UTM

UNIVERSITI TEKNOLOGI MALAYSIA

UNIVERSITI TEKNOLOGI MALAYSIA

FACULTY OF COMPUTING SEMESTER 2,

SESION 2023/2024

ASSIGNMENT 3 (HYPERPARAMETER OPTIMIZATION)

SECB4313 : BIOINFORMATIK MODEL DAN SIMULASI

SECTION 1

NAME	METRIC NO
MUHAMMAD DANISH BIN TAZURIN	B21EC0035
AMZAR RAZIQ KHAN BIN AZAM KHAN	B21EC0011

SUBMISSION DATE : 8 JUNE 2024

LECTURER'S NAME : DR AZURAH BTE A SAMAH

Table of Contents

1. Introduction **3**

2. Grid Search and Random Search **3**

3. Results **5**

4. Why Hyper Parameter Optimization/tuning is Vital in Order to Enhance the Model's Performance **6**

5. Python Appendix **6**

1.Introduction

Four hyperparameters which are learning rate, number of epochs, batch size, and number of neurons in the hidden layer were chosen in earlier trials. The combination that produced the best results was experiment number 0, which employed 64 neurons in the hidden layer, 50 epochs, a batch size of 16, and a learning rate of 0.01 to obtain an accuracy score of 0.836066. A learning rate of 0.01 allowed for steady progress without overshooting optimal weights. 50 epochs offered sufficient time for effective learning without overfitting. A batch size of 16 added useful noise to the gradient estimates and improved generalization and 64 neurons in the hidden layer offered sufficient capacity to capture data complexity without overfitting were the rationales behind the selection of each hyperparameter value.

2.Grid Search and Random Search

In this assignment, hyperparameters were optimized using Grid Search and Random Search as an alternative to manual tuning. These automated techniques offer a systematic approach to improve model performance. An appendix contains the implementation code for these searches. The ideal parameters and the model accuracy with these adjusted hyperparameters are displayed in the screenshot below.

Whereas Random Search chooses parameter combinations at random from a predetermined range, Grid Search thoroughly examines a given parameter grid to identify the ideal combination. By determining appropriate hyperparameters, these approaches efficiently enhance model performance while requiring less time and effort than manual tuning. The outcomes show how these automated search methods can greatly improve the generalizability and accuracy of the models.

Grid Search Output:

```
Grid Search Best Parameters: {'batch_size': 16, 'epochs': 100, 'learning_rate': 0.01, 'neurons': 64}
Grid Search Best Cross-Validation Accuracy: 0.7603395061728394
Grid Search Test Accuracy: 0.8524590163934426
Grid Search Computational Time: 113.44 seconds
```

Grid Search Best Parameter Setting:

Hyperparameter	Value
Learning Rate	0.01
Epochs	50
Batch Size	16
Neurons	64

Metric	Value
Grid Search Best Cross-Validation Accuracy	0.7603
Grid Search Test Accuracy	0.8525
Grid Search Computational Time	113.44 seconds

Random Search Output:

```
Random Search Best Parameters: {'neurons': 128, 'learning_rate': 0.01, 'epochs': 100, 'batch_size': 16}
Random Search Best Cross-Validation Accuracy: 0.7727366255144034
Random Search Test Accuracy: 0.8688524590163934
Random Search Computational Time: 66.78 seconds
```

Random Search Best Parameter Setting:

Hyperparameter	Value
Batch Size	16
Epochs	100
Learning Rate	0.01

Neurons	128

Metric	**Value**
Best Cross-Validation Accuracy (Random Search)	0.7727
Test Accuracy (Random Search)	0.8689
Computational Time (Random Search)	66.78

3.Results

a) Both Grid Search and Random Search provide substantial advantages over manual hyperparameter tuning in terms of work required to obtain the desired results. Grid Search guarantees a comprehensive search by carefully examining each and every combination inside a given parameter grid. However, because this method is computationally thorough and requires analyzing every potential combination, it can be labor-intensive. In contrast, Random Search chooses random subsets of parameters from a set of parameters. It can be implemented more quickly and with less work, but it might not cover the whole parameter space as thoroughly as Grid Search.

b) Grid Search's thorough nature makes it more likely to be time- and computationally-intensive. With the following parameters which were batch size of 16, 50 epochs, learning rate of 0.01, and 64 neurons in the hidden layer, Grid Search took 113.44 seconds to discover the best results, giving a cross-validation accuracy of 0.7603 and a test accuracy of 0.8525.

   On the other hand, because Random Search selects parameters at random, it is usually faster. With the following parameters, it finished the search in 66.78 seconds, with a better test accuracy of 0.8689 and a higher cross-validation accuracy of 0.7727: batch size of 16, 100 epochs, learning rate of 0.01 and 128 neurons in the hidden layer.

This shows that Random Search is more efficient and has the ability to find high-performing models at a lower computational cost than Grid Search. It also required less computational time and discovered a more effective parameter choice.

4. Why Hyper Parameter Optimization/tuning is Vital in Order to Enhance the Model's Performance

Hyperparameter optimization is vital for enhancing a model's performance because it directly influences the model's ability to learn and generalize from data. Proper tuning of hyperparameters like learning rate, batch size, number of epochs, and the number of neurons in hidden layers can significantly improve accuracy and prevent issues like underfitting and overfitting. An optimal learning rate ensures efficient convergence, while the appropriate number of epochs and batch size affects training stability and speed. Additionally, the right number of neurons in hidden layers ensures the model has enough capacity to capture data complexity without becoming overly complex.

Efficient hyperparameter tuning also increases computational efficiency, saving time and reducing costs by optimizing resource use during training. Automated techniques like Grid Search and Random Search streamline this process by systematically exploring various hyperparameter values, reducing the need for manual trial-and-error. These methods ensure the model is tailored to the dataset's specific characteristics, achieving a good bias-variance trade-off and enhancing the model's generalizability and robustness in real-world applications.

5.Appendix Python Codes

```
import pandas as pd
import numpy as np
from sklearn.metrics import accuracy_score
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from sklearn.base import BaseEstimator, ClassifierMixin
from scipy.stats import uniform
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import Adam
from sklearn.metrics import accuracy_score
import time
```

```
from google.colab import drive
drive.mount("/content/gdrive")
dataset_dir = "/content/gdrive/My Drive/Heart/"
data = pd.read_csv(dataset_dir+'heart.csv')
data.head()
```

```
Drive already mounted at /content/gdrive; to attempt to forcibly remount, call drive.mount("/content/gdrive", force_remount=True).
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1

```
catagorialList = ['sex','cp','fbs','restecg','exang','ca','thal']
for item in catagorialList:
    data[item] = data[item].astype('object')
```

```
data = pd.get_dummies(data, drop_first=True)
```

```
y = data['target'].values
y = y.reshape(y.shape[0],1)
X = data.drop(['target'],axis=1)

X.shape
```

```
(303, 21)
```

```
minx = np.min(X)
maxx = np.max(X)
X = (X - minx) / (maxx - minx)
X.head()
```

	age	trestbps	chol	thalach	oldpeak	slope	sex_1	cp_1	cp_2	cp_3	...
0	0.111702	0.257092	0.413121	0.265957	0.004078	0.000000	0.001773	0.000000	0.000000	0.001773	...
1	0.065603	0.230496	0.443262	0.331560	0.006206	0.000000	0.001773	0.000000	0.001773	0.000000	...
2	0.072695	0.230496	0.361702	0.304965	0.002482	0.003546	0.000000	0.001773	0.000000	0.000000	...
3	0.099291	0.212766	0.418440	0.315603	0.001418	0.003546	0.001773	0.001773	0.000000	0.000000	...
4	0.101064	0.212766	0.627660	0.289007	0.001064	0.003546	0.000000	0.000000	0.000000	0.000000	...

5 rows × 21 columns

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
class KerasClassifier(BaseEstimator, ClassifierMixin):
    def __init__(self, learning_rate=0.01, neurons=64, epochs=50, batch_size=16):
        self.learning_rate = learning_rate
        self.neurons = neurons
        self.epochs = epochs
        self.batch_size = batch_size
        self.model = None

    def fit(self, X, y):
        self.model = Sequential()
        self.model.add(Dense(self.neurons, input_dim=X.shape[1], activation='relu'))
        self.model.add(Dense(1, activation='sigmoid'))
        optimizer = Adam(learning_rate=self.learning_rate)
        self.model.compile(loss='binary_crossentropy', optimizer=optimizer, metrics=['accuracy'])
        self.model.fit(X, y, epochs=self.epochs, batch_size=self.batch_size, verbose=0)
        return self

    def predict(self, X):
        return (self.model.predict(X) > 0.5).astype("int32")

    def predict_proba(self, X):
        return self.model.predict(X)
```

```
# Hyperparameters Values
param_grid = {
    'learning_rate': [0.01, 0.1],
    'neurons': [64, 128],
    'epochs': [50, 100],
    'batch_size': [16, 32]
}
```

```
keras_clf = KerasClassifier()
```

```
# Grid Search
start_time_grid = time.time()
grid_search = GridSearchCV(estimator=keras_clf, param_grid=param_grid, cv=3, scoring='accuracy')
grid_search.fit(X_train, y_train)
end_time_grid = time.time()

# Get the best parameters and best score from Grid Search
best_params_grid = grid_search.best_params_
best_score_grid = grid_search.best_score_

# Test the best model from Grid Search on the test set
best_model_grid = grid_search.best_estimator_
y_pred_grid = best_model_grid.predict(X_test)
test_accuracy_grid = accuracy_score(y_test, y_pred_grid)

# Print Grid Search results
print("Grid Search Best Parameters:", best_params_grid)
print("Grid Search Best Cross-Validation Accuracy:", best_score_grid)
print("Grid Search Test Accuracy:", test_accuracy_grid)
print("Grid Search Computational Time: {:.2f} seconds".format(end_time_grid - start_time_grid))
```

```
Grid Search Best Parameters: {'batch_size': 16, 'epochs': 100, 'learning_rate': 0.01, 'neurons': 64}
Grid Search Best Cross-Validation Accuracy: 0.7603395061728394
Grid Search Test Accuracy: 0.8524590163934426
Grid Search Computational Time: 113.44 seconds
```

```python
# Random Search
start_time_random = time.time()
random_search = RandomizedSearchCV(estimator=keras_clf, param_distributions=param_grid, n_iter=10, cv=3, scoring='accuracy', random_state=42)
random_search.fit(X_train, y_train)
end_time_random = time.time()

# Get the best parameters and best score from Random Search
best_params_random = random_search.best_params_
best_score_random = random_search.best_score_

# Test the best model from Random Search on the test set
best_model_random = random_search.best_estimator_
y_pred_random = best_model_random.predict(X_test)
test_accuracy_random = accuracy_score(y_test, y_pred_random)

# Print Random Search results
print("Random Search Best Parameters:", best_params_random)
print("Random Search Best Cross-Validation Accuracy:", best_score_random)
print("Random Search Test Accuracy:", test_accuracy_random)
print("Random Search Computational Time: {:.2f} seconds".format(end_time_random - start_time_random))
```

```
Random Search Best Parameters: {'neurons': 128, 'learning_rate': 0.01, 'epochs': 100, 'batch_size': 16}
Random Search Best Cross-Validation Accuracy: 0.7727366255144034
Random Search Test Accuracy: 0.8688524590163934
Random Search Computational Time: 66.78 seconds
```