Department of Information Engineering Technology

The Superior University, Lahore

Artificial Intelligence Lab

Experiment No.7

Hyper Parameters Tunning

Prepared for

____

By:

Name: ____________

ID: _______________

Section: ___________

Semester: __________

Total Marks: _________________

Obtained Marks: ______________

Signature: __________________

Date: _____________

Experiment No. 7

Hyper Parameters Tunning

(Rubrics)

Name: ____ **Roll No. :** _____________________

A. PSYCHOMOTOR

Sr. No.	**Criteria**	**Allocated Marks**	**Unacceptable 0%**	**Poor 25%**	**Fair 50%**	**Good 75%**	**Excellent 100%**	**Total Obtained**
1	Follow Procedures	1	0	0.25	0.5	0.75	1	
2	Software and Simulations	2	0	0.5	1	1.5	2	
3	Accuracy in Output Results	3	0	0.75	1.5	2.25	3	
	Sub Total	**6**	**Sub Total marks Obtained in Psychomotor(P)**					

B. AFFECTIVE

Sr. No.	**Criteria**	**Allocated Marks**	**Unacceptable 0%**	**Poor 25%**	**Fair 50%**	**Good 75%**	**Excellent 100%**	**Total Obtained**
1	Respond to Questions	1	0	0.25	0.5	0.75	1	
2	Lab Report	1	0	0.25	0.5	0.75	1	
3	Assigned Task	2	0	0.5	1	1.5	2	
	Sub Total	**4**	**Sub Total marks Obtained in Affective (A)**					

Instructor Name:__________________________ **Total Marks (P+A):** _______________

Instructor Signature:________________________ **Obtained Marks (P+A):** ___________

Introduction to GridSearchCV

Hyperparameters are parameters that are not learned from the data but are set before training the model. Tuning these hyperparameters can significantly affect the performance of the model. Hyperparameter tuning using GridSearchCV is a common technique in machine learning for finding the best combination of hyperparameters for a model. GridSearchCV is a function provided by scikit-learn, a popular machine learning library in Python, that is used for hyperparameter tuning and model selection. It stands for "**Grid Search Cross-Validation**." This technique is especially useful when you're working with machine learning algorithms that have hyperparameters, which are parameters that are not learned from the data but need to be set prior to training a model.

Let's delve deeper into how to perform hyperparameter tuning using GridSearchCV in scikit-learn:

- **Select a Model:** First, you need to choose the machine learning model you want to work with. This could be any algorithm from scikit-learn, such as a decision tree, support vector machine, random forest, or any other model.
- **Define Hyperparameters:** Identify the hyperparameters of the selected model that you want to optimize. These are parameters that are not learned from the data but must be set before training. For example, in a support vector machine (SVM), hyperparameters might include the choice of kernel, the regularization strength (C), and kernel-specific parameters like gamma.
- **Create a Parameter Grid:** Specify a grid of hyperparameters and the values you want to explore. For each hyperparameter, create a list of values you'd like to try. The grid will define all the combinations to be tested.

GridSearchCV provides a structured and automated way to systematically search through various combinations of hyperparameters, evaluating a model's performance using cross-validation, and determining the best set of hyperparameters for the task at hand. By conducting an exhaustive search over predefined parameter grids, it takes the guesswork out of hyperparameter tuning and aids in finding the most effective configuration for your machine learning model.

In this process, GridSearchCV efficiently iterates through every possible combination of hyperparameter values specified in the grid, training and evaluating the model on each combination using cross-validation techniques. This approach enables data scientists to make informed decisions about the best hyperparameters, leading to models that are not only better optimized but also more reliable and robust in real-world scenarios.

GridSearchCV is an essential tool for anyone seeking to extract the full potential of machine learning algorithms and harness their power to solve complex problems. It simplifies the process of hyperparameter tuning, making it accessible to both beginners and experts, and significantly contributes to the success of machine learning projects across various domains.

GridSearchCV with Python

GridSearchCV is a function provided by the popular Python library scikit-learn (sklearn) for hyperparameter tuning of machine learning models. GridSearchCV is a way to systematically search for the best combination of hyperparameters for a machine learning algorithm.

Here's how GridSearchCV works:

1. **Importing Libraries:** Start by importing the necessary libraries.

```
from sklearn import svm, datasets
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
from sklearn import svm
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
```

2. **Load Data:** Load your dataset using pandas or any other method suitable for your data source.

```
iris = datasets.load_iris()
df['flower'] = iris.target
df['flower'] = df['flower'].apply(lambda x: iris.target_names[x])
df[47:150]
```

3. **Split the dataset into Training and Testing sets:** Divide your dataset into two parts: a training set used to train the model and a testing set used to evaluate its performance. The *train_test_split* function helps you do this.

```
X_train, X_test, y_train, y_test = train_test_split(iris.data,
iris.target, test_size=0.3)
```

4. **Create and Train the Model:** Creating a model and fit it to your training data.

```
model = svm.SVC(kernel='rbf',C=30,gamma='auto')
model.fit(X_train,y_train)
model.score(X_test, y_test)
```

5. **Cross_val_score using SVM Model:** Assessing the performance of the SVM model with the chosen hyperparameters using cross-validation, which can provide a more robust estimate of the model's generalization performance compared to a single train-test split. The output of these lines will be an array of accuracy scores, one for each fold of the cross-validation process..

```
cross_val_score(svm.SVC(kernel='linear',C=10,gamma='auto'),iris.data,
iris.target, cv=5)
```

```
cross_val_score(svm.SVC(kernel='linear',C=10,gamma='auto'),iris.data,
iris.target, cv=5)
cross_val_score(svm.SVC(kernel='rbf',C=20,gamma='auto'),iris.data,
iris.target, cv=5)
```

6. **Loop:** Performing cross-validation with different combinations of hyperparameters for a Support Vector Machine (SVM) model on the Iris dataset and calculates the average cross-validation scores for each combination. It uses two different kernels ('rbf' and 'linear') and three different values of the regularization parameter 'C' (1, 10, and 20).

```
kernels = ['rbf', 'linear']
C = [1,10,20]
avg_scores = {}
for kval in kernels:
    for cval in C:
        cv_scores =
cross_val_score(svm.SVC(kernel=kval,C=cval,gamma='auto'),iris.data,
iris.target, cv=5)
        avg_scores[kval + '_' + str(cval)] = np.average(cv_scores)

avg_scores
```

7. **GridSearchCV:** GridSearchCV from scikit-learn perform a grid search for hyperparameter tuning on a Support Vector Machine (SVM) model using the Iris dataset. The grid search is conducted to find the best combination of hyperparameters for the SVM model.

```
from sklearn.model_selection import GridSearchCV
clf = GridSearchCV(svm.SVC(gamma='auto'), {
    'C': [1,10,20],
    'kernel': ['rbf','linear']
}, cv=5, return_train_score=False)
clf.fit(iris.data, iris.target)
clf.cv_results_
```

8. **Dataframe:** Analyzing the results of the grid search, including the best hyperparameters and their associated performance metrics. This information is valuable for selecting the optimal hyperparameters for your SVM model.

```
df = pd.DataFrame(clf.cv_results_)
df
df[['param_C','param_kernel','mean_test_score']]
clf.best_params_
clf.best_score_
dir(clf)
```

How about different models with different hyperparameters?

1. Using SVM, Random Forest & Logistic Regression

```
from sklearn import svm
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression

model_params = {
    'svm': {
        'model': svm.SVC(gamma='auto'),
        'params' : {
            'C': [1,10,20],
            'kernel': ['rbf','linear']
        }
    },
    'random_forest': {
        'model': RandomForestClassifier(),
        'params' : {
            'n_estimators': [1,5,10]
        }
    },
    'logistic_regression' : {
        'model':
LogisticRegression(solver='liblinear',multi_class='auto'),
        'params': {
            'C': [1,5,10]
        }
    }
}
```

2. **Dataframe:** Conducting a grid search for multiple machine learning models and collecting the results in a DataFrame to compare the best scores and hyperparameters for each model

```
scores = []

for model_name, mp in model_params.items():
    clf =  GridSearchCV(mp['model'], mp['params'], cv=5,
return_train_score=False)
    clf.fit(iris.data, iris.target)
    scores.append({
        'model': model_name,
        'best_score': clf.best_score_,
        'best_params': clf.best_params_
    })
```

```
df = pd.DataFrame(scores,columns=['model','best_score','best_params'])
df
```

RandomizedSearchCV

RandomizedSearchCV is another hyperparameter tuning technique provided by scikit-learn. While GridSearchCV systematically searches through all possible hyperparameter combinations in a predefined grid, RandomizedSearchCV takes a different approach by randomly sampling a specified number of hyperparameter settings from predefined distributions. This approach can be more efficient, especially when the hyperparameter search space is large.

Here's an overview of how RandomizedSearchCV works:

- **Select a Model:** Just like with GridSearchCV, you start by choosing a machine learning model or algorithm that you want to work with.
- **Define Hyperparameters and Their Distributions:** Instead of specifying specific values for hyperparameters, you define distributions from which hyperparameters will be randomly sampled. This can include uniform, log-uniform, or other types of distributions for each hyperparameter.
- **Specify the Number of Random Samples:** You specify the number of random combinations or samples that should be drawn from the defined hyperparameter distributions.
- **Instantiate RandomizedSearchCV:** You create a RandomizedSearchCV object, passing in the machine learning model, the hyperparameter distributions, the number of random samples, and other configuration options like cross-validation settings and scoring metrics.
- **Fit the Model:** You fit the RandomizedSearchCV object to your training data. It will randomly sample hyperparameter combinations, train and evaluate the model using cross-validation for each combination.
- **Get the Best Parameters and Model:** After the search is complete, you can retrieve the best hyperparameters that resulted in the highest cross-validation performance, as well as the corresponding model.
- **Evaluate the Model**: Finally, you can use the best model obtained from the random search to make predictions on new data and assess its performance.

```
from sklearn.model_selection import RandomizedSearchCV
rs = RandomizedSearchCV(svm.SVC(gamma='auto'), {
        'C': [1,10,20],
        'kernel': ['rbf','linear']
    },
    cv=5,
    return_train_score=False,
    n_iter=2
)
rs.fit(iris.data, iris.target)
pd.DataFrame(rs.cv_results_)[['param_C','param_kernel','mean_test_score'
]]
```

Bias & Variance

Bias and variance are two fundamental concepts in the field of machine learning, particularly when it comes to model evaluation and understanding the performance of machine learning models. They represent two sources of errors that influence the predictive capabilities of a model.

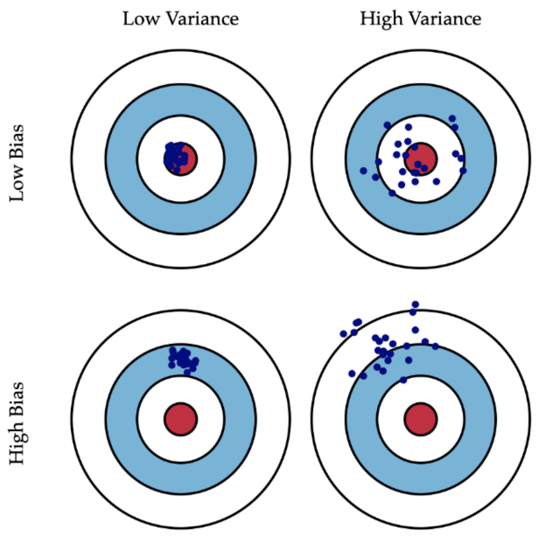
Bias:

- **High Bias:** A model with high bias is overly simplistic and fails to capture the underlying patterns in the data. This leads to underfitting, where the model's predictions are consistently far from the actual values. In other words, it is too biased towards a particular assumption about the data.
- **Low Bias:** A model with low bias is complex and flexible, capable of capturing fine-grained patterns in the data. It doesn't make strong assumptions and fits the training data very closely. Low bias models are better at modeling complex relationships in the data.

Variance:

- **High Variance:** A model with high variance is very sensitive to small fluctuations in the training data. It tends to fit the noise in the data, which means it performs well on the training data but poorly on unseen data. This is known as **overfitting**. High variance models are overly complex and capture noise in the data.
- **Low Variance:** A model with low variance is stable and generalizes well to new, unseen data. It doesn't overreact to small changes in the training data and is less likely to **overfit.** Low variance models are typically simpler and have smoother decision boundaries.

The relationship between bias and variance is often described as a trade-off. Increasing model complexity reduces bias but increases variance, and vice versa. The goal is to find a balance that minimizes both bias and variance, resulting in a model that generalizes well to new data. This concept is known as the bias-variance trade-off.



Model evaluation techniques, such as cross-validation, can help you assess the bias and variance of your model. When a model has high bias and high variance, it's said to have an underfitting problem. When a model has low bias and high variance, it's overfitting. Achieving the right balance between bias and variance is essential for building models that make accurate predictions and are robust to new data.

Lab Task:

Use 2 Classification algorithms and conclude which is best:

1. Load digits dataset from scikit-learn.

    - Apply classification algorithms such as Random Forest, and k-Nearest Neighbors.
    - Train the classification models on the training data.
    - Create a confusion matrix to visualize the model's performance in recognizing each digit.
    - Experiment with different hyperparameters for your chosen model to improve its performance. You can use techniques like GridSearchCV or RandomizedSearchCV.

Home Task:

Use 3 Regression algorithms and conclude which is best:

1. Build a regression model using scikit-learn. You can use a dataset like the Boston Housing dataset
    - Load the dataset.
    - Preprocess and explore the data.
    - Split the data into training and testing sets.
    - Use regression algorithm (e.g., Linear Regression, Decision Trees, Random Forest).
    - Tune hyperparameters for better performance (use GridSearchCV)