3.4: Hyperparameter Settings and Datasets

Extra experiment:

Since when the k increases, random search plus will reduce to a grid search, so in this experiment, there will be a comparison about random search plus with all values of k to the grid search by using neural network with 3 hyperparameters(See figure 3.4-a) as optimized object under three different size of datasets(See figure 3.4-b).

3.5.2 Experiment 1

The random search and random search plus have different way of sampling, which lead to that each run of random search would produce different trials to random search plus. Also, due to the different parameter k, random search plus will also have different trials for same times of running. A trial means randomly sampling once. Specifically speaking, for the random search, every run means a trial or randomly sampling once. However, each run for random search plus means randomly sampling every subspace once, so the total trials of each run for random search will depends on the numbers of the subspace it create. Obviously as it is discussed before, the numbers of the subspace of a random search plus will be decided by the parameter k which can tells us what g is chosen. So to keep same trials, we need to know the relationship between parameter k and the g.

The relationship between parameter k and the g in experiment 1:

Assume there are n choices of $g : (g_1, g_2, ...g_{n-1}, g_n)$, and $k : (1, 2, 3, 4)$

$$g = g_1 \text{ when } k = 1.$$
$$g = g_2 \text{ when } k = 2.$$
$$g = g_3 \text{ when } k = 3.$$
$$g = g_4 \text{ when } k = 4.$$

Because the set $(g_1, g_2, ...g_{n-1}, g_n)$ is a set of the factors of g, when $g_1 = 1$ and the subspace will equal to the entire space in random search plus. In this situation, random search plus is equal to random search, so actually the random search is provide by a random search plus with $k = 1$ but it is theoretically equivalent to a random search.

Experiment one is to compare the objects with the same number of trials of random search, which also means the random search and random search plus with different parameters will

be control to have same numbers of samplings.

To make random search, random plus with different parameters have same trials, assume $k = 2, k = 3, k = 4$ and $g : (g_1, g_2, ...g_{n-1}, g_n)$, and there will be the dimension of hyperparameters space is m, so trials $T = lcm(g_1^m, g_2^m, g_3^m)$.

Random search will run T times, return T results.

Random search plus with $k = 2$ will run $\frac{T}{g_1^m}$ times and return $\frac{T}{g_1^m}$ results.

Random search plus with $k = 3$ will run $\frac{T}{g_2^m}$ times, then return $\frac{T}{g_2^m}$ results.

Random search plus with $k = 4$ will run $\frac{T}{g_3^m}$ times, then return $\frac{T}{g_3^m}$ results.

Example:

2-D Dimension hyperparameter space:$([1, 2, 3..30], [1, 2, 3, ..30])$

$g : (1, 2, 5, 10, 15, 30)$

$$g = 1 \text{ when } k = 1.\text{(random search)}$$

$$g = 2 \text{ when } k = 2.$$

$$g = 5 \text{ when } k = 3.$$

$$g = 10 \text{ when } k = 4.$$

$$T = lcm(2^2, 5^2, 10^2) = 100$$

Random search will run 100 times, return 100 results.

Random search plus with $k = 2$ will run 25 times and return 25 results.

Random search plus with $k = 3$ will run 4 times and return 4 results.

Random search plus with $k = 4$ will run 1 times and return 1 results.

There will be 4 models to do a comparison. They all have same trials(100), so the runtime of them should be almost same.

3.5.3 Experiment 2

Experiment two is to compare random search_cv to random search plus with different parameter k and to see which one has higher efficiency of sampling when all of them run the same number of times.

Because random search_cv only samples 10 times per run, in this experiment I try to use a low sample number, so the relationship between k and g is different.

In order to reduce the number of samples, common factor g should be as small as possible.

The relationship between k and g in experiment 2:

$$\text{Assume there is } n \text{ choices of } g : (g_1, g_2 ... g_{n-1}, g_n), \text{ and } k : (1, 2, 3, 4)$$

$$g = g_1 \text{ when } k = 1.$$

$$g = g_2 \text{ when } k = 2.$$

$$g = g_3 \text{ when } k = 3.$$

$$g = g_4 \text{ when } k = 4.$$

Random search_cv will run T times, sample 10 return T results.

Random search plus with $k = 2$ will run T times, sample g_2^m and return T results.

Random search plus with k=3 will run T, sample g_3^m times,then return T results.

Random search plus with k=4 will run T, sample g_4^m times, then return T results.

Example:

2-D Dimension hyperparameters space:$([1, 2, 3..30], [1, 2, 3, ..30])$

$g : (1, 2, 5, 10, 15, 30)$

$$g = 1 \text{ when } k = 1.(\text{Random search\_cv})$$

$$g = 2 \text{ when } k = 2.$$

$$g = 5 \text{ when } k = 3.$$

$$g = 10 \text{ when } k = 4.$$

$$T \text{ will be chosen and in the experiment 2, it is 20}$$

Random search will run 20 times, sample 200 times, return 20 results.

Random search plus with $k = 2$ will run 20 times,sample 80 times, and return 20 results.

Random search plus with $k = 3$ will run 20 times,sample 180 times and return 20 results.

Random search plus with $k = 4$ will run 20 times,sample 320 times and return 20 results.

There will be 4 models to do a comparison.They all have different sampling(trials), so the runtime of them could be different.

3.5.4 Extra Experiment:

In this experiment, random search plus with all k's values will be compared to the grid search. The optimized object will be three different hyperparameters from neural network.

The experiment will test the accuracy, runtime and best parameters searched by running random search plus with different k and grid search once. The relationship between k and g is same as it in experiment 1 and 2.

Chapter 4

Results

4.1 Overview

In this chapter, we will show the results of different machine learning algorithms‘ hyperparameter optimizations from the two experiments and one extra experiment. In experiment 1, we will discuss and compare the case of random search plus with different k and random search in the same number of samples and to see which sample has a high mean value of accuracy or silhouette coefficient, which also means that which sample has a high expected value. In experiment 2, we will discuss and compare the case of random search plus with different k and random search_cv in the same number of runs and to see which algorithm can find the better answer with shorter time . In the extra experiment, we will discuss and compare the case of the random search plus with different k and grid search in one run and to see which one is better and what difference between them.

4.2 Experiment 1

4.2.1 K-NN

The hyperparameter space(search space) of K-NN is:

$$\text{N\_neigbors: } (1-30)$$

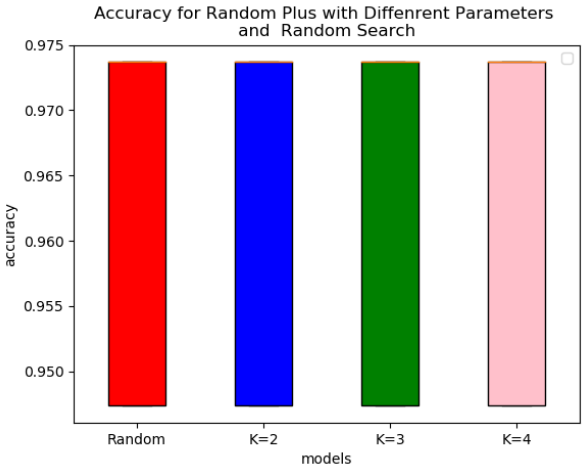
$$\text{Leaf\_size: } (1-30)$$

So the common factor of the maximum value of these two hyperparameters is $g = [1, 2, 5, 6, 10, 15, 30]$, and there are six different strategies to divide the hyperparameter space, which can let the numbers of hyperparameter subspace to be $[1, 4, 25, 36, 100, 225, 900]$.
In experiment 1 results' pictures of K-NN, random search means $g = 1$, there are only 1 hyperparameter subspace that is also the hyperparameter space, and $k = 2$ means $g = 2$, there are 4 hyperparameter subspaces in hyperparameter space, and $k = 3$ means $g = 5$, there are 25 hyperparameter subspaces in hyperparameter space, and $k = 4$ means $g = 10$, there are 100 hyperparameter subspaces in hyperparameter space. Therefore, I set 100 trials(samples) for each method. Random search will run 100 times, and random search plus with $k = 2, 3, 4$ will $25, 9, 16$ times.

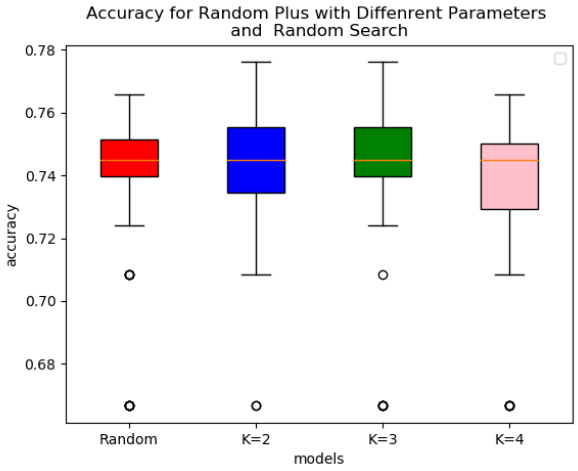
Accuracy: For all methods, they have same accuracy distributions(See figure 4.1). The median accuracy of 100 samples for all methods are almost same. The best case of 100 samples for all methods are different but such difference is slight which could be view as a reasonable error. What's more, the average accuracy for all methods are also same. So I believe that in this case, comparing to the random search, random search plus doesn't improve the accuracy for K-NN in its hyperparameter optimization.

Runtime: All methods sample same points so the runtime of them should be same or similar. Considering that the difference of runtime are varied in three different datasets(See figure 4.3), I believe it is a reasonable error that is caused by Leaf_size because this hyperparameter actually does influence K-NN's runtime.

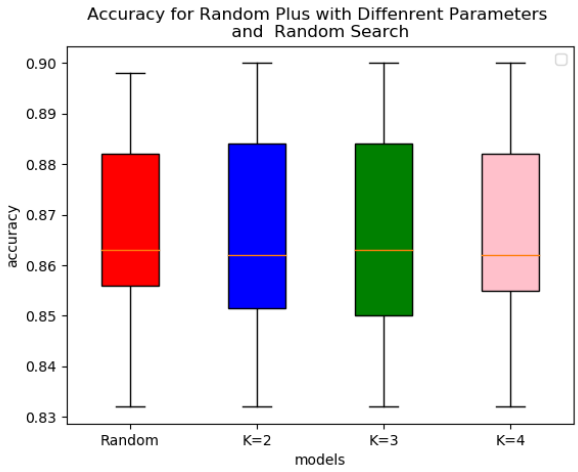
Returned hyperparameters: For random search, it will give all 100 hyperparameters searched in the figure 4.4 because it sample 100 points each run but for others, they will return the best answer for each run. In the chapter 2, we all know that, random search plus' samples will never be same in each run however many points are repeated in random search's 100 samples. Many points repeated in random search plus with different k but these points come from different run not a same run, which means no matter how much times searching whole space, the position of some best points in a certain space separation will not be changed. That is why they can be found again and again.



(a) Iris Flower

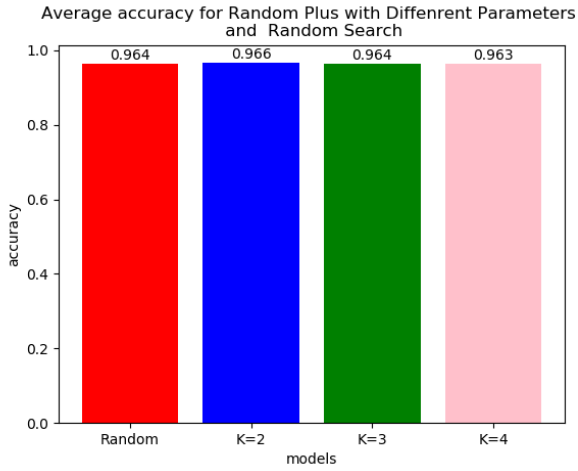


(b) Pima Indians Diabetes

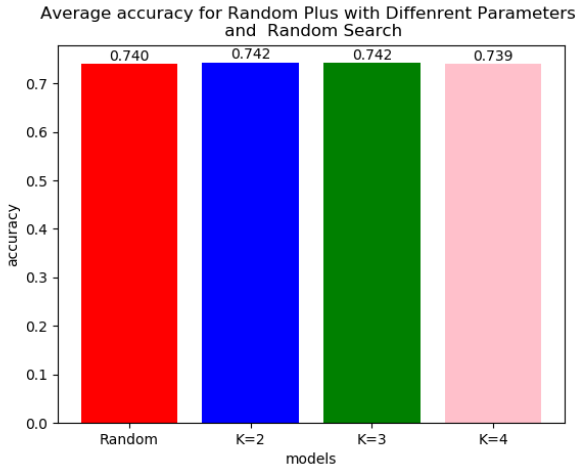


(c) MNIST Hand-written

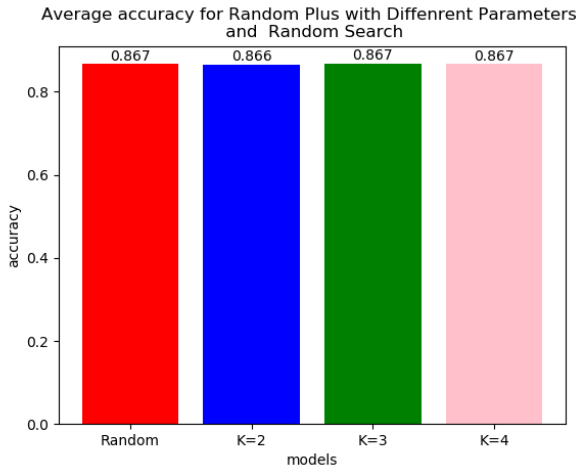
Figure 4.1: Accuracy Distributions for K-NN in Experiment 1



(a) Iris Flower

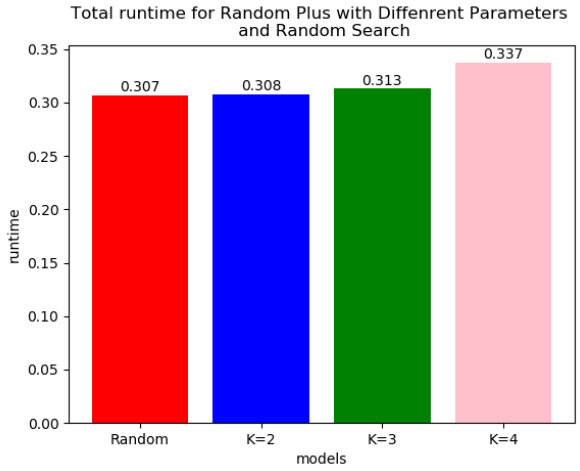


(b) Pima Indians Diabetes

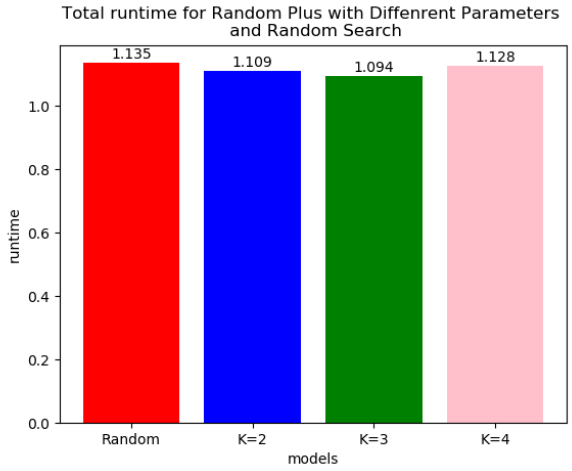


(c) MNIST Hand-written

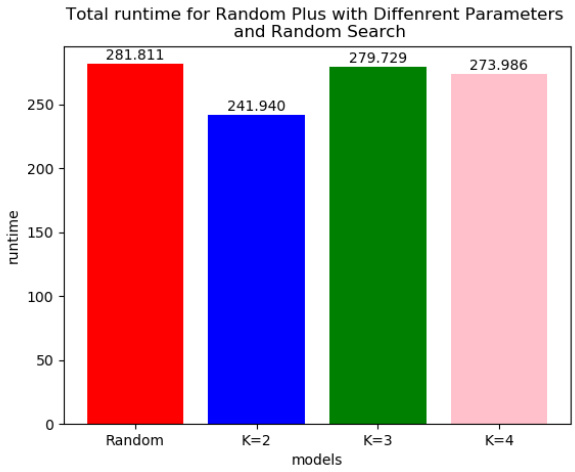
Figure 4.2: Average Accuracy for K-NN in Experiment 1



(a) Iris Flower

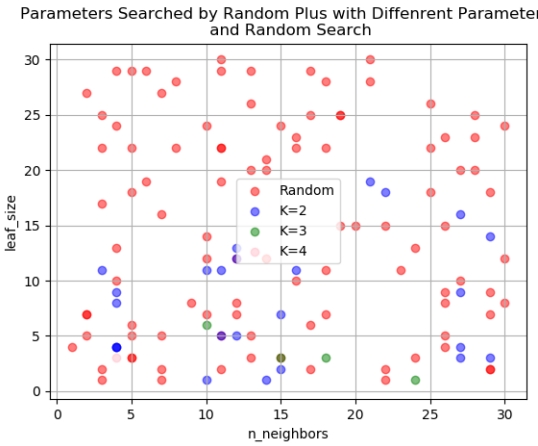


(b) Pima Indians Diabetes

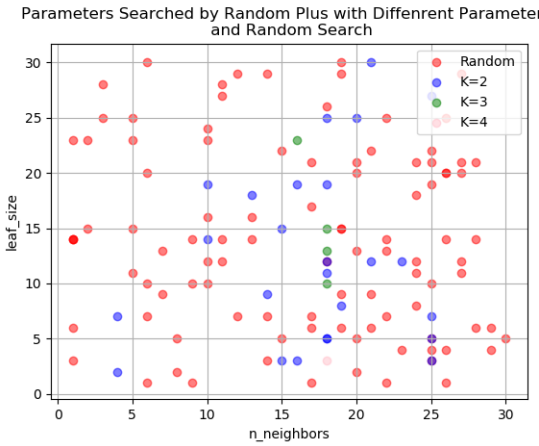


(c) MNIST Hand-written

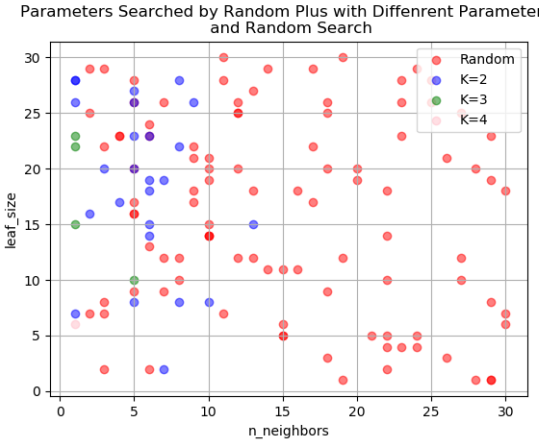
Figure 4.3: Total Runtime for 100 Trials



(a) Iris Flower



(b) Pima Indians Diabetes



(c) MNIST Hand-written

Figure 4.4: Hyperparameters Returned for Each Run of Each Method in K-NN's Hyperparameter Space

4.2.2 K-Means

The hyperparameter space(search space) of K-Means is:

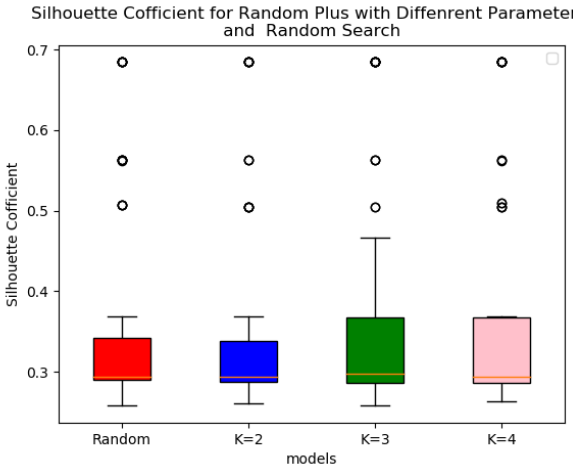
$$\text{N\_clusters: } (2-30)$$
$$\text{N\_init: } (1-30)$$

$g = [1, 2, 5, 6, 10, 15, 30]$ so the setup and relationship between k and g are same to them in K-NN.

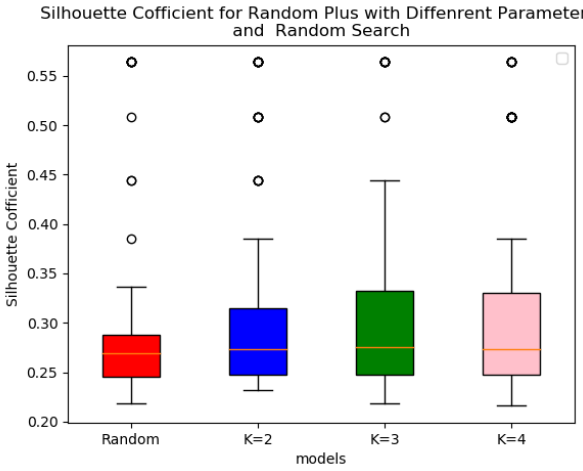
Silhouette Coefficient: These results are same as in K-NN. For all methods, they have same silhouette coefficient distributions(See figure 4.5). The median silhouette coefficient of 100 samples for all methods are almost same. The best case of 100 samples for all methods are different but such difference is more obvious than in K-NN. In the first two datasets, when random search plus with $k = 3$, it looks to find out more good points or optimal values, however, comparing to the results in average value, I believe such an improvement is very slight and it is necessary to reduce the trials or samples so that we can make sure such an improvement is real or just an accident. Because with enough samples, random search can have a huge chance of finding a optimal value while less samples' influence on random search plus will be smaller as the random samples are more evenly distributed on the search space(hyperparameter space).

Runtime: All methods sample same points so the runtime of them should be same or similar. However, here, the total runtime of random plus' methods are all longer than random search's. I don't believe it is a accident because in k-means, n_init heavily influence on the speed of training process and the way of random search plus to sample can explore all range of n_init with same probability, which lead the average runtime of random search plus is longer than random search.

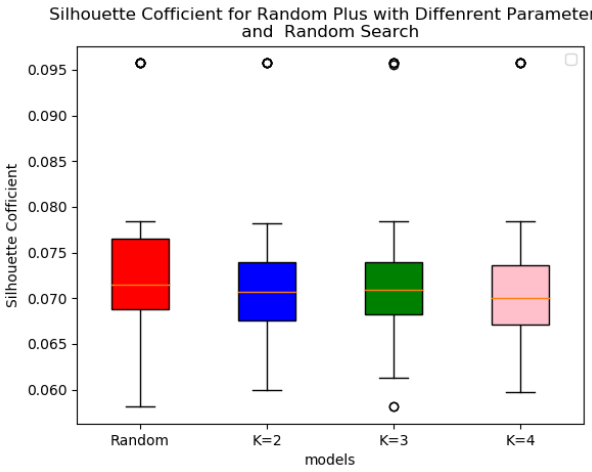
Returned hyperparameters: It is the same results to those in K-NN. Random search's returned parameters show that it almost explore each parts of search space but it is not evenly and a lot points are repeated. For random search plus, those returned parameters are best for each run, so it shows that if the optimized models work, the good points will be found on a certain subarea in search space(See figure 4.8), which also happens on K-NN's, even almost all hyperparameters exert almost same influence on the model's performance.



(a) Iris Flower

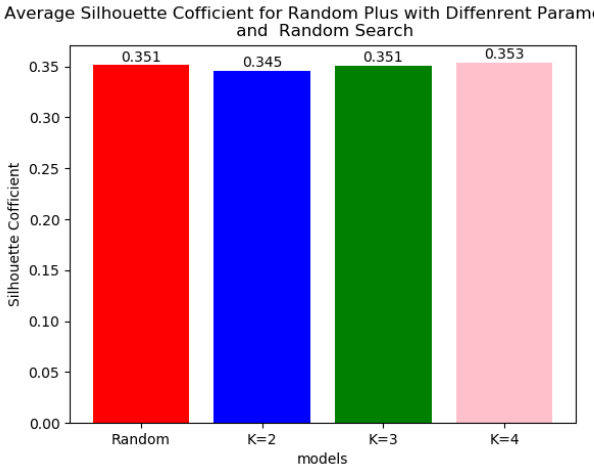


(b) Pima Indians Diabetes

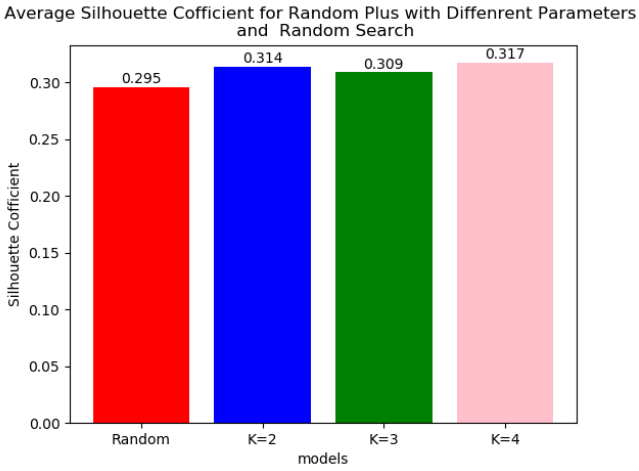


(c) MNIST Hand-written

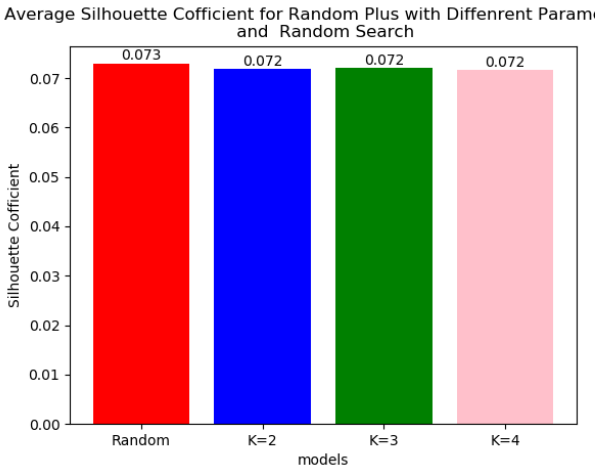
Figure 4.5: Silhouette Coefficient Distributions for K-means in Experiment 1



(a) Iris Flower

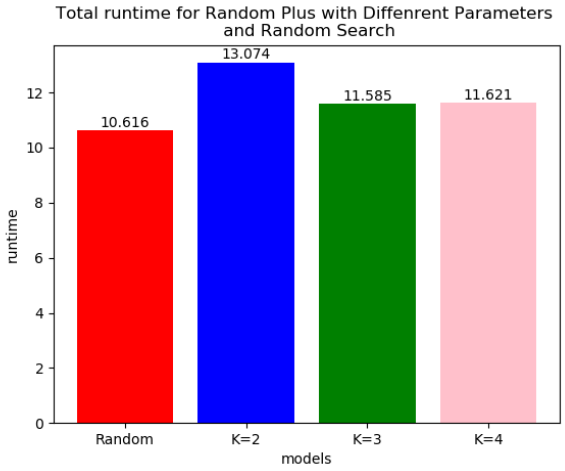


(b) Pima Indians Diabetes

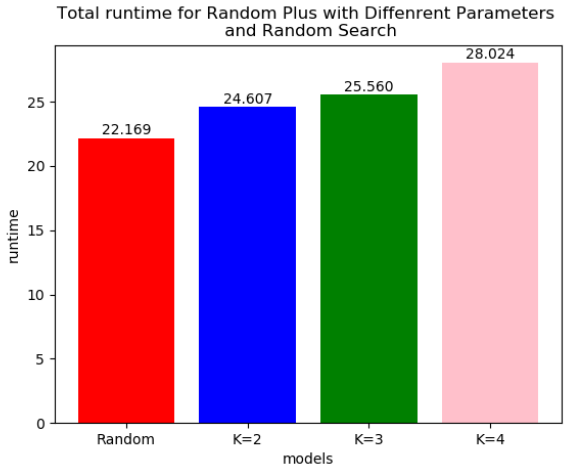


(c) MNIST Hand-written

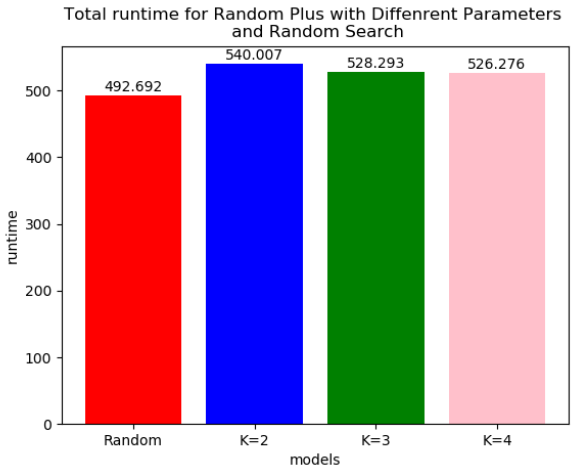
Figure 4.6: Average Silhouette Coefficient for K-means in Experiment 1



(a) Iris Flower

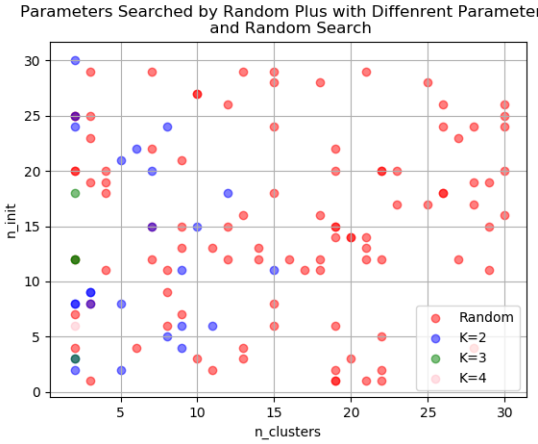


(b) Pima Indians Diabetes

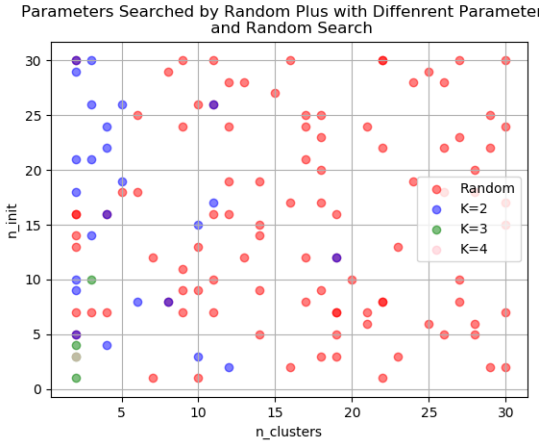


(c) MNIST Hand-written

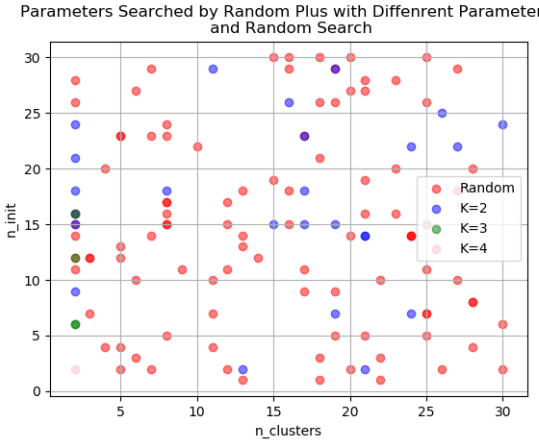
Figure 4.7: Total Runtime for 100 Trials for K-means in Experiment 1



(a) Iris Flower



(b) Pima Indians Diabetes



(c) MNIST Hand-written

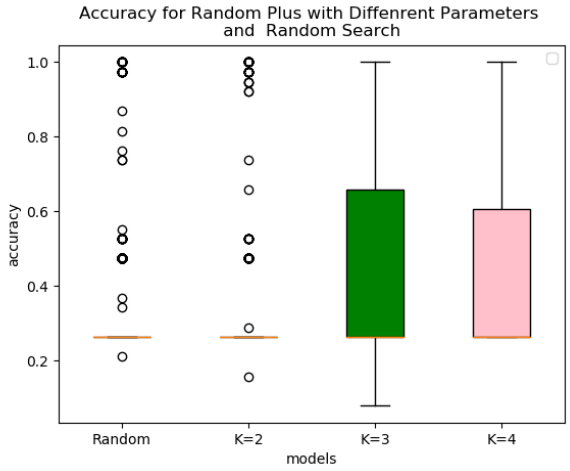
Figure 4.8: Hyperparameters Returned for Each Run of Each Method in K-means Hyperparameter Space

4.2.3 Neural Network

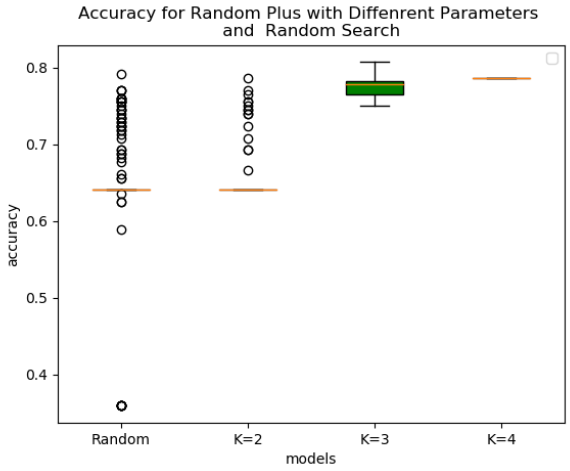
The hyperparameter space(search space) of neural network is:

$$\text{Hidden\_layers: } (1-30),\ \text{Neurons: } (1-20),\ \text{Learning\_rates: } (0,1)$$

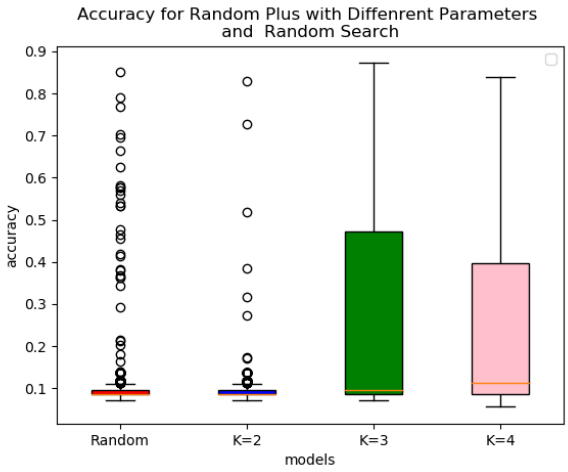
Accuracy: Now, the results of accuracy distributions are different to it in the first two study cases. For random search plus with $k = 3$ and $k = 4$, they have more points that are good than random search and random search plus with $k = 2$, even the median value of them are almost same in the iris flower and MNIST datasets (See figure 4.9). Typically, such an improvement in average accuracy is more obvious. For the three different datasets, random search with $k = 3$ and $k = 4$ increase the average accuracy of 1000 trials(models) by 20% to 50%(See figure 4.10). Such an obvious improvement with a huge numbers of samples is hard to be believed that it is just a accident. This improvement actually means that the expected value of 1000 samples in random plus with $k = 3, 4$ is higher than the random search and random plus with $k = 2$. The later two also means that degree of space separation is very low or 0, so I believe the space separation exerts an essential influence on improving the probability of finding good points, which also practically proves the hypothesis in chapter 2.
Runtime: All methods sample same points so the total runtime of them should be same or similar. However, here, the difference of total runtime among the random search plus varies with different datasets(See figure 4.10). It can be accident, because all methods I test here are random method and the learning rate can be randomly picked up by a log number, which means there are a lot chances that a method take a lot of low learning rates in one dataset but few low learning rates in another one.
Returned hyperparameters: Even in a 3-D search space, with enough sampling random search will cover almost all points in hyperparameter space(See figure 4.12). However, as it is discussed in above 2 sections, there are still a lot of points that are repeated. Such points even increase in a 3-D search space, which means that there are more useless samples in random search with dimension increasing. That could also be a reason why the expectation of accuracy of random search is lower than random search plus. In other words, those repeated samples could be bad points but also expensive.



(a) Iris Flower

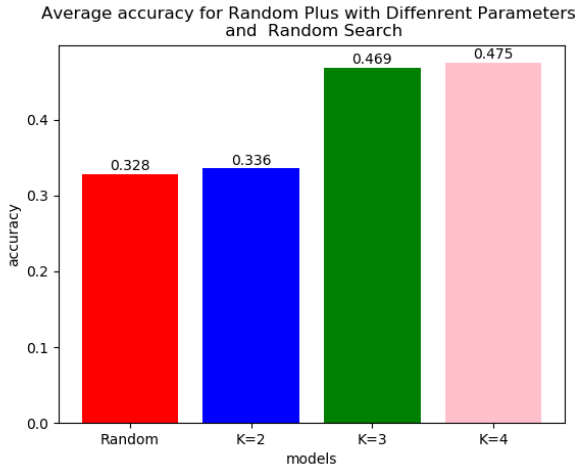


(b) Pima Indians Diabetes

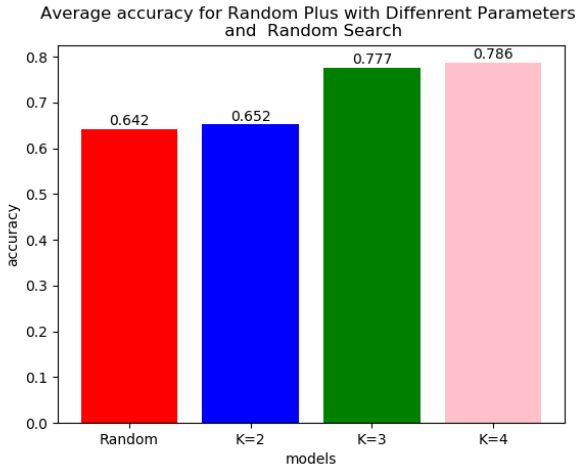


(c) MNIST Hand-written

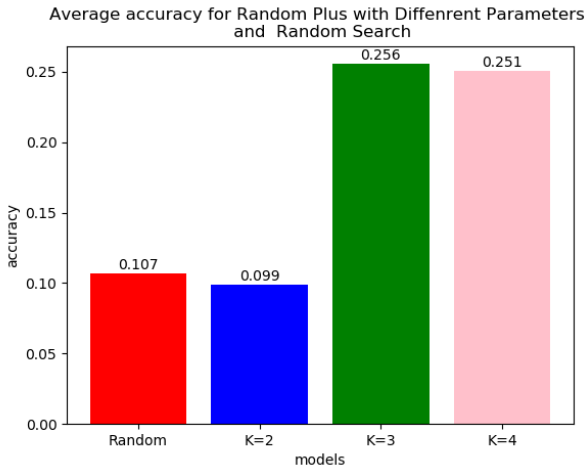
Figure 4.9: Accuracy Distributions for Neural Network in Experiment 1



(a) Iris Flower

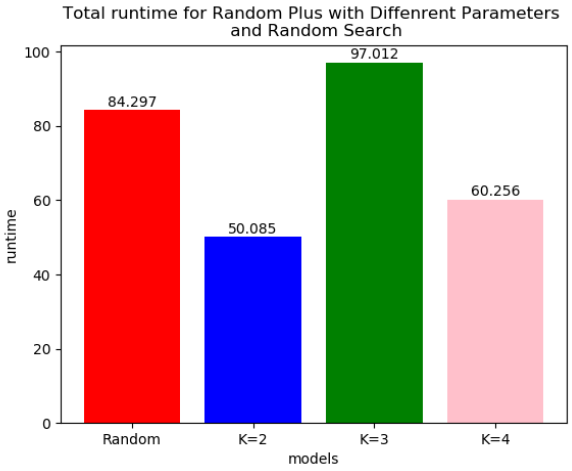


(b) Pima Indians Diabetes

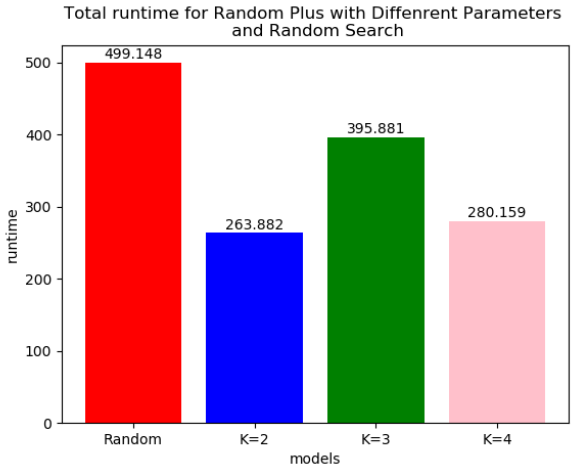


(c) MNIST Hand-written

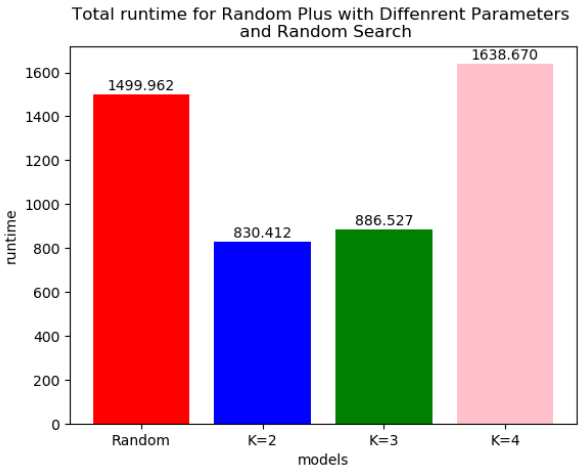
Figure 4.10: Average Accuracy for Neural Network in Experiment 1



(a) Iris Flower

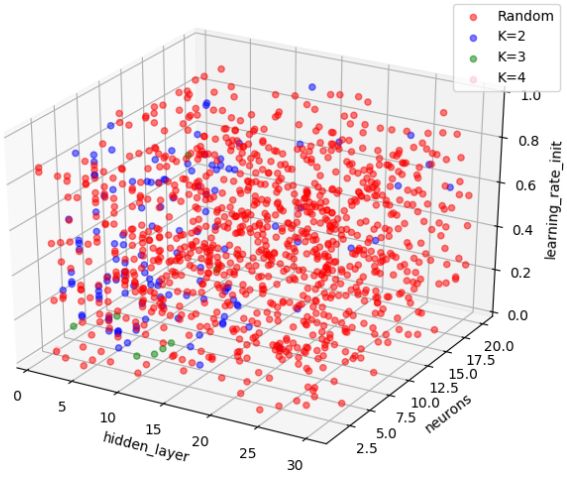


(b) Pima Indians Diabetes

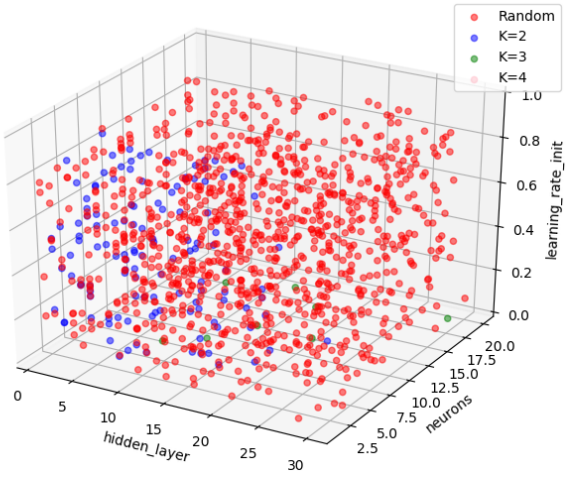


(c) MNIST Hand-written

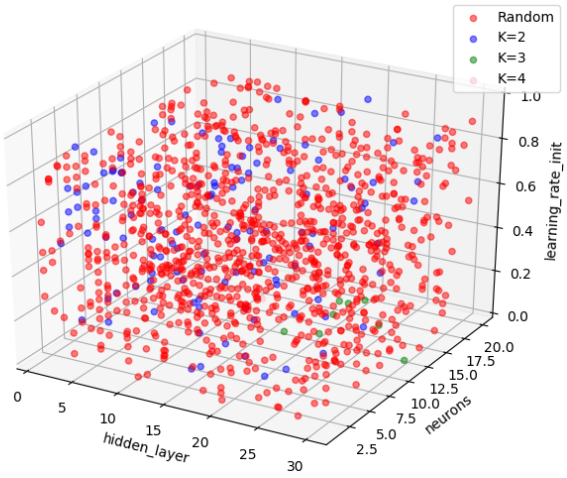
Figure 4.11: Total Runtime for 1000 Trials for Neural Network in Experiment 1



(a) Iris Flower



(b) Pima Indians Diabetes



(c) MNIST Hand-written

Figure 4.12: Hyperparameters Returned for Each Run of Each Method in Neural Network Hyperparameter Space

4.2.4 Support Vector Machine

The hyperparameter space(search space) of virtual support machine is:

$$\text{Penalty C: } (10^{-10}, 10^{10})$$
$$\text{Gamma: } (10^{-10}, 10^{10})$$

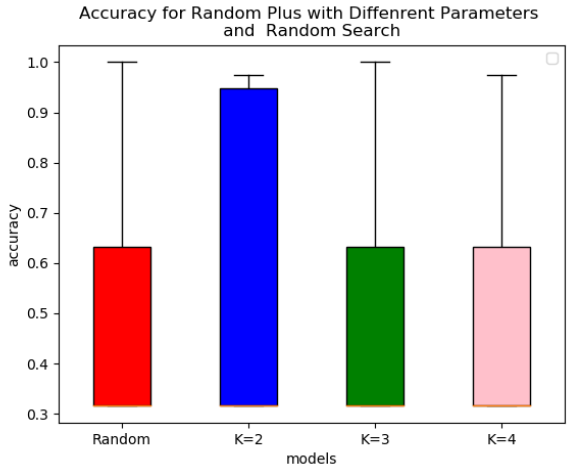
So the common factor of the maximum value of these two hyperparameters is $g = [1, 2, 5, 10, 20]$, and there are four different strategies to divide the hyperparameter space, which can let the numbers of hyperparameter subspace to be $[1, 4, 25, 100]$. Because the hyperparameter space(search space's dimension is also same to K-NN and K-means), the setup of this case is same to K-NN and K-means'.

In experiment 1 results' pictures of SVM, random search means $g = 1$, there is only 1 hyperparameter subspace that is also the hyperparameter space, and $k = 2$ means $g = 2$, there are 4 hyperparameter subspaces in hyperparameter space, and $k = 3$ means $g = 5$, there are 25 hyperparameter subspaces in hyperparameter space, and $k = 4$ means $g = 10$, there are 100 hyperparameter subspaces in hyperparameter space. Therefore, I set 100 trials(samples) for each method. Random search will run 100 times, and random search with $k = 2, 3, 4$ will run $25, 4, 1$ times.

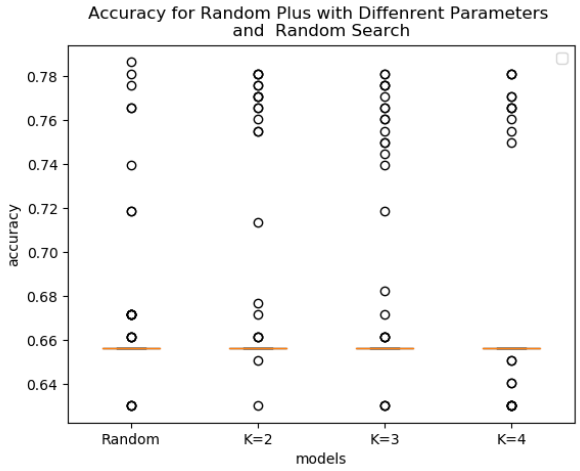
Accuracy: These results are similar to K-NN and K-means'(See figure 4.10, 4.11). As was discussed before, random search plus doesn't work well in this case. Personally speaking, not an obvious improvement is found in these three cases, which is probably because that in the landscape of objective function about the parameters and accuracy or silhouette coefficient; there are too many points that have the same or close value.

Runtime: Same analysis to those in K-NN, K-means. The total runtime of random search in Pima indians diabetes can be ignored as it does not show up again in other datasets(See figure 4.15). Only gamma influence the runtime of each trial but it looks very slight.

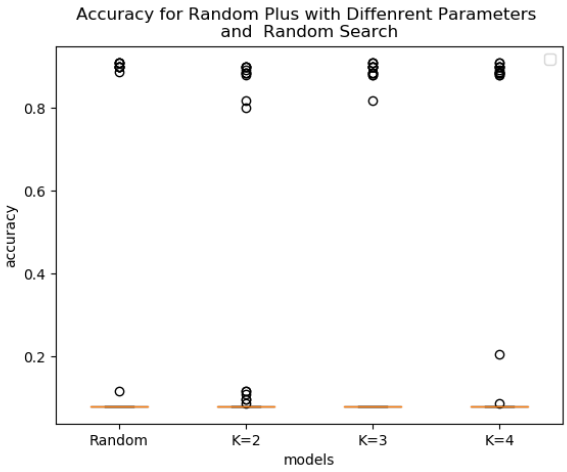
Returned hyperparameters: Same analysis to those in K-NN, K-means. Random search's return parameters shows that it almost explore each part of search space but it is not evenly distributed and a lot of points are repeated. For random search plus, those returned parameters are best for each run, which shows that if the optimized models work, the good points will be found on a certain subarea in search space(See figure 4.16).



(a) Iris Flower

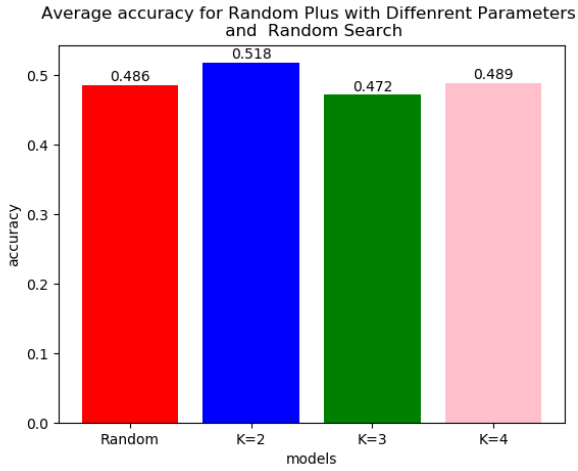


(b) Pima Indians Diabetes

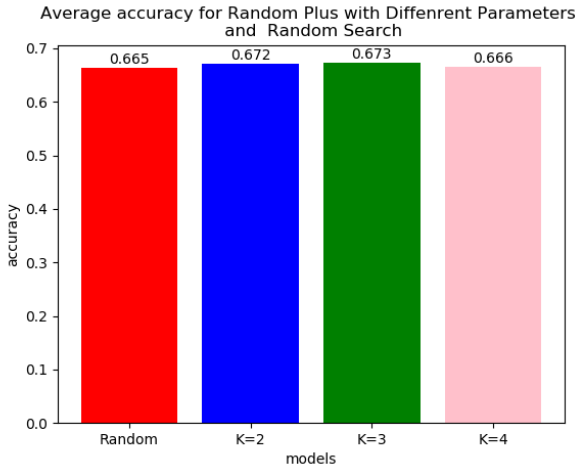


(c) MNIST Hand-written

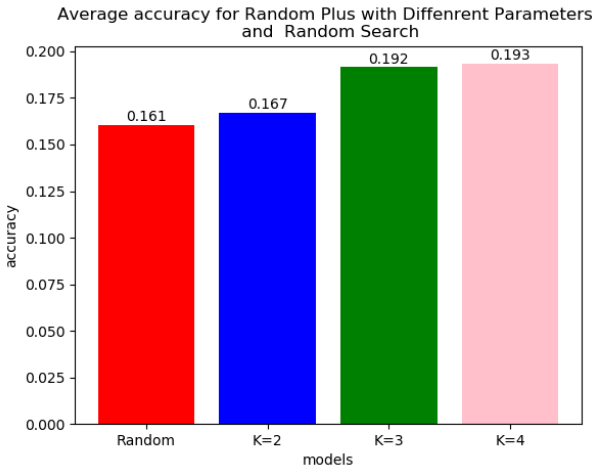
Figure 4.13: Accuracy Distributions for SVM in Experiment 1



(a) Iris Flower

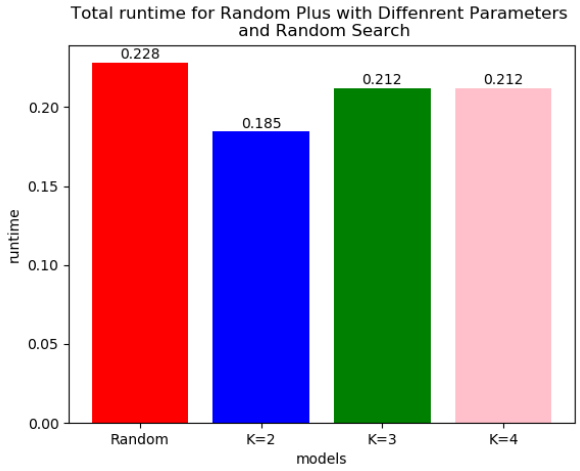


(b) Pima Indians Diabetes

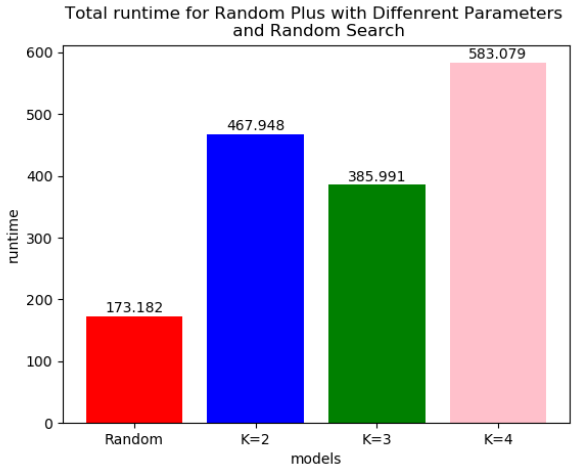


(c) MNIST Hand-written

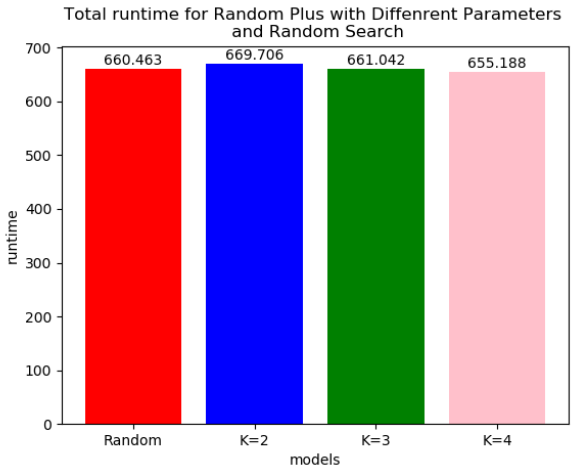
Figure 4.14: Average Accuracy for SVM in Experiment 1



(a) Iris Flower

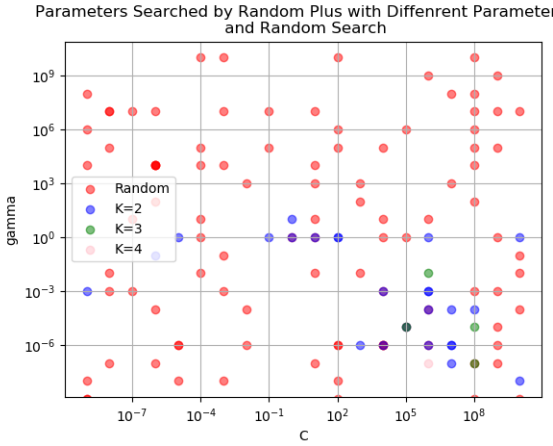


(b) Pima Indians Diabetes

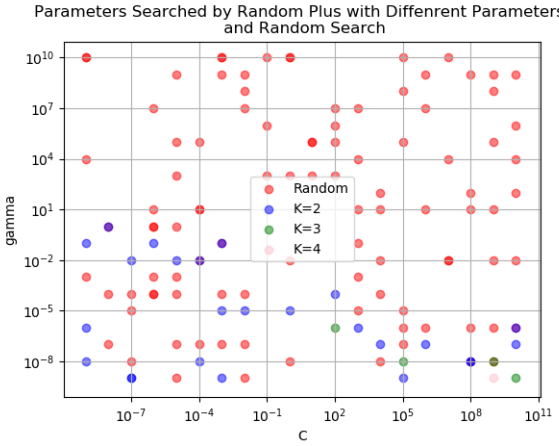


(c) MNIST Hand-written

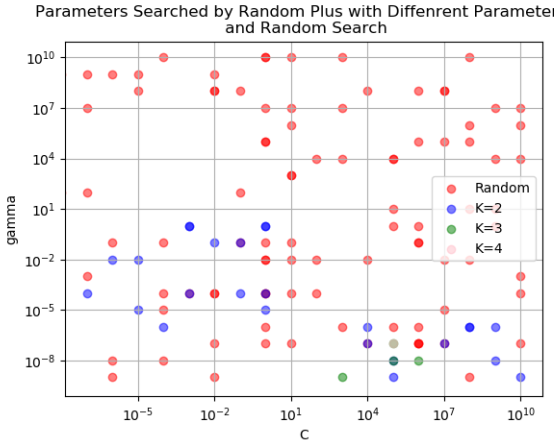
Figure 4.15: Total Runtime for 100 Trials for SVM in Experiment 1



(a) Iris Flower



(b) Pima Indians Diabetes



(c) MNIST Hand-written

Figure 4.16: Hyperparameters Searched for Each Run of Each Method in for SVM in Experiment 1

4.3 Experiment 2

4.3.1 K-NN

Same range of g and also same the relationship between k and g in experiment 1.

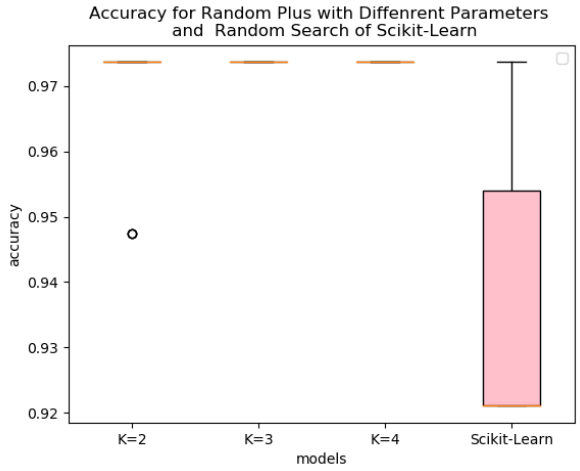
What should be mentioned here is that in the experiment, all methods will run same times but for different methods the numbers of samples per each run time will be also different. The total number of samples will not be kept same.

Each run is labeled as an experiment in the results' picture. So in experiment two, the accuracy is the best accuracy for each run with different numbers of samples. That is, when $k = 2$, $g = 2$, the numbers of subspace in K-NN case will be 4 because of 2 hyperparameters, and each run with $k = 2$ will sample 4 points and return a best one of four(For more information please see experiment description in chapter 3).

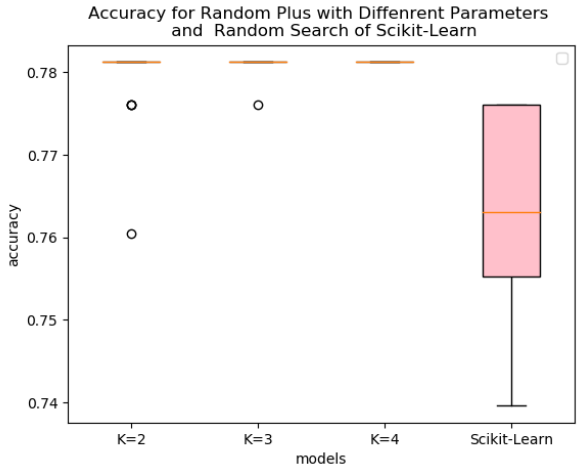
Accuracy: Variance of accuracy of 20 times of running random search plus are all smaller than the random search's from scikit-learn(See figure 4.17). Also the average accuracy of 20 times of running of random search plus are all better than scikit-learn even though it is slight(See figure 4.18). But I believe there is little improvement because it is hard to repeat such an accident(if it is) in three different datasets.

Runtime: The sample numbers of random search from scikit-learn is 10 per run while for random search plus with $k = 2$, $k = 3$, $k = 4$, the numbers are $4, 9, 16$. As it is shown in figure 4.19, only when the dataset is very huge the runtime per run of scikit-learn's is shorter than the random search plus with $k = 4$. However, scikit-learn's sample less than most random search plus methods even some sample more than scikit-learn's, which shows that the efficiency of sampling way of random search is lowest.

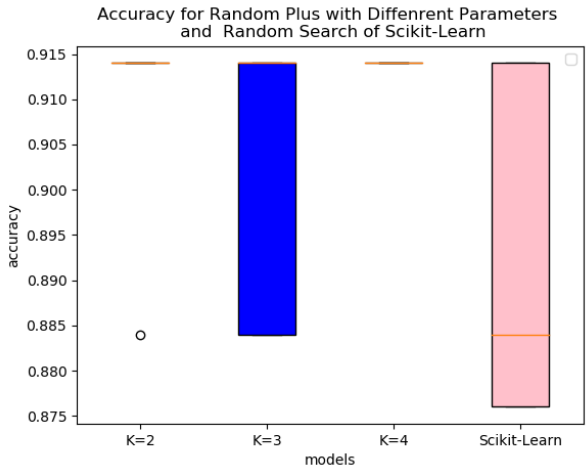
Best parameters: The best points from samples of each run is shown in figure 4.20. These best or good parameters are scattered in everywhere. It can prove that for K-NN, most parameters are good and would not heavily influence on the performance of K-NN, because in K-NN, there are not a very clear or real process of training.



(a) Iris Flower

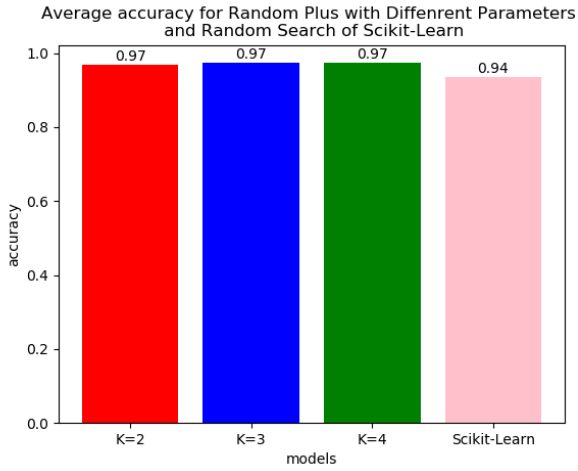


(b) Pima Indians Diabetes

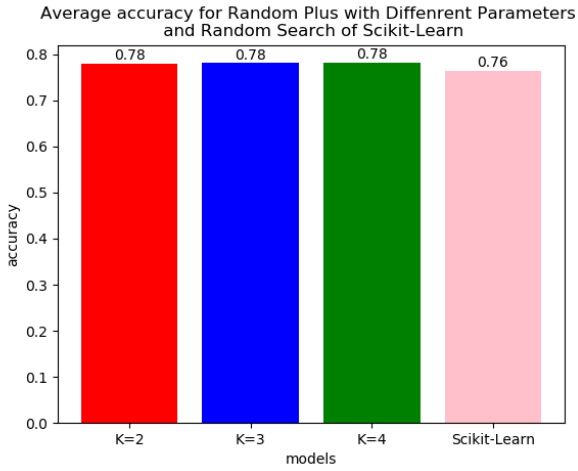


(c) MNIST Hand-written

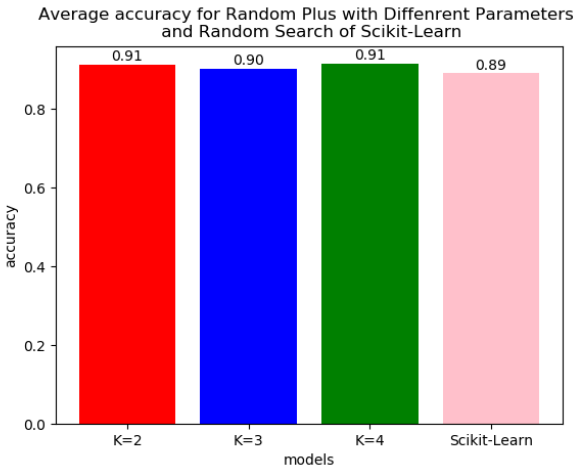
Figure 4.17: Accuracy Distributions for K-NN in Experiment 2



(a) Iris Flower

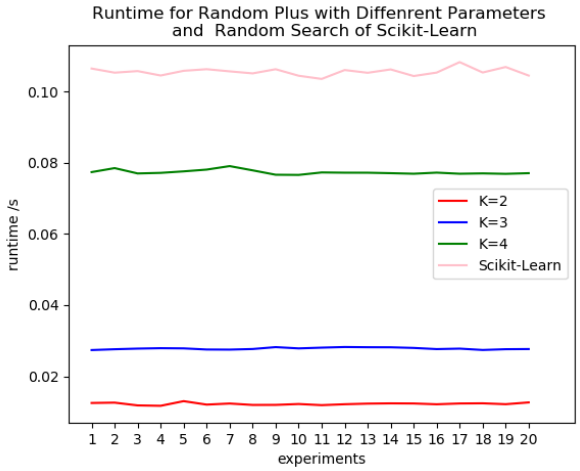


(b) Pima Indians Diabetes

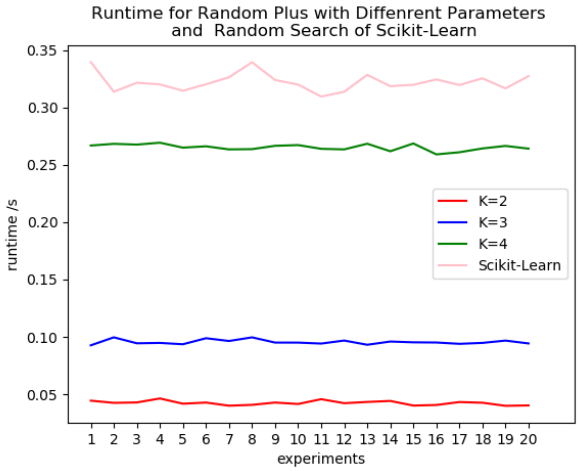


(c) MNIST Hand-written

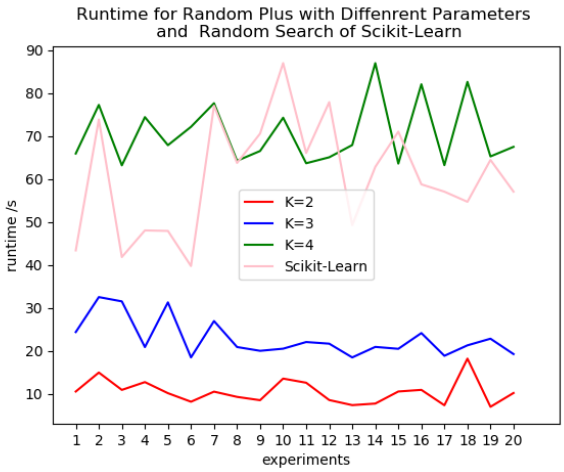
Figure 4.18: Average Accuracy for K-NN in Experiment 2



(a) Iris Flower

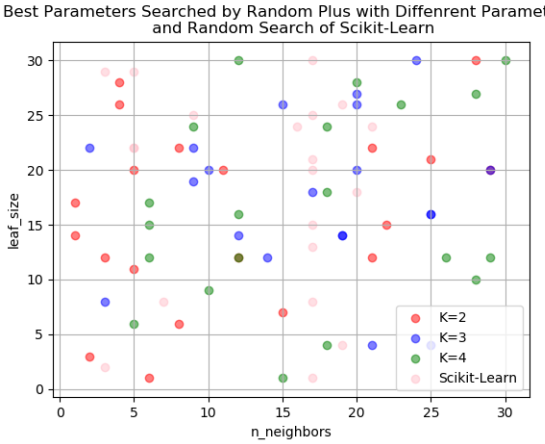


(b) Pima Indians Diabetes

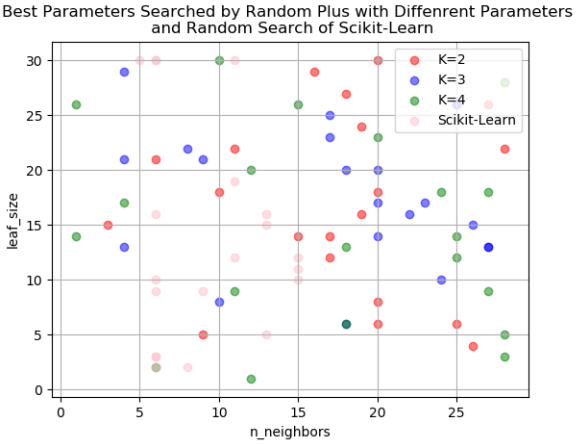


(c) MNIST Hand-written

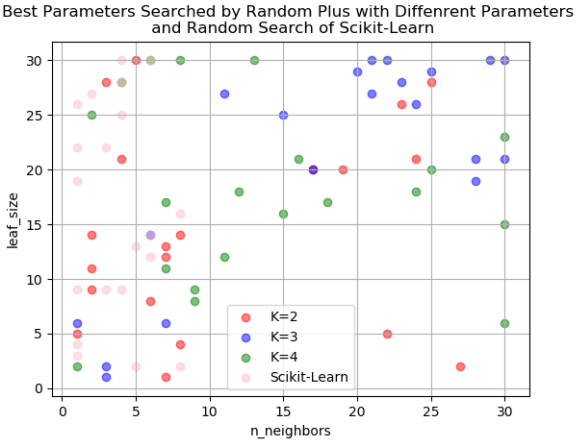
Figure 4.19: Runtime for Each Run for K-NN in Experiment 2



(a) Iris Flower



(b) Pima Indians Diabetes



(c) MNIST Hand-written

Figure 4.20: Best Parameters Returned for Each Run for K-NN in Experiment 2

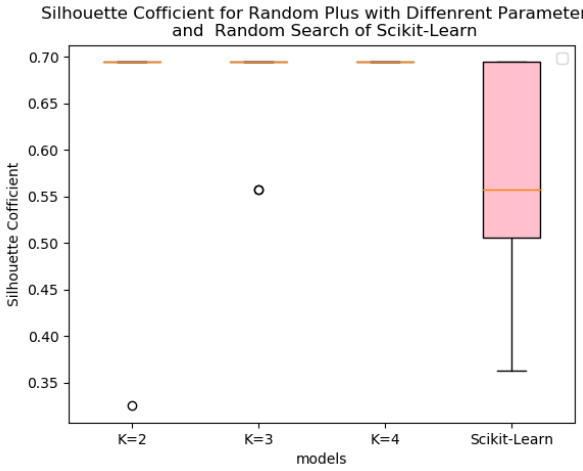
4.3.2 K-Means

Same setup to K-NN in experiment 2 but test the silhouette coefficient rather than accuracy. Each run is labeled as an experiment in the results' pictures. So in experiment two, the silhouette coefficient is the best silhouette coefficient for each run with different numbers of samples. That is, when $k = 2$, $g = 2$, the numbers of subspace in neural network case will be 8 because of 2 hyperparameters, and each run with $k = 2$ will sample 4 points and return a best one of four(For more information please see experiment description in chapter 3).

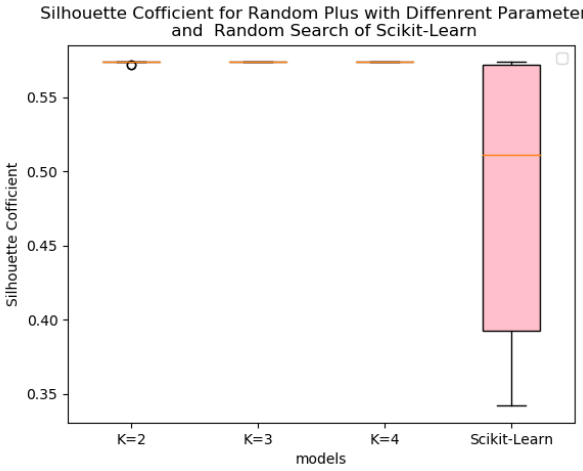
Silhouette coefficient: Variance of silhouette coefficient of 20 times of running random search plus are all smaller than the random search's from scikit-learn(See figure 4.21). Also the average silhouette coefficient of 20 times of running of random search plus are all better than scikit-learn's , and these improvements are very obvious(See figure 4.22). The improvement for average silhouette coefficient is about 20% - 30% compared to the scikit-learn's random search. However even in the MNIST dataset, all models don't work, random search plus can still return the best in the worse cases.

Runtime: The sample numbers of random search from scikit-learn is 10 per run while for random search plus with $k = 2$, $k = 3$, $k = 4$, the numbers are $4, 9, 16$. As it is shown in figure 4.23, this time, the most numbers of samples' way, random search plus with $k = 4$, is slowest one, and the $k = 2$ is faster than scikit-learn's. It makes sense because $k = 4$ does have the most numbers of samples for each run, and $k = 2$ does have less samples than scikit-learn's. However, the $k = 3$ can still run faster than scikit-learn's. According to the results from silhouette coefficient, all values of k make the random search plus get better models than scikit-learn's, so it means that in this case, random search can find out better points with less samples and also shorter runtime.

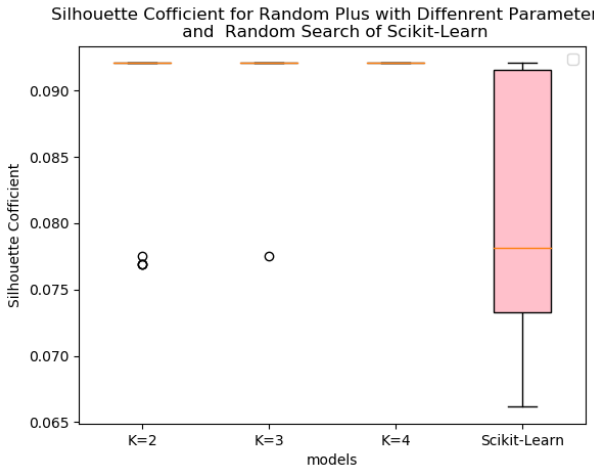
Best parameters: When the parameters found by all methods can make model work well, the best answers or parameters given by random search and random search plus has different distributions in the search space(See figure 4.24). Whether they really have different best points distributions in K-means or not should be proven by more experiments with more different datasets.



(a) Iris Flower

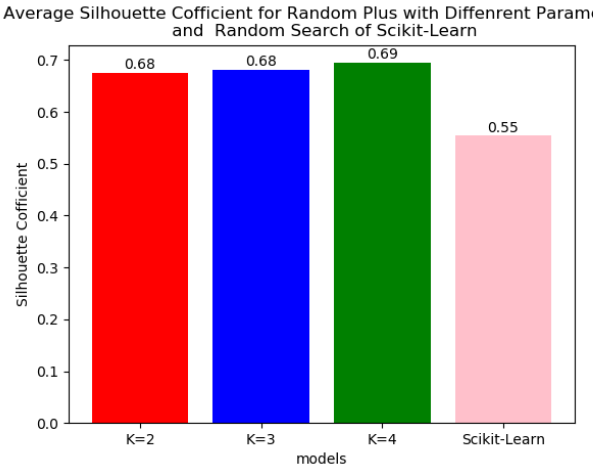


(b) Pima Indians Diabetes

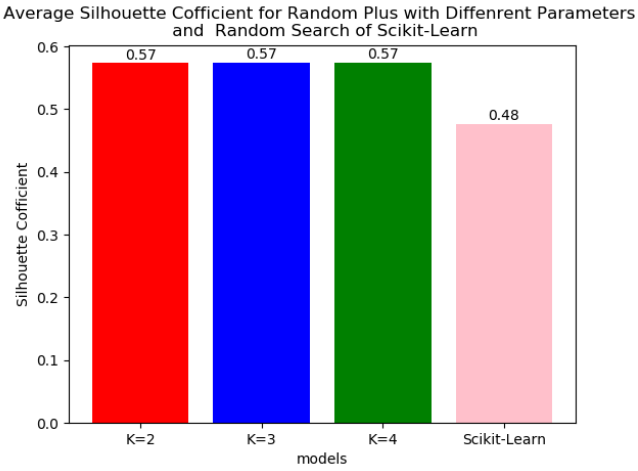


(c) MNIST Hand-written

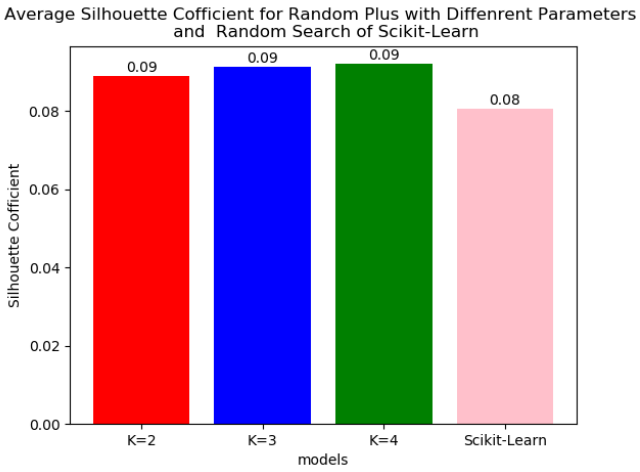
Figure 4.21: Silhouette Coefficient Distributions for K-means in Experiment 2



(a) Iris Flower

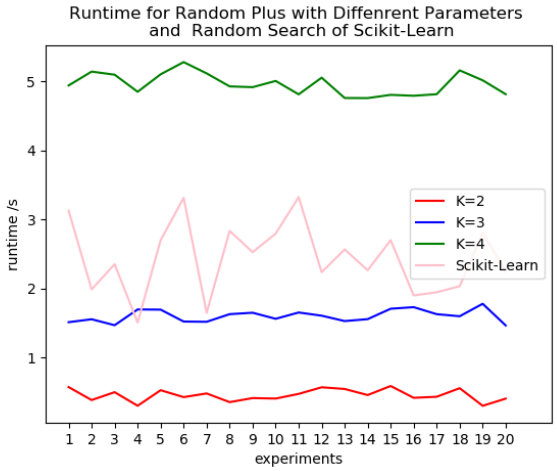


(b) Pima Indians Diabetes

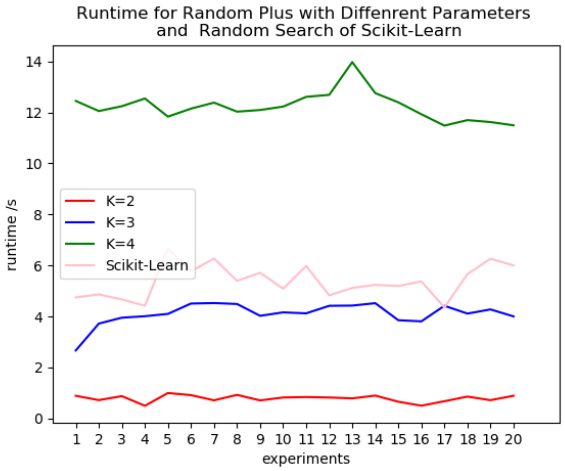


(c) MNIST Hand-written

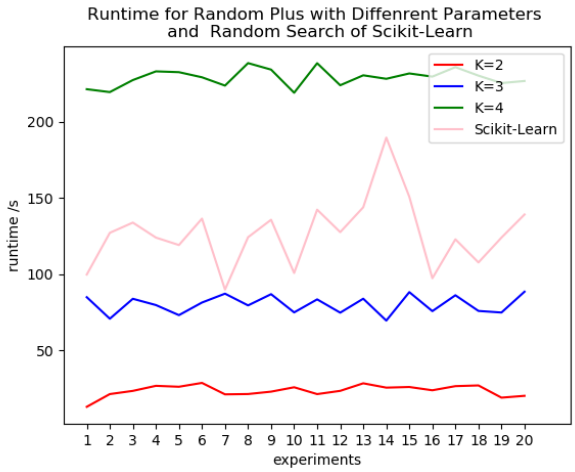
Figure 4.22: Average Silhouette Coefficient for K-means in Experiment 2



(a) Iris Flower

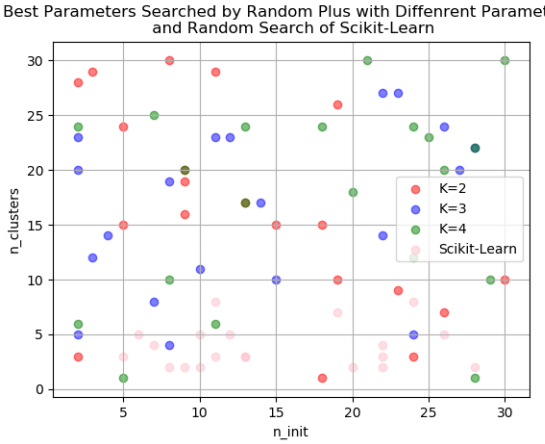


(b) Pima Indians Diabetes

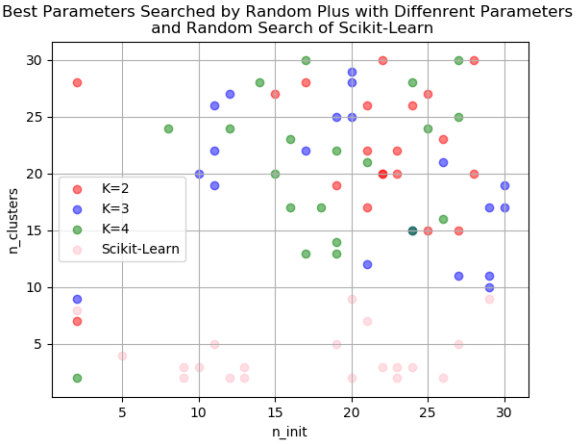


(c) MNIST Hand-written

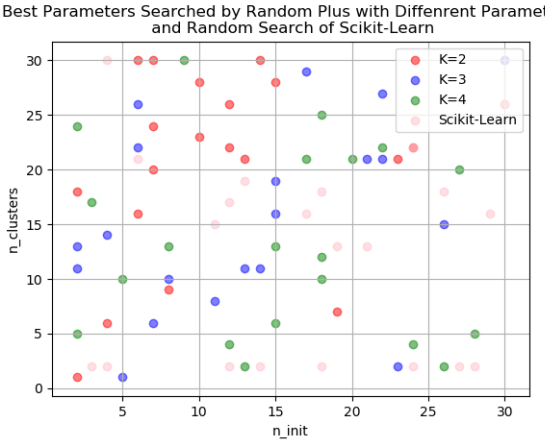
Figure 4.23: Runtime for Each Run for Each Run for K-means in Experiment 2



(a) Iris Flower



(b) Pima Indians Diabetes



(c) MNIST Hand-written

Figure 4.24: Best Parameters Returned for Each Run for K-means in Experiment 2

4.3.3 Neural Network

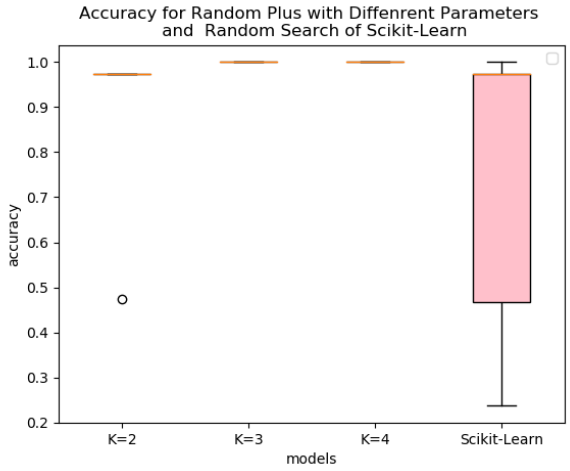
Same range of g and also same the relationship between k and g as in experiment 1.

What should be mentioned here is that in the experiment, all methods will run same times but for different methods the numbers of samples per each run time will be also different. The total number of samples will not be kept in same.

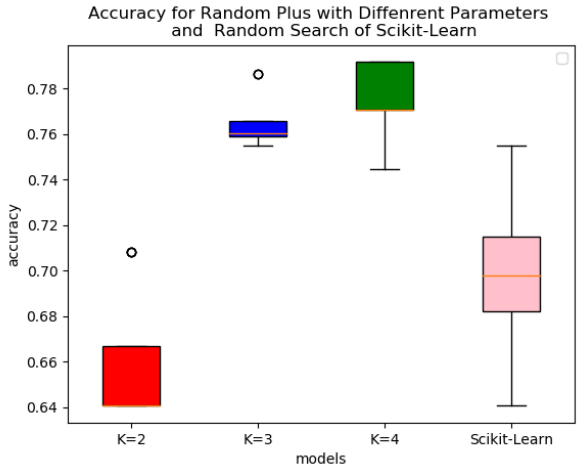
Each run is labeled as an experiment in the results' pictures. So in experiment two, the accuracy is the best accuracy for each run with different numbers of samples. That is, when $k = 2$, $g = 2$, the numbers of subspace in neural network case will be 8 because of 3 hyperparameters, and each run with $k = 2$ will sample 8 points and return a best one of eight(For more information pleas see experiment description in chapter 3).

Accuracy distributions: Variance of accuracy of 20 times of running random search plus with $k = 3$, $k = 4$ are smaller than the random search's from scikit-learn, but for random search plus with $k = 2$, it performs worse than random search(See figure 4.25). Also the average accuracy of 20 times of running of random search plus with $k = 3$, $k = 4$ better than scikit-learn's , and these improvement are very obvious(See figure 4.26). The improvement for average accuracy is about 30% - 40% comparing to the scikit-learn's random search. However, when the $k = 2$, random search plus show an absolute advantage over random search.

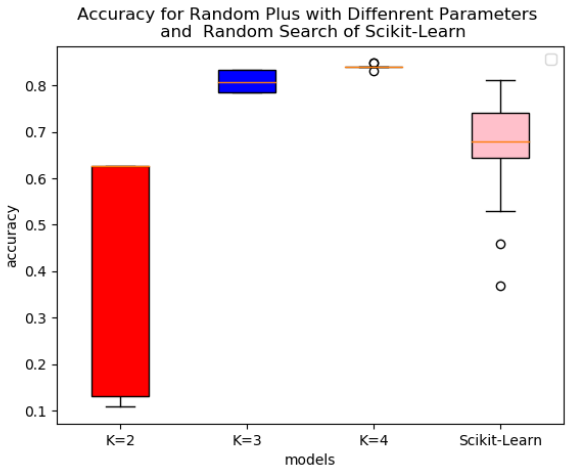
Runtime: The sample numbers of random search from scikit-learn is 10 per run while for random search plus with $k = 2$,$k = 3$,$k = 4$, the numbers are $8, 125, 1000$. As it is shown in figure 4.23, this time, the most numbers of samples' way, random search plus with $k = 4$, is slowest one, and the $k = 2$ is faster than scikit-learn's. It makes sense because $k = 4$ does have the most numbers of samples for each run, and $k = 2$ does have less samples than scikit-learn's. However, the $k = 3$ can still keep the same runtime as scikit-learn's but even with 125 samples. According to the results from accuracy, $k = 4$ make the random search plus get better models than scikit-learn's and improve it a lot, so it means that in this case, random search with $k = 3$ can also find out better points with less samples and also shorter runtime.



(a) Iris Flower

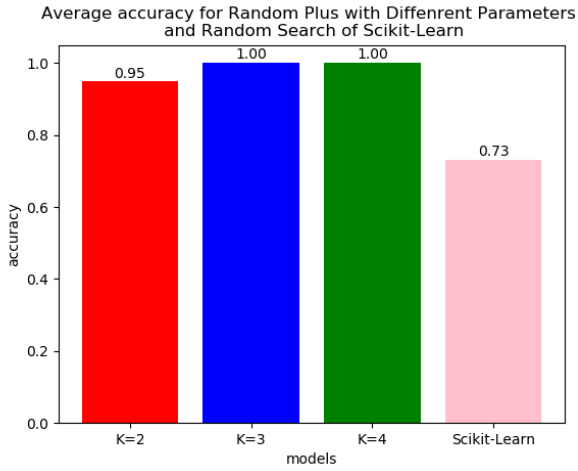


(b) Pima Indians Diabetes

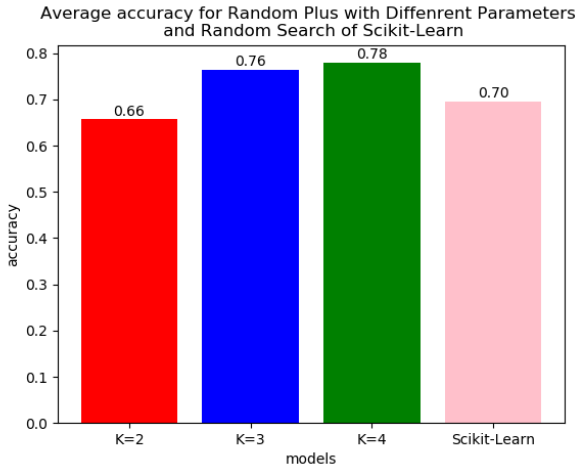


(c) MNIST Hand-written

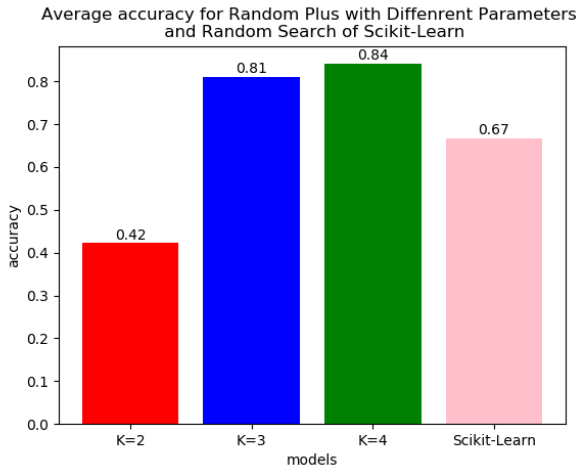
Figure 4.25: Accuracy Distributions for Neural Network in Experiment 2



(a) Iris Flower

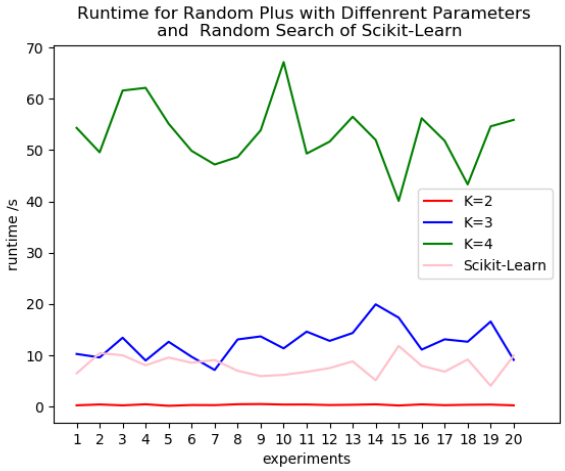


(b) Pima Indians Diabetes

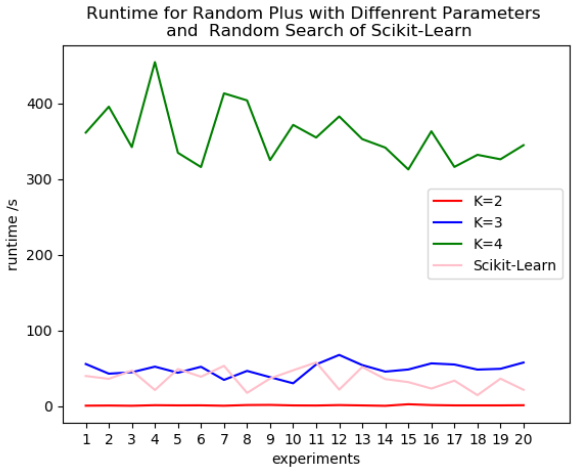


(c) MNIST Hand-written

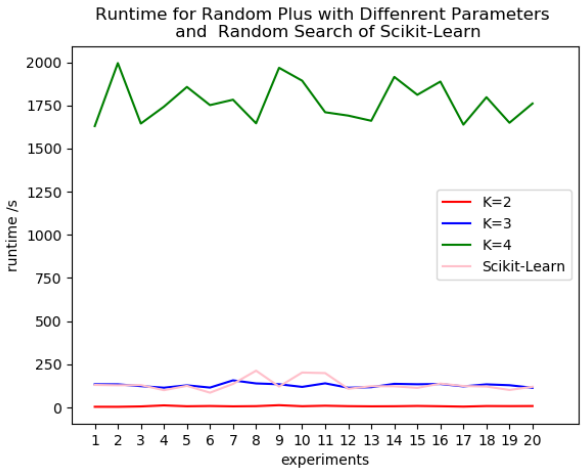
Figure 4.26: Average accuracy for Neural Network in Experiment 2



(a) Iris Flower

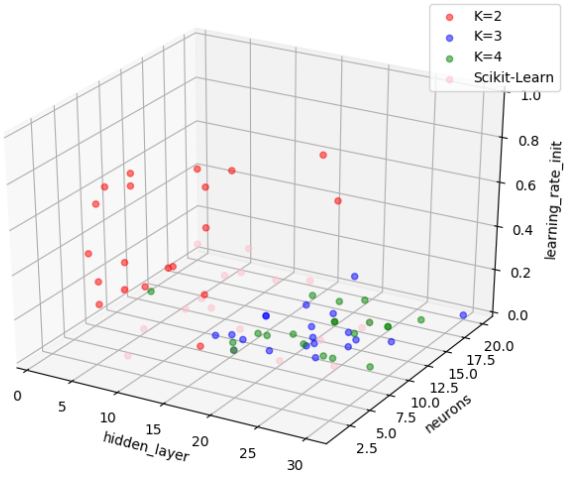


(b) Pima Indians Diabetes

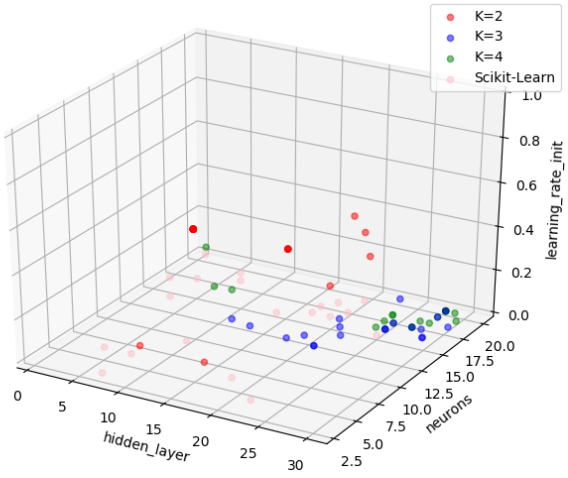


(c) MNIST Hand-written

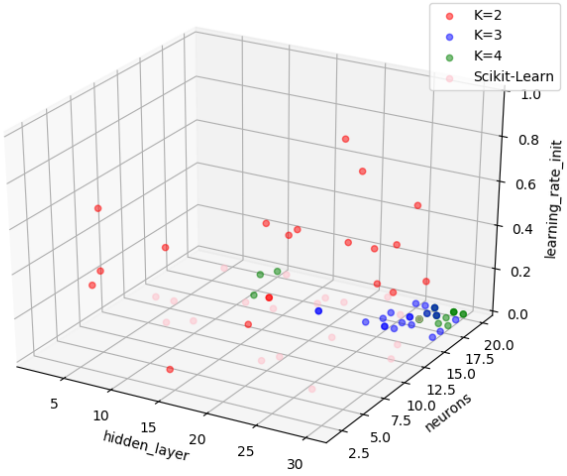
Figure 4.27: Runtime for Each Run for Neural Network in Experiment 2



(a) Iris Flower



(b) Pima Indians Diabetes



(c) MNIST Hand-written

Figure 4.28: Best Parameters Returned for Neural Network in Experiment 2

Best parameters: Even in the 3-D search space, the points found by random search and random search plus are scattered at different areas(See figure 4.28). But they all prefer to choose the low learning rate just with different preference about the other 2 parameters. Also it can show that random search actually is more global than the random search plus, which can be explained by that the points sampled by random search plus are distributed more evenly in the search space.

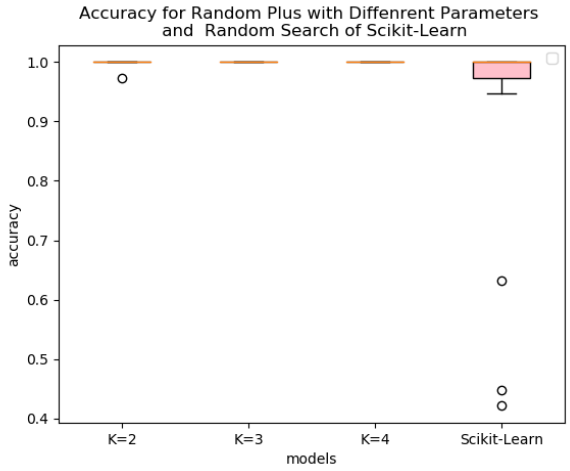
4.3.4 Support Vector Machine

Same relationship between k and g as it is in experiment 1.

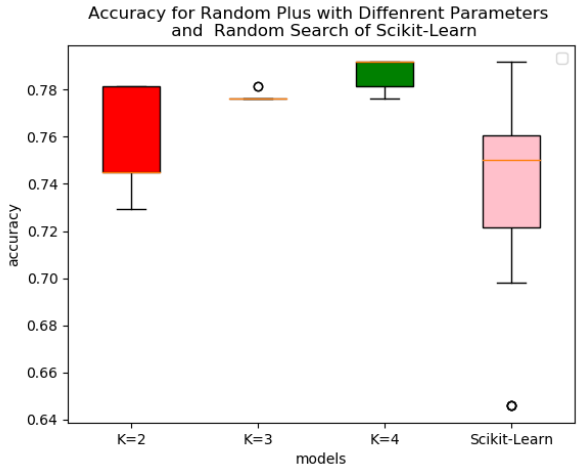
Accuracy distributions: Variance of accuracy of 20 times of running random search plus with $k = 2$, $k = 3$, $k = 4$ are all smaller than the random search's from scikit-learn, even random search plus with $k = 2$, performs a little worse than $k = 3$, $k = 4$(See figure 4.29). Also the average accuracy of 20 times of running of random search plus with $k = 3$, $k = 4$ better than scikit-learn's , and these improvement are very obvious(See figure 4.30). The improvement for average accuracy is about 10% - 50% compared to the scikit-learn's random search.

Runtime: The sample numbers of random search from scikit-learn is 10 per run while for random search plus with $k = 2$, $k = 3$, $k = 4$, the numbers are $4, 25, 10$. As it is shown in figure 4.31, the runtime of different methods for each run is varied, which are caused by the gamma effect the runtime of training and testing of SVM a lot, and also the range of the gamma is log in the search space. But it is still clear that random search plus with $k = 2$ is the fast one and comparing to the accuracy results, it is not hard to know the random search is more efficient.

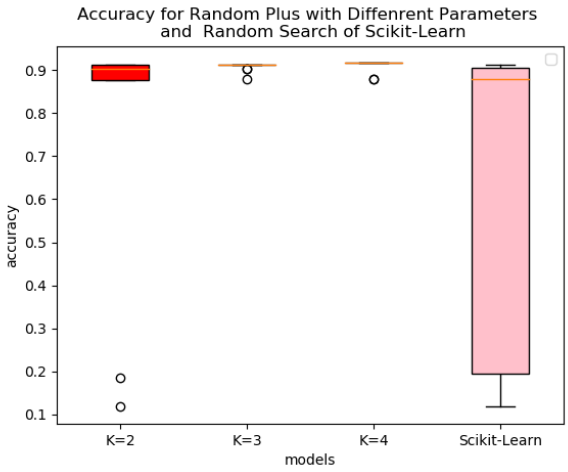
Best parameters: still in the 2-D search space, the points found by random search and random search plus are scattered in different areas(See figure 4.32). Also it can show that random search actually is more global than the random search plus, which can be explained by that the points sampled by random search plus are distributed more evenly in the search space.



(a) Iris Flower

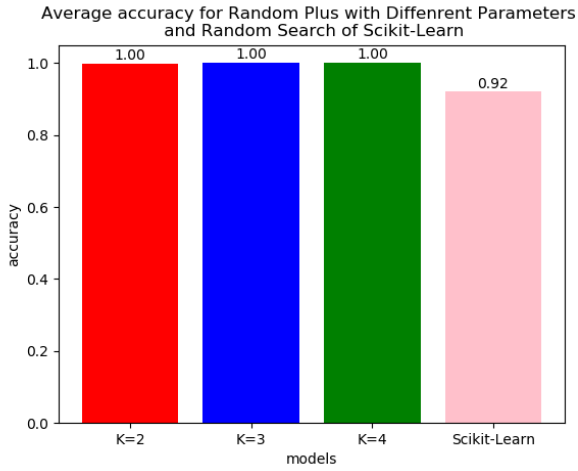


(b) Pima Indians Diabetes

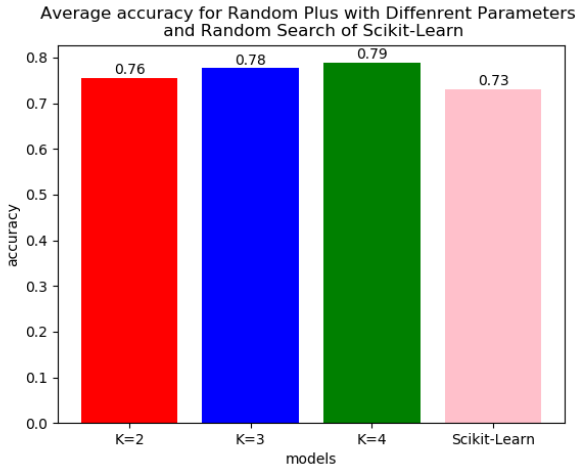


(c) MNIST Hand-written

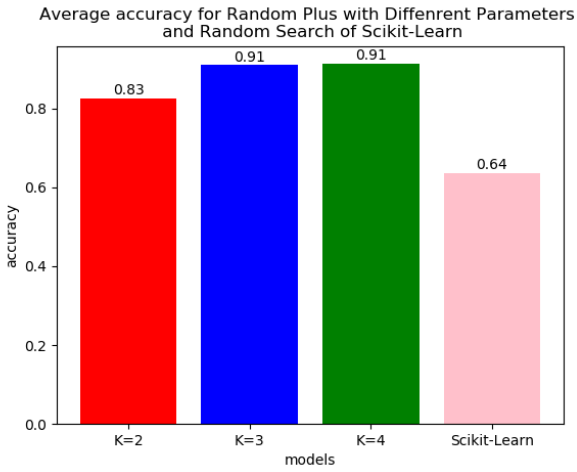
Figure 4.29: Accuracy Distributions for SVM in Experiment 2



(a) Iris Flower

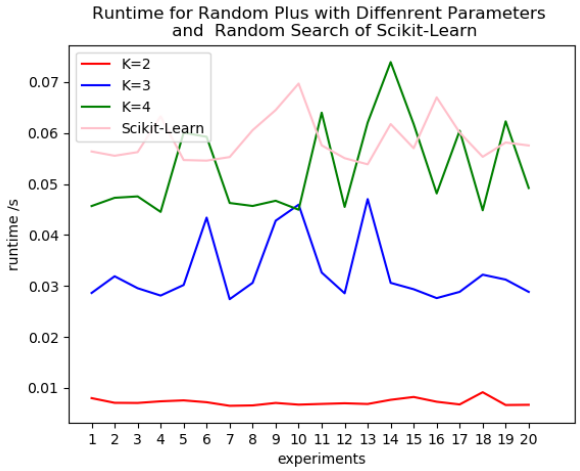


(b) Pima Indians Diabetes

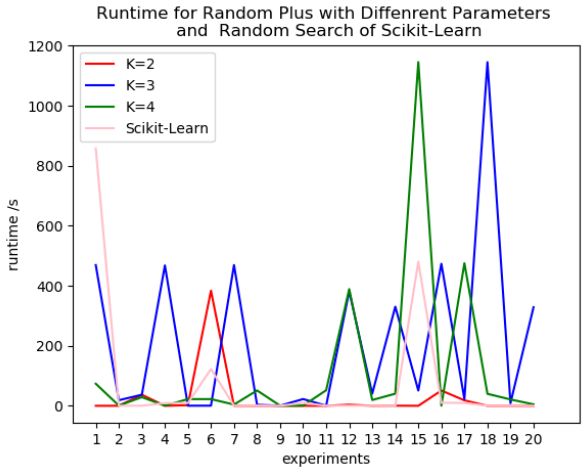


(c) MNIST Hand-written

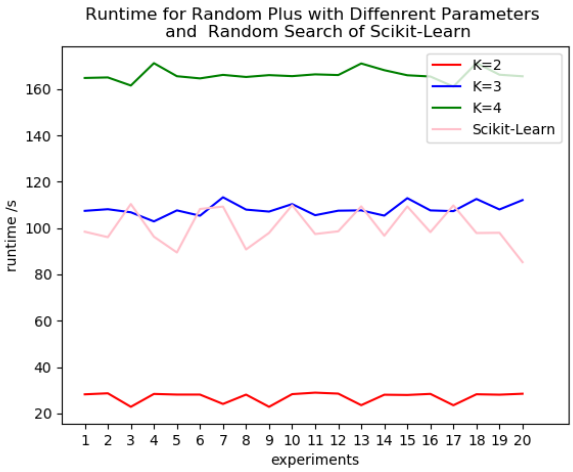
Figure 4.30: Average Accuracy for SVM in Experiment 2



(a) Iris Flower

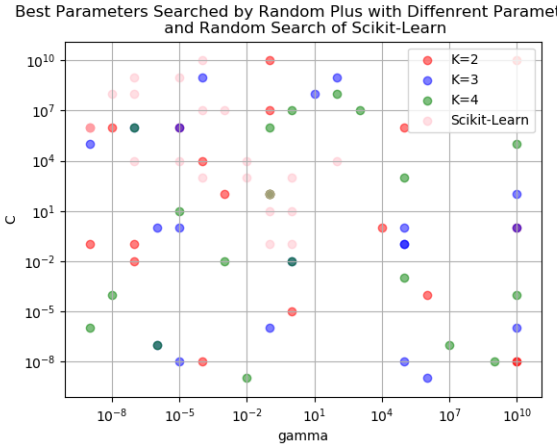


(b) Pima Indians Diabetes

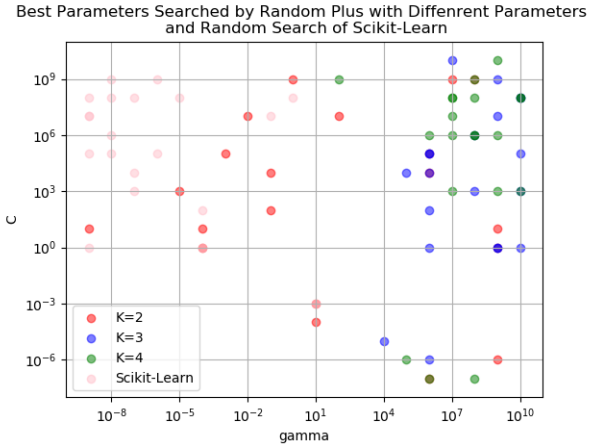


(c) MNIST Hand-written

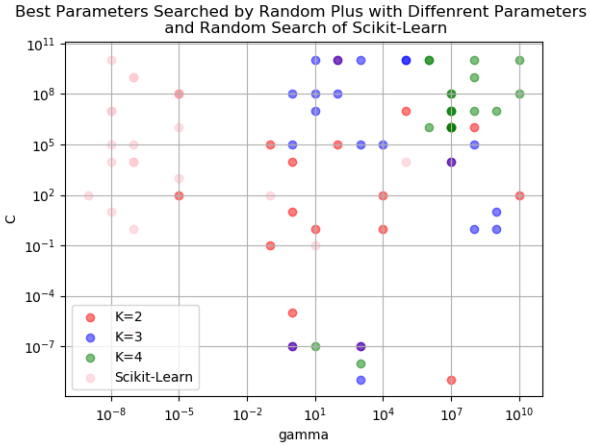
Figure 4.31: Runtime for Each Run for SVM in Experiment 2



(a) Iris Flower



(b) Pima Indians Diabetes



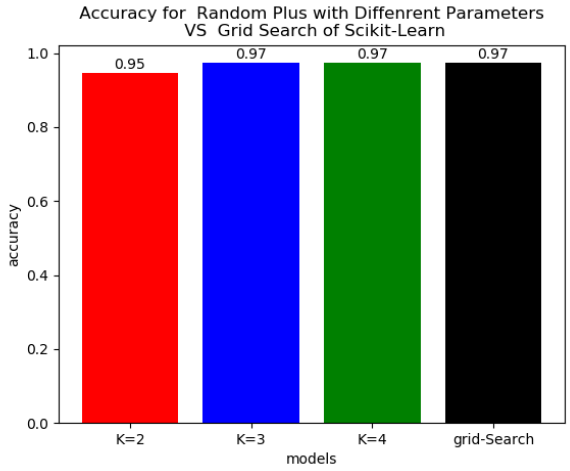
(c) MNIST Hand-written

Figure 4.32: Best Parameters Returned for Each Run for SVM in Experiment 2

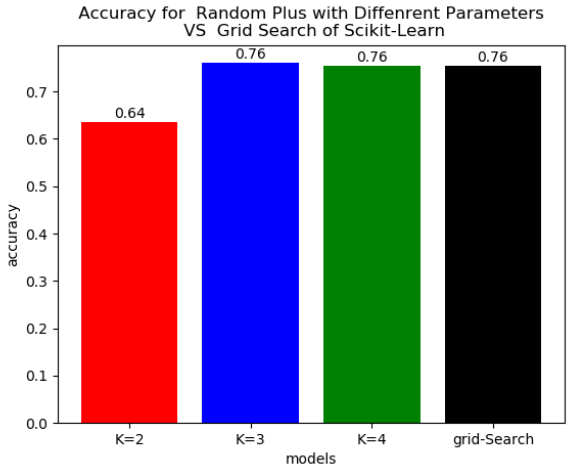
4.4 Extra experiment

4.4.1 Random search plus vs grid search (neural network)

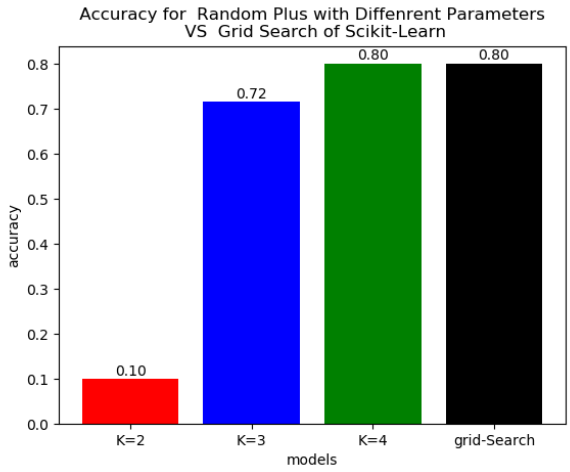
Setup: Same relationship between k and g as in experiment 1 and 2 about neural network.
Accuracy: The accuracy of the model found by grid search should the best one because it is a exhaustive search while the random search plus with $k = 3$, $k = 4$ are very close to grid search and both of them are better than the random search plus with $k = 2$, which is the worest case.
Runtime: The runtime of grid search is longer than all method also because it tried all points in the search space and each point is very expensive. However for random search plus with $k = 3$, $k = 4$ which has same performance to grid search, they just can use only 1% of running time.
Best parameters: Random search plus with $k = 4$ finds out same answer as grid search while with $k = 3$, it finds our different points but with almost same performance as the grid search's, and with $k = 2$, unfortunately it doesn't. The bad performance with random search with low level separation maybe because of how it divides the continuous parameters with a large range and also at same time, it is the important parameter.
Similarity and Difference: The similarity of random search plus and grid search is that they all divide the search space into a lot of parts. For grid search, each part includes a point while for random search plus each part includes a certain numbers of points(depends on k). When k increases, the hyperparameter subspace or the sub regions of search space will also increase, and if the number of hyperparameter subspace is max value, then the random search plus will reduce to a grid search. However, the difference is that if not in the extreme case, random search plus would always randomly choose a point from each hyperparameter subspace or the sub regions of search space. Even if it is the max value like $k = 4$ (there are 1000 hyperparameter subspaces), it can still reduce the runtime a lot because for grid search, it will try a least $30 \times 20 \times 10 = 6000$ such sub area including 1 point.



(a) Iris Flower

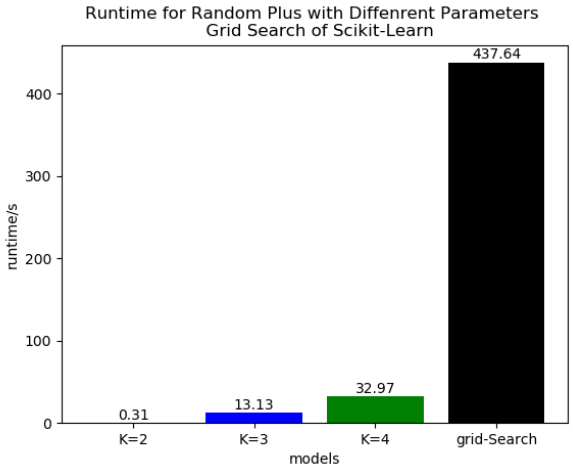


(b) Pima Indians Diabetes

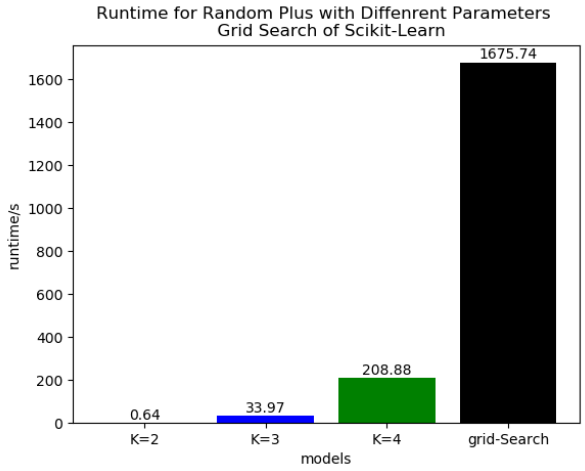


(c) MNIST Hand-written

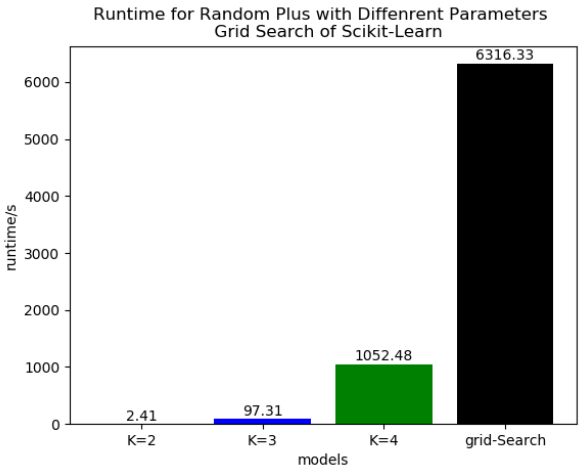
Figure 4.33: Accuracy by Random Search Plus and Grid Search



(a) Iris Flower

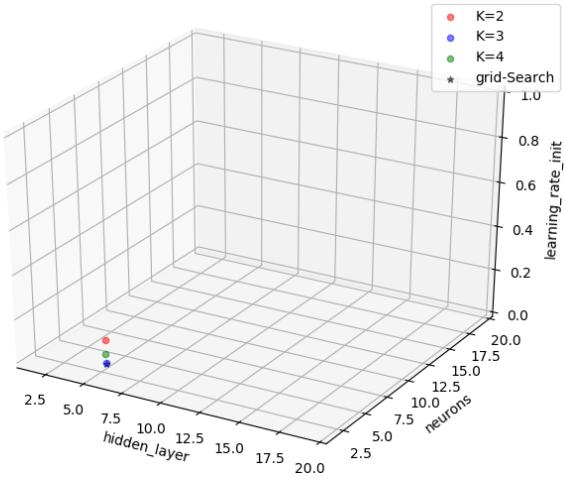


(b) Pima Indians Diabetes

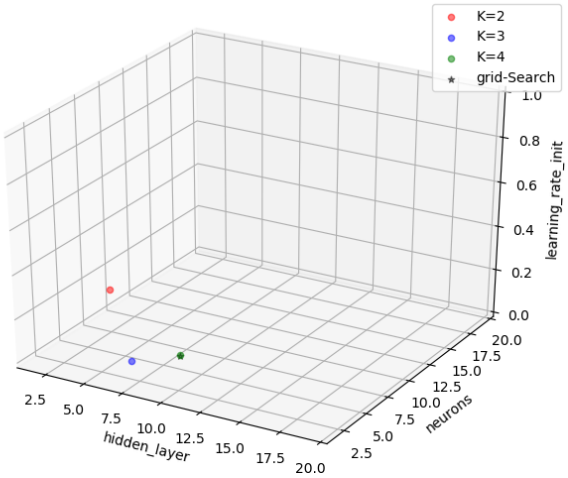


(c) MNIST Hand-written

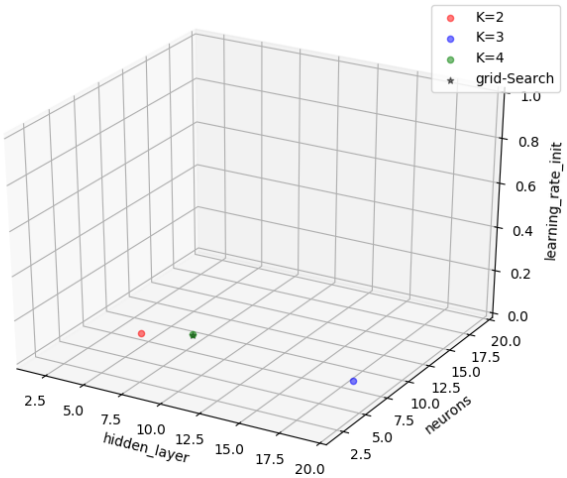
Figure 4.34: Runtime by Random Search Plus and Grid Search



(a) Iris Flower



(b) Pima Indians Diabetes



(c) MNIST Hand-written

Figure 4.35: Best Parameters Returned by Random Search Plus and Grid Search

Chapter 5

Conclusions and Future Work

5.1 Conclusions

1. If a machine learning method has an objective function or a real training process, random search plus has a higher expected value of the samples than random search's.

2. For some machine learning methods which don't have an objective function or a real training process, the random search's effect on hyperparameter optimization will not be obvious (K-NN).

3. Compared to a random search method from scikit-learn(randomly sampling 10 times), random search plus's way of sampling based on an appropriate hyperparameters space separation can be more efficient. It can sample more but with shorter runtime for each run.

4. Compared to a random search method from scikit-learn(randomly sampling 10 times), random search plus can find better parameters than random search's, which can improve supervised learning method's validation accuracy or unsupervised learning method's silhouette coefficient.

5. Comparing different parameters, the $k = 3$ is the best strategy for separating space.While ensuring the optimization effect, it also ensures the least time-consuming.

6. Compared to the grid search, when k is very small, the effect of random search plus is not ideal, but when the appropriate value of k is obtained, the same effect can be achieved

by random search plus but only with one-tenth of the running time or less.

5.2 Future Work

5.2.1 Experiments Improvement

1. More experiments about other machine learning methods like decision tree.
2. More experiments about the optimization on more number and types of hyperparameters through random search plus to observe how it performs in a hyperparameter space with huge dimensions. For example, CNN.
3. Comparing random search to random search plus in a high dimensional hyperparameter optimization.
4. Try some datasets which have size between Pima indians diabetes and MNIST as here is a huge gap.
5. Since there are some atypical ways to do a grid search, like deterministic focused grid search or annealed focused grid search[19], which look similar to random search plus, it is necessary to do comparison experiments of the three in the future.

5.2.2 Algorithm Improvement

1. Change the off-line search into the on-line search; In the core algorithm, model's building and training always happens after sampling is finished. However, it can be changed to do an evaluation immediately after sampling a point and set a score of model's accuracy or some others indexes. Let it stop searching once it get a good model(See figure 5.1).
2. The random search plus's core algorithm of hyperparameter space separations include a process of produce some hyperparameter subspace. Each point at those hyperparameter subspace can match a unique model according to the assumption 3, so those models can be viewed as a population and then they can have chance of exchanging the value of same dimension, which means they can communicate with each others. This feature supports

Algorithm 3 Move_Score

Input: $\mathbb{S}^u(h_1, h_2...h_n)$;
Output: Best_model, Best_parameter

```
Global g
if g > 0 then
    for j = 1 to n do
        S = S^u
        for i = 1 to g do
            h_j = h_j + ic_i
            h_j = h_j − uc_i
            Parameters = Random(S^u)
            Model = Create_Model(Machine_learning_method, Parameters)
            Model.fit()
            if Model.Validation()>Score then
                Best_parameter = Parameters
                Best_model = Model
                return Best_model, Best_parameter
            end if
        end for
        S^u = S
    end for
    Parameters = Random(S^u)
    Model = Create_Model(Machine_learning_method, Parameters)
    Model.fit()
    if Model.Validation()>Score then
        Best_parameter = Parameters
        Best_model = Model
        return Best_model, Best_parameter
    end if
    for j = 1 to n do
        h_j = h_j + ic_i
        h_j = h_j − uc_i
    end for
    g = g − 1
    Move(S^u)
end if
```

Figure 5.1: Code Improvement

that a evolutionary optimization like genetic algorithm or population-based algorithm like particle swarm optimization can work on improving the performance of random search plus on hyperparameter optimization in the future[7].

Bibliography

[1] Babalola, A. E., Ojokoh, B. A., and Odili, J. B. (2020). A review of population-based optimization algorithms. In *2020 International Conference in Mathematics, Computer Engineering and Computer Science (ICMCECS)*, pages 1–7. 7

[2] Bergstra, J. and Bengio, Y. (2012). Random search for hyper-parameter optimization. *J. Mach. Learn. Res.*, 13:281–305. 4, 15, 16

[3] Bergstra, J. S., Bardenet, R., Bengio, Y., and Kégl, B. (2011). Algorithms for hyper-parameter optimization. In *Advances in neural information processing systems*, pages 2546–2554. 4

[4] Bhat, G. S., Shankar, N., and Panahi, I. M. (2020). Automated machine learning based speech classification for hearing aid applications and its real-time implementation on smartphone. In *2020 42nd Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC)*. IEEE. 8

[5] Bhavsar, H. P. and Panchal, M. H. (2012). A review on support vector machine for data classification. 2, 4, 12

[6] Blum, C. and Roli, A. (2003). Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM computing surveys (CSUR)*, 35(3):268–308. 8

[7] Jaderberg, M., Dalibard, V., Osindero, S., Czarnecki, W. M., Donahue, J., Razavi, A., Vinyals, O., Green, T., Dunning, I., Simonyan, K., et al. (2017). Population based training of neural networks. *arXiv preprint arXiv:1711.09846*. 79

[8] Lee, J. H., Shin, J., and Realff, M. J. (2018). Machine learning: Overview of the recent progresses and implications for the process systems engineering field. *Computers Chemical Engineering*, 114:111 – 121. FOCAPO/CPC 2017. 2

[9] Leung, F. H.-F., Lam, H.-K., Ling, S.-H., and Tam, P. K.-S. (2003). Tuning of the structure and parameters of a neural network using an improved genetic algorithm. *IEEE Transactions on Neural networks*, 14(1):79–88. 8

[10] Liashchynskyi, P. and Liashchynskyi, P. (2019). Grid search, random search, genetic algorithm: A big comparison for NAS. *CoRR*, abs/1912.06059. 7

[11] Liu, W., Huang, Q., and Wei, M. (2020). Image quality evaluation based on svm and improved grid search algorithm. In *2020 IEEE 15th International Conference on Advanced Trends in Radioelectronics, Telecommunications and Computer Engineering (TCSET)*, pages 842–845. 12

[12] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al. (2011). Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12(Oct):2825–2830. 4

[13] Romeijn, H. E. (2001). *Random search methodsRandom Search Methods*, pages 2175–2180. Springer US, Boston, MA. 4

[14] Shahriari, B., Swersky, K., Wang, Z., Adams, R. P., and de Freitas, N. (2016). Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175. 8

[15] Wikipedia contributors (2020). Support vector machine — Wikipedia, the free encyclopedia. [Online; accessed 14-November-2020]. 12

[16] Yu, T. and Zhu, H. (2020). Hyper-parameter optimization: A review of algorithms and applications. 2

[17] Zabinsky, Z. B. (2010). Random search algorithms. In Cochran, J. J., Cox, L. A., Keskinocak, P., Kharoufeh, J. P., and Smith, J. C., editors, *Wiley Encyclopedia of Operations Research and Management Science*. John Wiley & Sons, Hoboken, NJ, USA. 4, 5

[18] Zou, J., Han, Y., and So, S.-S. (2008). Overview of artificial neural networks. In *Artificial Neural Networks*, pages 14–22. Springer. 11

[19] Álvaro Barbero Jiménez, López Lázaro, J., and Dorronsoro, J. R. (2009). Finding optimal model parameters by deterministic and annealed focused grid search.

Neurocomputing, 72(13):2824 – 2832. Hybrid Learning Machines (HAIS 2007) / Recent Developments in Natural Computation (ICNC 2007). 77

Appendices

Vita

Originally from China, Bohan Li grew up in Nanchang, China. After high school, he attended North China Institute of Science and Technology and received a Bachelor of Engineering degree in Geological Engineering. Before graduating with his undergraduate degree, he knew he wanted to attend graduate school. He chose to attend the University of Tennessee, Knoxville to pursue a Master of Science degree in Computer Science with a concentration in Machine Learning and Intelligent System. His research interest includes hyperparameter optimization, facial recognition and computer vision. After graduation, he will begin his new position as an research associate at University of Tennessee, Knoxville. He is incredibly grateful for all the support from his family as he begins his new career.