Hyperparameter Tuning :

- It is the main part when you build your machine learning model. As all you know in ml model there are overfiting concept so to reduce this overfit we use hyperparameter tuning.
- If we train a linear regression with SGD,parameters of a model are the slope and the bias and hyperparameter is learning rate.
- Some examples of model hyperparameters include:

```
- The C and sigma hyperparameters for support vector machines.
- The k in k-nearest neighbors.
- Max_Depth in Decision tree
```

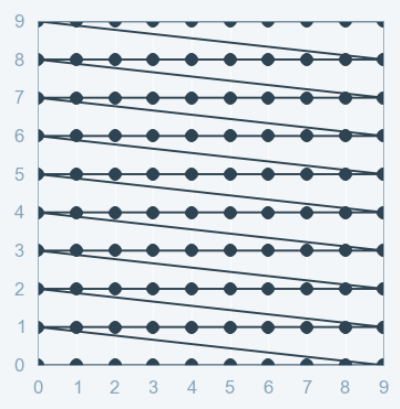
To perform this we have two strategy

```
1. GridSearchCV
2. RandomizedSearchCV
```

- Here i use CV(Cross validation) term in end of every hyperparameter approach because here cross validation also performed.

1. GridSearchCV

- In GridSearchCV approach, machine learning model is evaluated for a range of hyperparameter values. This approach is called GridSearchCV, because it searches for best set of hyperparameters from a grid of hyperparameters values.
- In Grid Search, we try every combination of a preset list of values of the hyper-parameters and evaluate the model for each combination.



Implementation

It Include 4 steps:

1. **initialize all parameter**
2. **Instantiating Ml Model**
3. **Instantiating the GridSearchCV object**
4. **Print the tuned parameters and score**

Here I am taking only Logistic Regression "C" value for hyperparameter tuning. you can take any model.

```
# Necessary imports
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
import numpy as np
```

```
# initialize all parameter
param_grid = {'C': np.logspace(-5, 8, 15) }
```

```
# Instantiating logistic regression classifier
logreg = LogisticRegression()
```

```
# Instantiating the GridSearchCV object
logreg_cv = GridSearchCV(logreg, param_grid, cv = 5)
logreg_cv.fit(X, y) #here always put all data not train data
```

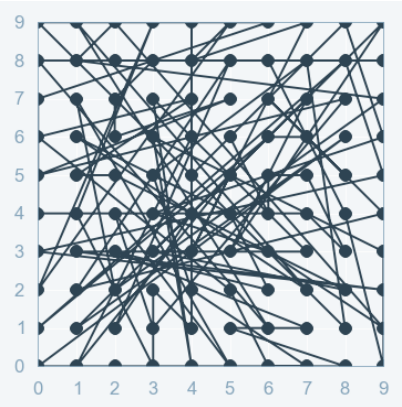
```
# Print the tuned parameters and score
print("Tuned Logistic Regression Parameters: {}".format(logreg_cv.best_params_))
print("Best score is {}".format(logreg_cv.best_score_))
```

Drawbacks Of Gridsearch Cv

- GridSearchCV will go through all the intermediate combinations of hyperparameters which makes grid search computationally very expensive.
- Now i dicuss about Random Search Cv which is faster than grid search cv

2. Random Search Cv

- RandomizedSearchCV solves the drawbacks of GridSearchCV, as it goes through only a fixed number of hyperparameter settings. It moves within the grid in random fashion to find the best set hyperparameters.
- Random search is a technique where random combinations of the hyperparameters are used to find the best solution for the built model.



- Code is same as GridSearch Cv , here instaed of Gridsearch we use Random Search

Implementation

It Include 4 steps:

1. **initialize all parameter**
2. **Instantiating Ml Model**
3. **Instantiating the GridSearchCV object**
4. **Print the tuned parameters and score**

Here I use decision tree parameter for tuning

```
# Necessary imports
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import RandomizedSearchCV
```

```
# initialize all parameter
param_grid = {
    "max_depth" : [50,100,150,200],
    "min_samples_split" : [1,2,3,4,5,6,7,8,9],
    "min_samples_leaf" : [1,2,3,4,5,6,7,8,9],
    "criterion": ["gini", "entropy"],
    "max_leaf_nodes":[1,5,10,15,20]
        }
```

```
# Instantiating logistic regression classifier
classifier = DecisionTreeClassifier()
```

```
# Instantiating the GridSearchCV object
classifier_cv =  RandomizedSearchCV(classifier, param_grid, cv = 5)
classifier_cv.fit(X, y) #here always put all data not train data
```

```
# Print the tuned parameters and score
print("Tuned Decision Tree Parameters: {}".format(classifier_cv.best_params_))
print("Best score is {}".format(classifier_cv.best_score_))
```