SECB 4313-01

BIOINFORMATICS MODELING AND SIMULATION

ASSIGNMENT 3

Lecturer:

Dr. Azurah bte A Samah

Group Members:

Chong Kah Wei (A20EC0027)

Heong Yi Qing (A20EC0043)

Mek Zhi Qing (A20EC0077)

Zereen Teo Huey Huey (A20EC0173)

Semester 2 2023/2024

1. According to assignment 2, list out the selected FOUR hyper parameters and its corresponding values. Summary on combination of hyper parameters that generate the most improved result.

The four hyperparameters chosen are loss function, optimizer, learning rate of the optimizer, and the number of epochs during the training process. Table 1 shows the hyperparameter with their corresponding values.

Hyperparameter	**Values**
Loss function	Mean Squared Error
	Binary Cross Entropy
Optimizer	Adam
	Lion
Learning rate of optimizer	0.01
	0.001
Number of Epochs	500
	1000

Table 1: Summary of hyperparameters and values

The combination of hyperparameters that generate the most improved results are obtained with the selection of Adam optimizer, a learning rate of 0.01, the Mean Squared Error (MSE) as the loss function, and 500 epochs. This combination of hyperparameters achieved a testing accuracy of 87.10%, precision of 93.33%, and recall of 82.35%. Table 2 summarizes the best combination of hyperparameters with its result.

Hyperparameter Values	Accuracy (%)	Precision (%)	Recall (%)
Loss function: Mean Squared Error	87.10	93.33	82.35
Optimizer: Adam			
Learning Rate: 0.01			
Epochs: 500			

Table 2: Best combination of hyperparameters and its result

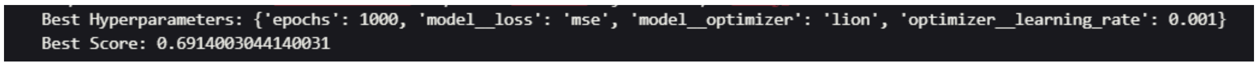
2. Perform a hyperparameter optimization using

a) Grid Search

b) Random Search

Then, screenshot the report (generated at the end of each run) that shows the best score and hyper parameter configuration that achieved the best performance.

Grid Search is one of the hyperparameter optimization techniques used by the researchers to find out the best combination of hyperparameters. It will work by exploring all the predefined set of values that have been given for different hyperparameters. The classification model will be trained and evaluated for each combination of hyperparameters. In this experiment, there are four predefined hyperparameters, each with two values. Hence, there will be sixteen combinations of hyperparameters to be explored by the Grid Search. The cross validation is set to three which means a three-fold cross validation was done to prevent the result becoming biased. The result of the Grid Search is shown in Figure 1.

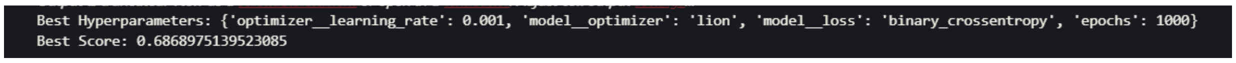


```
Best Hyperparameters: {'epochs': 1000, 'model__loss': 'mse', 'model__optimizer': 'lion', 'optimizer__learning_rate': 0.001}
Best Score: 0.6914003044140031
```

Figure 1: Best Combination of hyperparameters and scores for Grid Search

As shown in Figure 1, the best combination of hyperparameters are 1000 epochs, Mean Squared Error as the loss function, Lion optimizer, and 0.001 learning rate. These combinations get the highest average accuracy (score) which is 69.14% in the evaluation. The classification model will perform the best when these values are fixed to the parameters.

Other than that, the Random Search is performed to obtain an optimal result from the classification model. Instead of searching through all the possible combinations of hyperparameters predefined, Random Search will only select and test the combination randomly. The number of iteration is determined as ten where the Random Search should try and test ten combinations of hyperparameters before it returns the result. For evaluation purposes, a three-fold cross validation has been implemented followed by accuracy as the scoring metrics. Figure 2 shows the result of the Random Search.



```
Best Hyperparameters: {'optimizer__learning_rate': 0.001, 'model__optimizer': 'lion', 'model__loss': 'binary_crossentropy', 'epochs': 1000}
Best Score: 0.6868975139523085
```

Figure 2: Best Combination of hyperparameters and scores for Random Search

According to Figure 2, the best hyperparameter combinations are 0.001 learning rate, Lion optimizer, binary cross entropy as the loss function, and 1000 epochs. Through these combinations, the highest average accuracy (score) of the classification model built is 68.69%. The result of Random Search is vary for each time of running as the combination of hyperparameters are selected randomly.

3. Discuss the results obtained in (1) and (2).

The result from the previous assignment (looping through all the possible combinations) , Grid Search, and Random Search is tabulated in Table 3 for easier visualization and comparison.

Parameter	Result from Previous Assignment	Grid Search	Random Search
Optimizer	Adam	Lion	Lion
Learning Rate of Optimizer	0.01	0.001	0.001
Loss Function	Mean Squared Error	Mean Squared Error	Binary Cross Entropy
Number of Epochs	500	1000	1000
Testing Accuracy (%)	87.10	83.87	83.87

Precision (%)	93.33	92.86	92.86
Recall (%)	82.35	76.47	76.47

Table 3: Comparison between different types of method used

The method used in the previous assignment can be considered a manual search using a for-loop, where the best hyperparameter set is determined directly based on the classification performance of each model on the testing set. This approach can produce different results compared to GridSearchCV and RandomSearchCV, which use cross-validation to evaluate each hyperparameter combination by training and validating the model on multiple different subsets of the data. Based on Table 3, it is observed that using the Adam optimizer, a 0.01 learning rate, Mean Squared Error as the loss function, and 500 epochs, the classification model yields the best result. Conversely, Grid Search found that the best hyperparameters were the Lion optimizer, a 0.001 learning rate, Mean Squared Error loss function, and 1000 epochs, achieving the highest score of 69.14%. In contrast, Random Search concluded that the optimal configuration was the Lion optimizer, a 0.001 learning rate, Binary Cross Entropy as the loss function, and 1000 epochs, with a slightly lower best score of 68.69%. This indicates that the hyperparameters generated by Random Search might be less generalized compared to Grid Search. Nonetheless, when applying the models with these hyperparameters, all models achieved an accuracy above 80%, a considerably good result.

Make comparisons in terms of
a) effort to get the results

In Assignment 2, the hyperparameters are manually defined and iterated over. The effort to implement manual search is high as it requires manually specifying each combination and interpreting the results without automation tools. On the other hand, Grid Search uses libraries such as ‘GridSearchCV’ from scikit-learn to automate the exhaustive search over the hyperparameter space, which can evaluate all possible combinations. The effort to implement Grid Search is much lower than manual search as the setup effort is lower, but it requires moderate to high effort due to computational demands. For Random Search, it uses libraries like ‘RandomizedSearchCV’ to randomly sample combinations from the hyperparameter space which is more efficient than grid search, often finding good hyperparameters with fewer evaluations. The effort to apply the Random Search is lower than both Manual and Grid Search. Although Manual Search is less efficient compared to the automated methods like Grid and Random Search, the Manual Search still has its advantages on customized experimentation.

b) computational time

In terms of computational time, manual search is generally the most time-consuming approach, as it involves sequentially testing each manually selected combination of hyperparameters. This method's time requirement is highly variable and depends on the number of combinations chosen to test. Grid search, on the other hand, systematically evaluates all possible combinations of specified hyperparameters. While this ensures thorough exploration, it can be computationally expensive and time-consuming, especially with a large hyperparameter space. For instance, if testing 4 hyperparameters with 3 possible values each, grid search would evaluate 34=813^4 = 8134=81 combinations. Random search mitigates some of this inefficiency by sampling random combinations from the hyperparameter space. This often results in finding good hyperparameters faster, as it avoids evaluating every possible combination, thus reducing computational time significantly. Overall, manual search is the slowest, grid search is thorough but can be very time-consuming, and random search is the quickest while still finding good results.

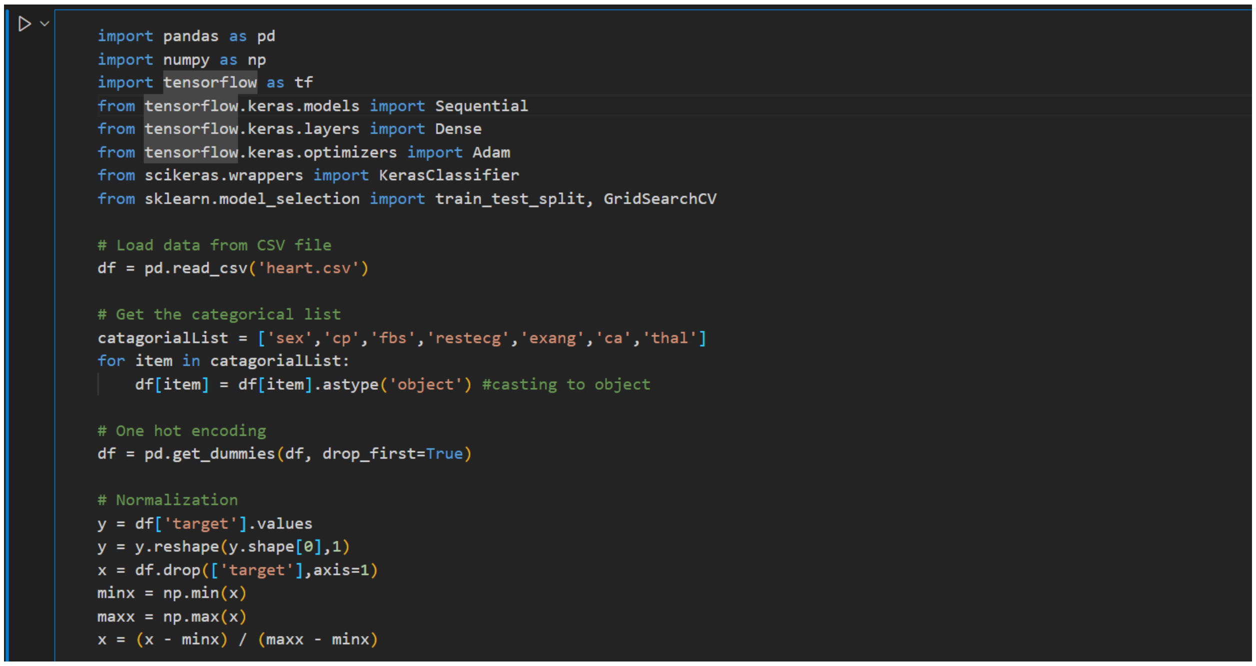
4. Highlight why hyperparameter optimization/tuning is vital in order to enhance your model’s performance?

Hyperparameter optimization is a process which helps to find the ideal set of parameters for a machine learning model in which the selection chosen at the end of this process is the most suitable parameter for the model. By undergoing the process, it can provide the best performance on the model. Besides, the hyperparameter optimization can help the model to generalize better to the unseen data. It can reduce the risk of overfitting and underfitting. The efficiency is also being improved through the hyperparameter. For example, the training time can be reduced by selecting an optimal batch size for training. Last but not least, every machine learning algorithm has their different hyperparameter to control various aspects of the learning process and model structure. A proper tuning can ensure that each algorithm adapts well to the specific problem and improve the accuracy.

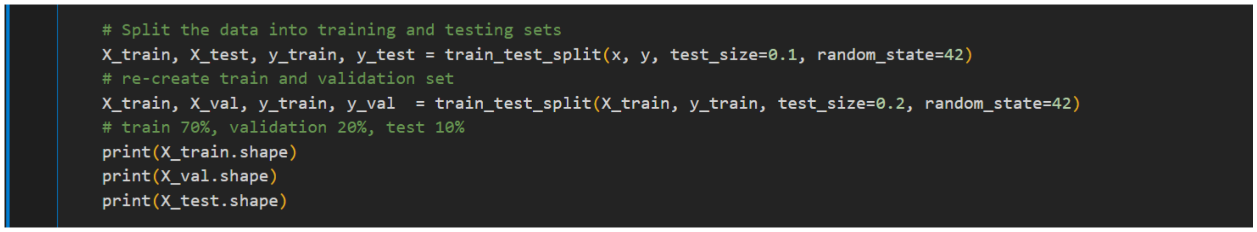
From the classification model, the hyperparameters chosen are loss function, optimizer, learning rate of the optimizer, and the number of epochs. The choice of loss function gives impacts on how the model penalizes error. Tuning this parameter can ensure the model uses the most appropriate loss function. Next, different optimizers also affect the convergence speed and final accuracy of the model. Hyperparameter optimization can help to determine which optimizer performs better for the given task. The learning rate is a hyperparameter which is used to control the step size during gradient descent updates. Therefore, an unsuitable learning rate will cause the model to converge too quickly or result in long training times. Lastly, the model can trains for an optimal amount of time by tuning the number of epochs which can prevent the overfitting and underfitting.

Appendix:

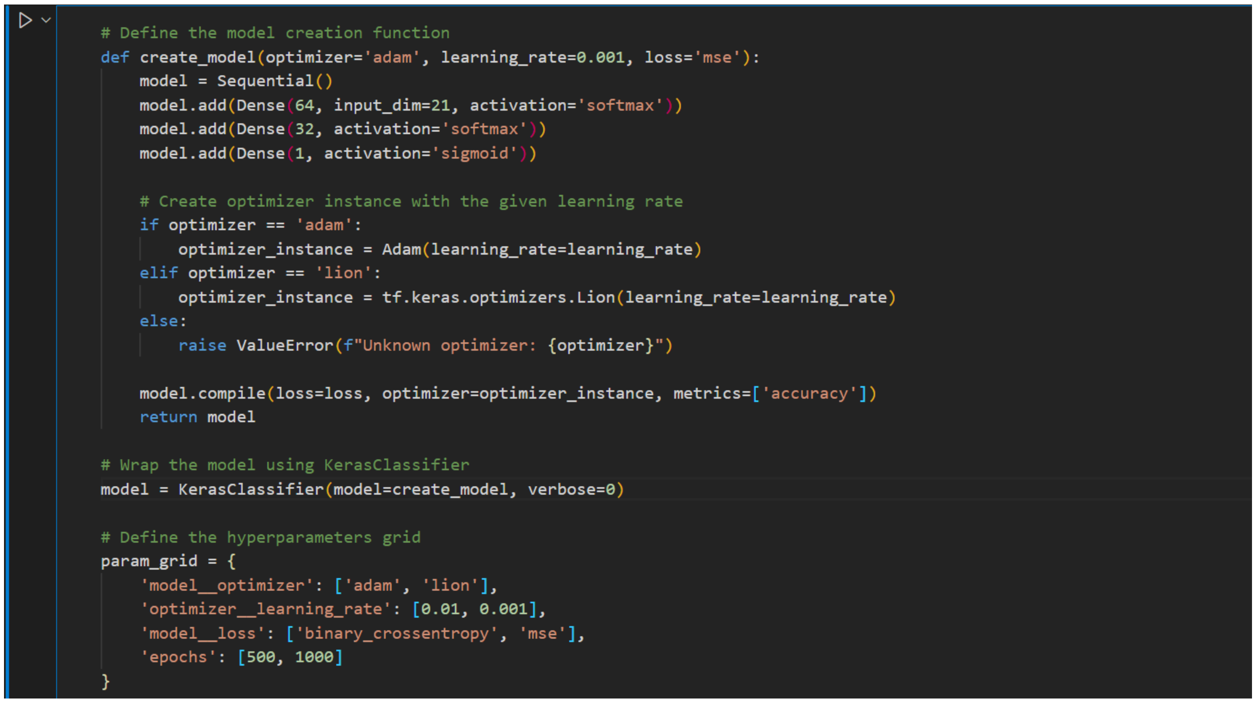
Grid Search



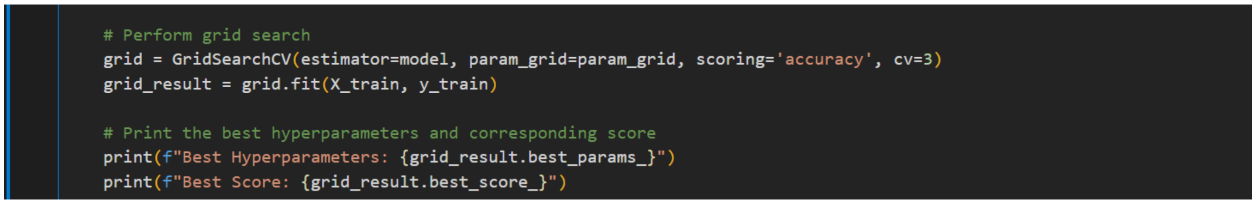
```python
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam
from scikeras.wrappers import KerasClassifier
from sklearn.model_selection import train_test_split, GridSearchCV

# Load data from CSV file
df = pd.read_csv('heart.csv')

# Get the categorical list
catagorialList = ['sex','cp','fbs','restecg','exang','ca','thal']
for item in catagorialList:
    df[item] = df[item].astype('object') #casting to object

# One hot encoding
df = pd.get_dummies(df, drop_first=True)

# Normalization
y = df['target'].values
y = y.reshape(y.shape[0],1)
x = df.drop(['target'],axis=1)
minx = np.min(x)
maxx = np.max(x)
x = (x - minx) / (maxx - minx)
```



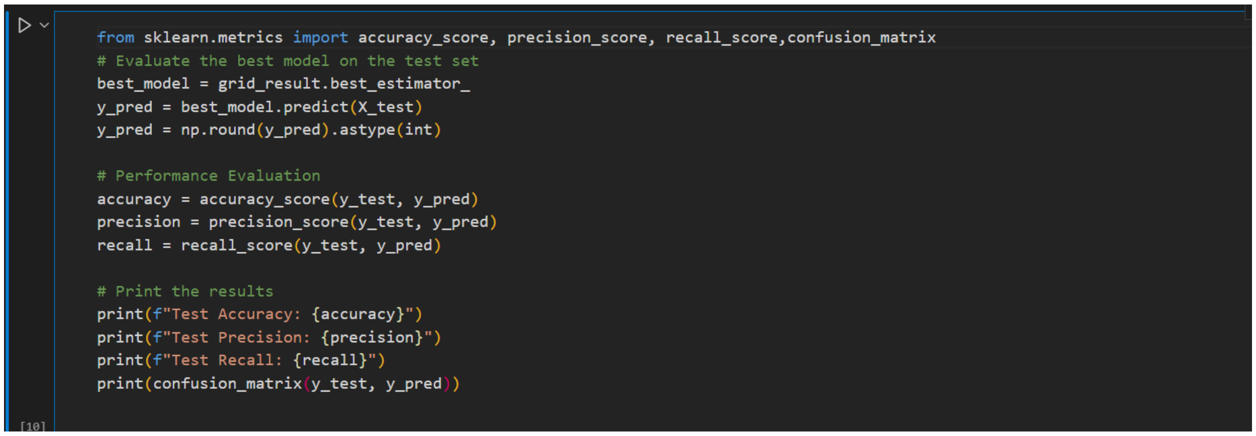
```python
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.1, random_state=42)
# re-create train and validation set
X_train, X_val, y_train, y_val  = train_test_split(X_train, y_train, test_size=0.2, random_state=42)
# train 70%, validation 20%, test 10%
print(X_train.shape)
print(X_val.shape)
print(X_test.shape)
```



```python
# Define the model creation function
def create_model(optimizer='adam', learning_rate=0.001, loss='mse'):
    model = Sequential()
    model.add(Dense(64, input_dim=21, activation='softmax'))
    model.add(Dense(32, activation='softmax'))
    model.add(Dense(1, activation='sigmoid'))

    # Create optimizer instance with the given learning rate
    if optimizer == 'adam':
        optimizer_instance = Adam(learning_rate=learning_rate)
    elif optimizer == 'lion':
        optimizer_instance = tf.keras.optimizers.Lion(learning_rate=learning_rate)
    else:
        raise ValueError(f"Unknown optimizer: {optimizer}")

    model.compile(loss=loss, optimizer=optimizer_instance, metrics=['accuracy'])
    return model

# Wrap the model using KerasClassifier
model = KerasClassifier(model=create_model, verbose=0)

# Define the hyperparameters grid
param_grid = {
    'model__optimizer': ['adam', 'lion'],
    'optimizer__learning_rate': [0.01, 0.001],
    'model__loss': ['binary_crossentropy', 'mse'],
    'epochs': [500, 1000]
}
```

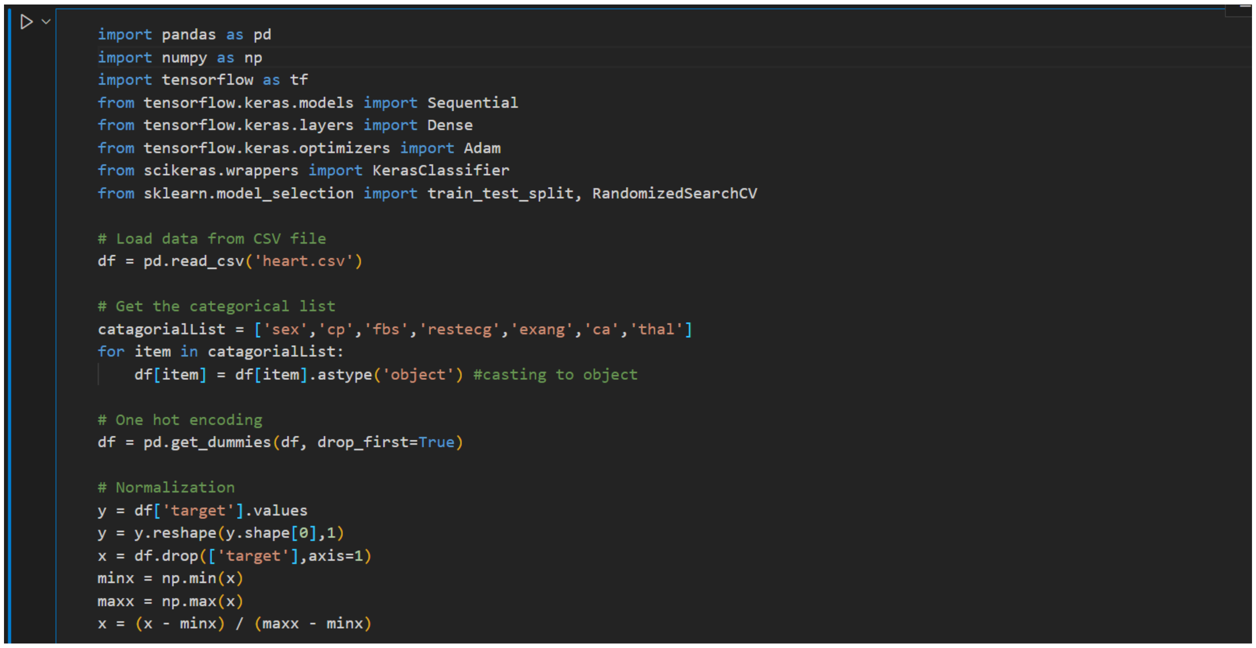


```python
# Perform grid search
grid = GridSearchCV(estimator=model, param_grid=param_grid, scoring='accuracy', cv=3)
grid_result = grid.fit(X_train, y_train)

# Print the best hyperparameters and corresponding score
print(f"Best Hyperparameters: {grid_result.best_params_}")
print(f"Best Score: {grid_result.best_score_}")
```

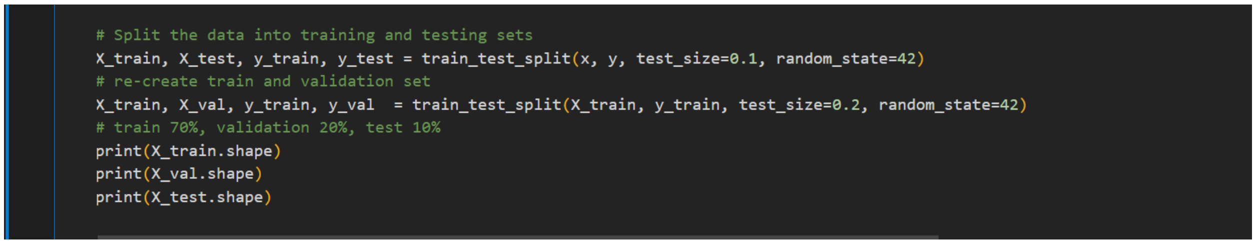


```python
from sklearn.metrics import accuracy_score, precision_score, recall_score,confusion_matrix
# Evaluate the best model on the test set
best_model = grid_result.best_estimator_
y_pred = best_model.predict(X_test)
y_pred = np.round(y_pred).astype(int)

# Performance Evaluation
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)

# Print the results
print(f"Test Accuracy: {accuracy}")
print(f"Test Precision: {precision}")
print(f"Test Recall: {recall}")
print(confusion_matrix(y_test, y_pred))
```

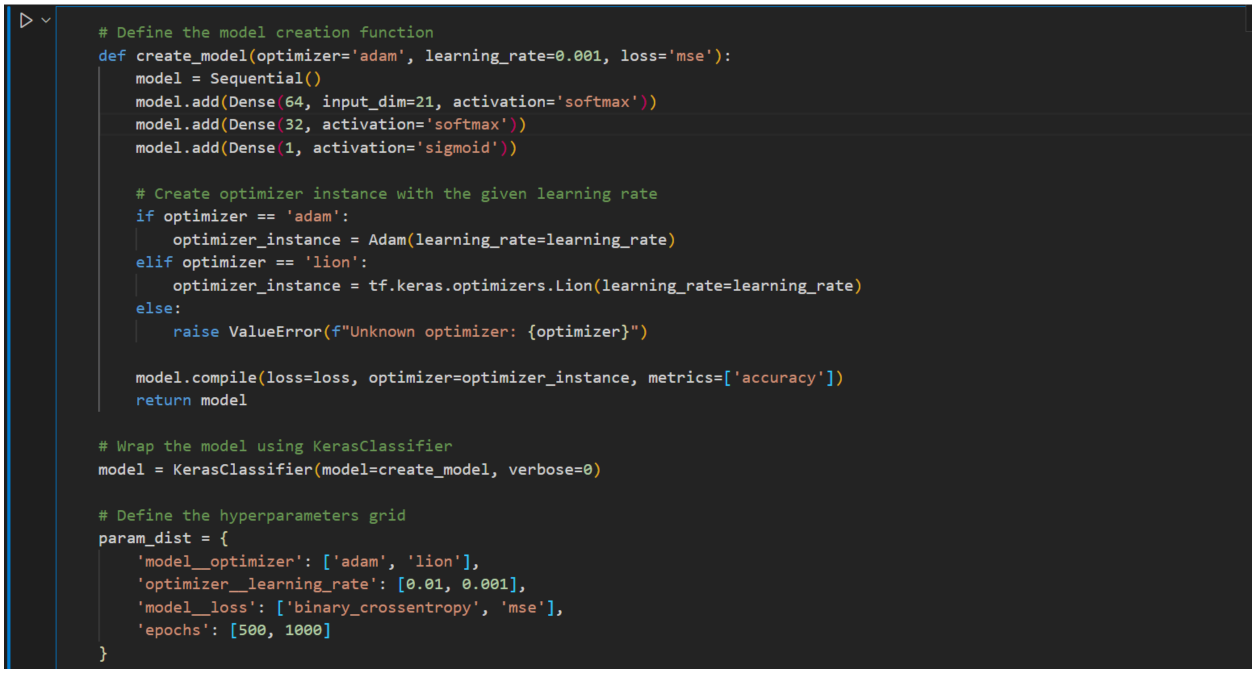
Random Search



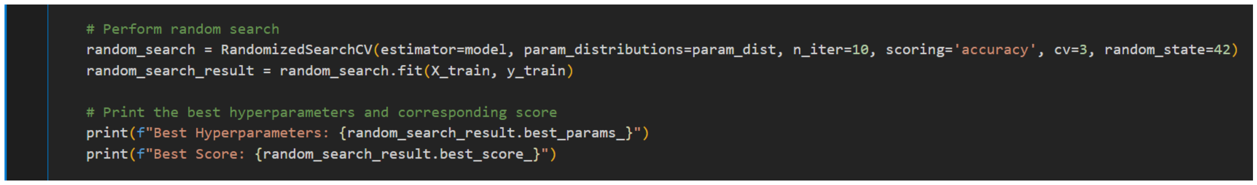
```
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam
from scikeras.wrappers import KerasClassifier
from sklearn.model_selection import train_test_split, RandomizedSearchCV

# Load data from CSV file
df = pd.read_csv('heart.csv')

# Get the categorical list
catagorialList = ['sex','cp','fbs','restecg','exang','ca','thal']
for item in catagorialList:
    df[item] = df[item].astype('object') #casting to object

# One hot encoding
df = pd.get_dummies(df, drop_first=True)

# Normalization
y = df['target'].values
y = y.reshape(y.shape[0],1)
x = df.drop(['target'],axis=1)
minx = np.min(x)
maxx = np.max(x)
x = (x - minx) / (maxx - minx)
```



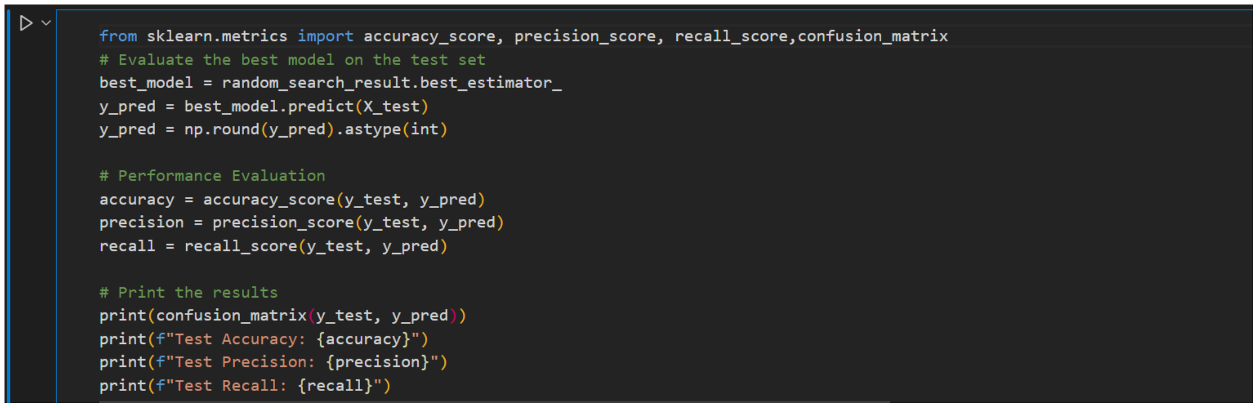
```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.1, random_state=42)
# re-create train and validation set
X_train, X_val, y_train, y_val  = train_test_split(X_train, y_train, test_size=0.2, random_state=42)
# train 70%, validation 20%, test 10%
print(X_train.shape)
print(X_val.shape)
print(X_test.shape)
```



```
# Define the model creation function
def create_model(optimizer='adam', learning_rate=0.001, loss='mse'):
    model = Sequential()
    model.add(Dense(64, input_dim=21, activation='softmax'))
    model.add(Dense(32, activation='softmax'))
    model.add(Dense(1, activation='sigmoid'))

    # Create optimizer instance with the given learning rate
    if optimizer == 'adam':
        optimizer_instance = Adam(learning_rate=learning_rate)
    elif optimizer == 'lion':
        optimizer_instance = tf.keras.optimizers.Lion(learning_rate=learning_rate)
    else:
        raise ValueError(f"Unknown optimizer: {optimizer}")

    model.compile(loss=loss, optimizer=optimizer_instance, metrics=['accuracy'])
    return model

# Wrap the model using KerasClassifier
model = KerasClassifier(model=create_model, verbose=0)

# Define the hyperparameters grid
param_dist = {
    'model__optimizer': ['adam', 'lion'],
    'optimizer__learning_rate': [0.01, 0.001],
    'model__loss': ['binary_crossentropy', 'mse'],
    'epochs': [500, 1000]
}
```



```python
# Perform random search
random_search = RandomizedSearchCV(estimator=model, param_distributions=param_dist, n_iter=10, scoring='accuracy', cv=3, random_state=42)
random_search_result = random_search.fit(X_train, y_train)

# Print the best hyperparameters and corresponding score
print(f"Best Hyperparameters: {random_search_result.best_params_}")
print(f"Best Score: {random_search_result.best_score_}")
```



```python
from sklearn.metrics import accuracy_score, precision_score, recall_score,confusion_matrix
# Evaluate the best model on the test set
best_model = random_search_result.best_estimator_
y_pred = best_model.predict(X_test)
y_pred = np.round(y_pred).astype(int)

# Performance Evaluation
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)

# Print the results
print(confusion_matrix(y_test, y_pred))
print(f"Test Accuracy: {accuracy}")
print(f"Test Precision: {precision}")
print(f"Test Recall: {recall}")
```