FACULTY OF COMPUTING

SECB4313-01
BIOINFORMATICS MODELING AND SIMULATION
ASSIGNMENT 3

LECTURER:

DR. AZURAH BTE A SAMAH

GROUP MEMBERS:

AMIR ISKANDAR BIN NORKHAIRULAZADDIN (A20EC0011)

IMAN EHSAN BIN HASSAN (A20EC0048)

Summary of the Best Hyper Parameters

There are 4 hyper parameters that were selected for previous experiments. The hyperparameters are learning rate, number of epochs, batch size and number of neurons in the hidden layer. From the previous experiment, the best combination of hyperparameters that was obtained is from experiment number 0 with the value learning_rate of 0.01, 50 epochs, batch_size of 16 and the number of neurons in hidden layers is 64 which produces the accuracy score of 0.868852. For more explanation, each of the hyperparameter values was justified. The learning rate of 0.01 allowed for steady pace as if it is the rate too high, the model can be overshoot in optimal weight. For 50 epochs were sufficient for learning without overfitting. A batch size of 16 introduced good noise into the gradient estimates while helping in enhancing the generalization and for the 64 neurons in the hidden layer offered enough capacity to capture the data's complexity without overfitting.

Implementation of Grid Search and Random Search

In this assignment, Grid Search and Random Search was implemented for this experiment instead of tuning it using a set of hyperparameters values. The code for the implementation has been attached at the appendix of the document. In this section, we will display the results that were obtained after implementing the Grid Search and Random Search. Below is the screenshot of the results that was obtained after running the search which produced the best parameter and the accuracy for the model which uses the hyperparameters.

Grid Search

```
Grid Search Best Parameters: {'batch_size': 16, 'epochs': 100, 'learning_rate': 0.01, 'neurons': 128}
Grid Search Best Cross-Validation Accuracy: 0.7604938271604939
Grid Search Test Accuracy: 0.8524590163934426
Grid Search Computational Time: 138.87 seconds
```

Grid Search Best Parameter :
'batch_size': 16
'epochs': 100
'learning_rate': 0.01
'neurons': 128

Grid Search Best Cross-Validation Accuracy: 0.76049
Grid Search Test Accuracy: 0.8524590163934426

Random Search

```
Random Search Best Parameters: {'neurons': 128, 'learning_rate': 0.01, 'epochs': 100, 'batch_size': 16}
Random Search Best Cross-Validation Accuracy: 0.7935699588477366
Random Search Test Accuracy: 0.8524590163934426
Random Search Computational Time: 89.54 seconds
```

Random Search Best Parameters:
'neurons': 128
'learning_rate': 0.01
'epochs': 100
'batch_size': 16

Random Search Best Cross-Validation Accuracy: 0.7935699588477366
Random Search Test Accuracy: 0.8524590163934426

Results Discussion

In this section, we will discuss the results of the grid search and random search implementation that has been made. The comparisons are in terms of how much effort the method needs to achieve the optimal solution and their computational time. Below shows the screenshot of the report that was generated at the end of each run that shows the best score and hyper parameter configuration that achieved the best performance.

```
print("\nComparison:")
print(f"Grid Search - Best Params: {best_params_grid}, CV Accuracy: {best_score_grid}, Test Accuracy: {test_accuracy_grid}")
print(f"Random Search - Best Params: {best_params_random}, CV Accuracy: {best_score_random}, Test Accuracy: {test_accuracy_random}")
print("Grid Search Computational Time: {:.2f} seconds".format(end_time_grid - start_time_grid))
print("Random Search Computational Time: {:.2f} seconds".format(end_time_random - start_time_random))
```

```
Comparison:
Grid Search - Best Params: {'batch_size': 16, 'epochs': 100, 'learning_rate': 0.01, 'neurons': 128}, CV Accuracy: 0.7604938271604939, Test Accuracy: 0.8524590163934426
Random Search - Best Params: {'neurons': 128, 'learning_rate': 0.01, 'epochs': 100, 'batch_size': 16}, CV Accuracy: 0.7935699588477366, Test Accuracy: 0.8524590163934426
Grid Search Computational Time: 138.87 seconds
Random Search Computational Time: 89.54 seconds
```

Comparison:
Grid Search - Best Params: {'batch_size': 16, 'epochs': 100, 'learning_rate': 0.01, 'neurons': 128},
CV Accuracy: 0.7604938271604939, Test Accuracy: 0.8524590163934426

Random Search - Best Params: {'neurons': 128, 'learning_rate': 0.01, 'epochs': 100, 'batch_size': 16},
CV Accuracy: 0.7935699588477366, Test Accuracy: 0.8524590163934426

Grid Search Computational Time: 138.87 seconds

Random Search Computational Time: 89.54 seconds

From results above, both Grid Search and Random Search identified the same optimal hyperparameters which are the neurons in the hidden layer is 128, the learning rate is 0.01, number of epochs is 100, the batch size is 16 which leads to the highest Test Accuracy of 0.852459. This consistency indicates that these hyperparameters are likely very well-suited for the model and data. The higher CV accuracy in Random Search suggests it may have been more effective in generalizing during the hyperparameter search process.

For more comprehensive comparison, we will look at the effort to get the results and the computational time of these methods. In comparison, grid search is more time consuming when compared to random search. This is because grid search involves an exhaustive search through the hyperparameter space which it needs to evaluate every possible combination of hyperparameters. Meanwhile, random search samples a subset of the hyperparameter space randomly which results in a less thorough but much faster way to get the optimal solution. In terms of computational time, grid search took 138.87 seconds to complete while random search just took 89.54 seconds. This is because of the effort that grid search needs to take are explained above which is longer when compared to random search in terms of finding the best hyperparameter which results in a longer computational time.

Explanation of the importance of hyperparameter optimization

Hyperparameter optimization is crucial to enhance a model's performance in order to achieve high model accuracy, avoiding overfitting and underfitting, and improving computational efficiency. It also helps to transform a good model into a great one by selecting the right hyperparameters that align with the specific data and task. Properly tuned hyperparameters can significantly improve model accuracy, stability, and generalization capabilities, ensuring better performance on unseen data. Efficient hyperparameter optimization can also save computational resources and reduce training time.

In conclusion, proper hyperparameter optimization helps the model perform consistently across different datasets and scenarios. From this experiment, by achieving similar Test Accuracy of 0.852459 in both Grid Search and Random Search, it shows that the tuning process has found a set of parameters that generalize well with the dataset.

Appendix - Python Codes

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import Adam
from sklearn.metrics import accuracy_score
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from sklearn.base import BaseEstimator, ClassifierMixin
from scipy.stats import uniform
import time
from google.colab import drive
drive.mount("/content/gdrive")
dataset_dir = "/content/gdrive/My Drive/Colab Notebooks/"
data = pd.read_csv(dataset_dir+'heart.csv')
data.head()
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
minx = np.min(X)
maxx = np.max(X)
X = (X - minx) / (maxx - minx)
X.head()
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
# Custom Keras classifier for use with sklearn's GridSearchCV and
RandomizedSearchCV
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)
# Define hyperparameters and their 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_
```

```python
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))
# 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))
print("\nComparison:")
print(f"Grid Search - Best Params: {best_params_grid}, CV Accuracy:
{best_score_grid}, Test Accuracy: {test_accuracy_grid}")
```

```
print(f"Random Search - Best Params: {best_params_random}, CV Accuracy:
{best_score_random}, Test Accuracy: {test_accuracy_random}")
print("Grid Search Computational Time: {:.2f}
seconds".format(end_time_grid - start_time_grid))
print("Random Search Computational Time: {:.2f}
seconds".format(end_time_random - start_time_random))
```