



UTM
UNIVERSITI TEKNOLOGI MALAYSIA

SECB 4313 BIOINFORMATICS MODELING & SIMULATION

ASSIGNMENT 3

LECTURER: DR. AZURAH BINTI A SAMAH

SECTION 1


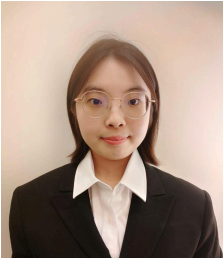


TEAM MEMBERS:

NO.	NAME	MATRIC NUMBER
1.	FELICIA CHIN HUI FEN	A20EC0037
2.	GOH YITIAN	A20EC0038
3.	PHANG CHENG YI	A20EC0131
4.	MOHD FIRDAUS BIN ZAMRI	A20EC0080

Table of Contents

1. Group Profile.....	3
2. Results Summary.....	4
3. Hyperparameter Optimization Algorithm (Grid Search and Random Search).....	5
4. Discussion.....	6
5. Importance of Hyperparameter Optimization/Tuning.....	7
6. Appendix.....	8

1. Group Profile

No.	Group Member Name	Picture	Roles
1.	FELICIA CHIN HUI FEN		<ul style="list-style-type: none"> • Execute the coding for manually finding hyperparameters. • Tabulate the results of the hyperparameter tuning proposed. • Perform Grid Search for hyperparameter tuning.
2.	GOH YITIAN		<ul style="list-style-type: none"> • Analyse the result of the hyperparameter tuning proposed. • Explain the importance of Hyperparameter Optimization/Tuning.
3.	PHANG CHENG YI		<ul style="list-style-type: none"> • Analyse the result of the hyperparameter tuning proposed. • Perform Random Search for hyperparameter tuning.
4.	MOHD FIRDAUS BIN ZAMRI		<ul style="list-style-type: none"> • Construct the tree diagram of the proposed experiment. • Tabulate the proposed experiment design. • Discussion on the findings based on the results.

2. Results Summary

No.	Activation Function	Neurons in Layer	Learning Rate	Batch Size	Model Loss	Accuracy	Precision	Recall	F1-Score
1.	'relu' for 1st layer	128 for 1st layer	0.001	32	0.15	0.84	0.84	0.84	0.84
2.	'relu' for 1st layer	128 for 1st layer	0.001	64	0.15	0.84	0.84	0.84	0.84
3.	'relu' for 1st layer	128 for 1st layer	0.0001	32	0.24	0.55	0.28	0.50	0.36
4.	'relu' for 1st layer	128 for 1st layer	0.0001	64	0.25	0.55	0.28	0.50	0.36
5.	'relu' for 1st layer	64 for 2nd layer	0.001	32	0.15	0.84	0.84	0.84	0.84
6.	'relu' for 1st layer	64 for 2nd layer	0.001	64	0.14	0.84	0.84	0.84	0.84
7.	'relu' for 1st layer	64 for 2nd layer	0.0001	32	0.25	0.55	0.28	0.50	0.36
8.	'relu' for 1st layer	64 for 2nd layer	0.0001	64	0.25	0.55	0.28	0.50	0.36
9.	'relu' for 1st, 2nd layers	128 for 1st layer	0.001	32	0.18	0.77	0.79	0.79	0.77
10	'relu' for 1st, 2nd layers	128 for 1st layer	0.001	64	0.16	0.77	0.78	0.77	0.77
11	'relu' for 1st, 2nd layers	128 for 1st layer	0.0001	32	0.17	0.84	0.84	0.84	0.84
12	'relu' for 1st, 2nd layers	128 for 1st layer	0.0001	64	0.18	0.81	0.81	0.81	0.81
13	'relu' for 1st, 2nd layers	64 for 2nd layer	0.001	32	0.17	0.71	0.71	0.71	0.71

14	'relu' for 1st, 2nd layers	64 for 2nd layer	0.001	64	0.17	0.77	0.77	0.78	0.78
15	'relu' for 1st, 2nd layers	64 for 2nd layer	0.0001	32	0.17	0.81	0.81	0.81	0.81
16	'relu' for 1st, 2nd layers	64 for 2nd layer	0.0001	64	0.18	0.81	0.81	0.81	0.81

3. Hyperparameter Optimization Algorithm (Grid Search and Random Search)

a. Grid search:

```

Best: 0.830811 using {'batch_size': 32, 'model_activation1': 'relu', 'model_activation2': 'relu', 'model_neurons1': 64, 'model_neurons2': 64, 'optimizer_learning_rate': 0.0001}
Model 1: 0.811074 (0.860530) with: {'batch_size': 32, 'model_activation1': 'relu', 'model_activation2': 'softmax', 'model_neurons1': 128, 'model_neurons2': 32, 'optimizer_learning_rate': 0.0001}
Model 2: 0.796899 (0.858303) with: {'batch_size': 32, 'model_activation1': 'relu', 'model_activation2': 'softmax', 'model_neurons1': 128, 'model_neurons2': 32, 'optimizer_learning_rate': 0.0001}
Model 3: 0.792027 (0.865642) with: {'batch_size': 32, 'model_activation1': 'relu', 'model_activation2': 'relu', 'model_neurons1': 128, 'model_neurons2': 32, 'optimizer_learning_rate': 0.0001}
Model 4: 0.782835 (0.849232) with: {'batch_size': 32, 'model_activation1': 'relu', 'model_activation2': 'relu', 'model_neurons1': 128, 'model_neurons2': 32, 'optimizer_learning_rate': 0.0001}
Model 5: 0.796456 (0.881510) with: {'batch_size': 64, 'model_activation1': 'relu', 'model_activation2': 'softmax', 'model_neurons1': 128, 'model_neurons2': 32, 'optimizer_learning_rate': 0.0001}
Model 6: 0.797010 (0.844666) with: {'batch_size': 64, 'model_activation1': 'relu', 'model_activation2': 'softmax', 'model_neurons1': 128, 'model_neurons2': 32, 'optimizer_learning_rate': 0.0001}
Model 7: 0.820377 (0.883859) with: {'batch_size': 64, 'model_activation1': 'relu', 'model_activation2': 'relu', 'model_neurons1': 128, 'model_neurons2': 32, 'optimizer_learning_rate': 0.0001}
Model 8: 0.801550 (0.860369) with: {'batch_size': 64, 'model_activation1': 'relu', 'model_activation2': 'relu', 'model_neurons1': 128, 'model_neurons2': 32, 'optimizer_learning_rate': 0.0001}
Model 9: 0.801440 (0.875420) with: {'batch_size': 32, 'model_activation1': 'relu', 'model_activation2': 'softmax', 'model_neurons1': 64, 'model_neurons2': 64, 'optimizer_learning_rate': 0.001}
Model 10: 0.810963 (0.872930) with: {'batch_size': 32, 'model_activation1': 'relu', 'model_activation2': 'softmax', 'model_neurons1': 64, 'model_neurons2': 64, 'optimizer_learning_rate': 0.0001}
Model 11: 0.768328 (0.870818) with: {'batch_size': 32, 'model_activation1': 'relu', 'model_activation2': 'relu', 'model_neurons1': 64, 'model_neurons2': 64, 'optimizer_learning_rate': 0.001}
Model 12: 0.830811 (0.892354) with: {'batch_size': 32, 'model_activation1': 'relu', 'model_activation2': 'relu', 'model_neurons1': 64, 'model_neurons2': 64, 'optimizer_learning_rate': 0.0001}
Model 13: 0.806202 (0.856175) with: {'batch_size': 64, 'model_activation1': 'relu', 'model_activation2': 'softmax', 'model_neurons1': 64, 'model_neurons2': 64, 'optimizer_learning_rate': 0.001}
Model 14: 0.801550 (0.860369) with: {'batch_size': 64, 'model_activation1': 'relu', 'model_activation2': 'softmax', 'model_neurons1': 128, 'model_neurons2': 64, 'optimizer_learning_rate': 0.0001}
Model 15: 0.815504 (0.882460) with: {'batch_size': 64, 'model_activation1': 'relu', 'model_activation2': 'relu', 'model_neurons1': 64, 'model_neurons2': 64, 'optimizer_learning_rate': 0.001}
Model 16: 0.792137 (0.866952) with: {'batch_size': 64, 'model_activation1': 'relu', 'model_activation2': 'relu', 'model_neurons1': 64, 'model_neurons2': 64, 'optimizer_learning_rate': 0.0001}
Best Model is Model 12 with score 0.830811

```

The best score shown by grid search is experiment 15. The model is built with 'relu' for the first and second layer, number of neurons is 64, learning rate is 0.0001, and batch size is 32.

b. Random Search:

```

Best: 0.837931 using {'optimizer_learning_rate': 0.001, 'model_neurons2': 32, 'model_neurons1': 128, 'model_activation2': 'relu', 'model_activation1': 'relu', 'batch_size': 32}
Model 1: 0.809606 (0.054191) with: {'optimizer_learning_rate': 0.0001, 'model_neurons2': 64, 'model_neurons1': 64, 'model_activation2': 'softmax', 'model_activation1': 'relu', 'batch_size': 64}
Model 2: 0.802709 (0.036719) with: {'optimizer_learning_rate': 0.0001, 'model_neurons2': 64, 'model_neurons1': 64, 'model_activation2': 'relu', 'model_activation1': 'relu', 'batch_size': 32}
Model 3: 0.802956 (0.041603) with: {'optimizer_learning_rate': 0.0001, 'model_neurons2': 64, 'model_neurons1': 64, 'model_activation2': 'softmax', 'model_activation1': 'relu', 'batch_size': 32}
Model 4: 0.803202 (0.060077) with: {'optimizer_learning_rate': 0.0001, 'model_neurons2': 32, 'model_neurons1': 128, 'model_activation2': 'softmax', 'model_activation1': 'relu', 'batch_size': 32}
Model 5: 0.837931 (0.036621) with: {'optimizer_learning_rate': 0.001, 'model_neurons2': 32, 'model_neurons1': 128, 'model_activation2': 'relu', 'model_activation1': 'relu', 'batch_size': 32}
Model 6: 0.802956 (0.026653) with: {'optimizer_learning_rate': 0.0001, 'model_neurons2': 32, 'model_neurons1': 128, 'model_activation2': 'relu', 'model_activation1': 'relu', 'batch_size': 32}
Model 7: 0.816749 (0.061524) with: {'optimizer_learning_rate': 0.001, 'model_neurons2': 64, 'model_neurons1': 64, 'model_activation2': 'softmax', 'model_activation1': 'relu', 'batch_size': 32}
Model 8: 0.809606 (0.029347) with: {'optimizer_learning_rate': 0.001, 'model_neurons2': 64, 'model_neurons1': 64, 'model_activation2': 'relu', 'model_activation1': 'relu', 'batch_size': 64}
Model 9: 0.816749 (0.047845) with: {'optimizer_learning_rate': 0.001, 'model_neurons2': 64, 'model_neurons1': 64, 'model_activation2': 'softmax', 'model_activation1': 'relu', 'batch_size': 64}
Model 10: 0.788670 (0.059871) with: {'optimizer_learning_rate': 0.001, 'model_neurons2': 32, 'model_neurons1': 128, 'model_activation2': 'softmax', 'model_activation1': 'relu', 'batch_size': 64}
Best Model is Model 5 with score 0.837931

```

The best score shown by random search is experiment 9. The model is built with 'relu' for the first and second layer, 128 neurons for the first layer, learning rate of 0.001 and batch size of 32.

4. Discussion

In terms of the effort to obtain the findings and the computational time involved, there are a number of significant differences and similarities between the manual search in (2), grid search and random search approaches used in (3).

For the manual search, the model is trained and the parameters are manually set for every experiment. This method is highly labour-intensive because it requires building and evaluating each model configuration individually. The manual search produced the best model for our project, with an F1-score of 0.84 and accuracy, precision, and recall of 0.84. The model made use of 64 neurons in the second layer, a 'relu' activation function in the first layer, a learning rate of 0.001, and a batch size of 64. The main advantages of the manual search are its simplicity and complete control over the parameters. However, its major drawback is the significant effort and time required to try out different parameter combinations. Because manual involvement is required for every new configuration, this method is laborious and unsuitable for exploring a wide parameter space.

Grid search, on the other hand, automates the tuning of hyperparameters by methodically attempting each possible combination inside a given grid. The best grid search model is from experiment 15, using a 'relu' activation function for the first and second layer, 64 neurons, a learning rate of 0.0001, and a batch size of 32. While grid search reduces manual effort, it is also computationally expensive and time-consuming, with each run taking at least 30 minutes. Despite this, it ensures comprehensive coverage of the parameter space.

In the meantime, parameter combinations are chosen at random for analysis via random search. The best model from random search, experiment 9, had a 'relu' activation function for both layers, 128 neurons in the first layer, a learning rate of 0.001, and a batch size of 32. Random search reduces computational time and effort compared to grid search but may miss optimal settings due to its reliance on randomness. This is because it does not run all the combination models to find the best solution. In this case, only 10 models are built and run.

In conclusion, the manual search gives exact control over the model parameters, but it takes a lot of time and work. Grid search automates the parameter tuning process but is computationally expensive and time-consuming. While there is a

chance that random search will overlook ideal configurations, it offers a more economical and effective way to explore the parameter space than grid search. Each method has its trade-offs, and the choice of which to use depends on the specific constraints and goals of the project. For this project, random search is the best method. It strikes a good balance between reducing manual effort and computational time while still effectively exploring the hyperparameter space. Although it may miss some optimal settings, it is more efficient than grid search and less labour-intensive than manual search. The results for the best solution are different from manual search as in manual search, early callbacks are used to prevent overfitting but are not performed in grid search and random search.

5. Importance of Hyperparameter Optimization/Tuning

Hyperparameter optimization, including techniques like Grid Search and Random Search, is essential for enhancing the performance of machine learning models, particularly in deep learning. This process systematically adjusts the model's hyperparameters to identify the optimal configuration that maximises its performance.

Grid Search is an exhaustive method, it evaluates all possible combinations of hyperparameters within a defined set, ensuring a comprehensive exploration of the search space. Although computationally intensive, it guarantees the identification of the global optimum, leading to significant performance improvements and establishing strong baselines for comparison.

Conversely, Random Search selects random combinations of hyperparameters, offering a more efficient and scalable approach, particularly suitable for high-dimensional spaces. It is less resource-intensive and often yields comparable results by focusing on diverse combinations.

In the experiment above, hyperparameter tuning explored different configurations, including variations in activation functions, neurons per layer, learning rates, and batch sizes. This process not only improves model accuracy and generalisation but also optimises training efficiency and prevents overfitting. Thus, hyperparameter optimization is vital for developing robust, high-performing machine learning models.

6. Appendix

a. Grid Search:

```
! pip install scikeras
```

```
# Use scikit-learn to grid search the activation function
import numpy as np
import tensorflow as tf
from sklearn.model_selection import GridSearchCV
from keras import Sequential
from keras import layers
from scikeras.wrappers import KerasClassifier
```

```
# Function to create model, required for KerasClassifier
# Default values for activation function is softmax, learning rate is 0.01, based on the original model
# neurons1 default value is 64, while neurons 2 is 32.
def create_model(neurons1=64, neurons2=32, learn_rate=0.01, activation1='softmax', activation2='softmax'):
    model = Sequential()
    model.add(Dense(neurons1, input_shape=(21,), activation=activation1))
    model.add(Dense(neurons2, activation=activation2))
    model.add(Dense(1, activation='sigmoid'))
    # Compile model
    model.compile(loss='mse', optimizer=tf.keras.optimizers.Adam(learning_rate=learn_rate, beta_1=0.9, beta_2=0.999, epsilon=1e-07, amsgrad=False, name='Adam'), metrics=['acc'])
    return model

# create model
model = KerasClassifier(model=create_model, epochs=1000, verbose=0)

# define the grid search parameters
param_grid = [
    {'model_activation1': ['relu'], 'model_activation2': ['softmax', 'relu'], 'optimizer_learning_rate': [0.001, 0.0001], 'model_neurons1': [128], 'model_neurons2': [32], 'batch_size': [32, 64]},
    {'model_activation1': ['relu'], 'model_activation2': ['softmax', 'relu'], 'optimizer_learning_rate': [0.001, 0.0001], 'model_neurons1': [64], 'model_neurons2': [64], 'batch_size': [32, 64]}
]

# perform grid search
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=-1)
grid_result = grid.fit(x_train, y_train)
```

```
# summarize results
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']

best_model = None
best_score = 0

for i, (mean, stdev, param) in enumerate(zip(means, stds, params)):
    print("Model %d: %f (%f) with: %r" % (i+1, mean, stdev, param))
    if mean > best_score:
        best_score = mean
        best_model = i+1

print("Best Model is Model %d with score %f" % (best_model, best_score))
```


b. Random Search:

```
!pip install scikeras
```

```
# Use scikit-learn to perform random search
import numpy as np
import tensorflow as tf
import keras
import scikeras
from keras import Sequential
from sklearn.model_selection import RandomizedSearchCV
from scikeras.wrappers import KerasClassifier
```

```
def build_model(neurons1=64, neurons2=32, learning_rate=0.01, activation1='softmax', activation2='softmax'):
    model = Sequential()
    model.add(Dense(neurons1, input_dim=21, activation=activation1))
    model.add(Dense(neurons2, activation=activation2))
    model.add(Dense(1, activation='sigmoid'))
    model.summary()
    # compile model
    model.compile(loss='mse',
                  optimizer=tf.keras.optimizers.Adam(learning_rate=learning_rate, beta_1=0.9, beta_2=0.999, epsilon=1e-07, amsgrad=False, name='Adam'),
                  metrics=['acc'])

    return model

# define the grid search parameters
param_dist = [
    {'model_activation1': ['relu'], 'model_activation2': ['softmax', 'relu'], 'optimizer_learning_rate': [0.001, 0.0001], 'model_neurons1': [128], 'model_neurons2': [32], 'batch_size': [32, 64]},
    {'model_activation1': ['relu'], 'model_activation2': ['softmax', 'relu'], 'optimizer_learning_rate': [0.001, 0.0001], 'model_neurons1': [64], 'model_neurons2': [64], 'batch_size': [32, 64]}
]

# Wrapping the model using KerasClassifier
model = KerasClassifier(build_fn=build_model, epochs=1000, verbose=0)

# perform random search
random_search = RandomizedSearchCV(estimator=model, param_distributions=param_dist, n_jobs=-1)
random_result = random_search.fit(x_train, y_train)

# summarize results
print("Best: %f using %s" % (random_result.best_score_, random_result.best_params_))
means = random_result.cv_results_['mean_test_score']
stds = random_result.cv_results_['std_test_score']
params = random_result.cv_results_['params']

best_model = None
best_score = 0

for i, (mean, stdev, param) in enumerate(zip(means, stds, params)):
    print("Model %d: %f (%f) with: %s" % (i+1, mean, stdev, param))
    if mean > best_score:
        best_score = mean
        best_model = i+1

print("Best Model is Model %d with score %f" % (best_model, best_score))
```