Evaluating the Efficiency of Genetic Algorithm in Hyperparameter Tuning and Feature Selection for Casper Model When Applied to Predict Mobile Web Search Satisfaction

Anonymous

School of Computing, Australian National University
u7227871@anu.edu.au

Abstract. Genetic Algorithm is one of the evolutionary algorithms that can solve optimisation problems, including hyperparameter tuning and feature selection. This study explores the efficiency aspect of the Genetic Algorithm by calculating the number of computations required for the Genetic Algorithm to obtain the ground truth, i.e., the best hyperparameter combination or feature subsets within the search space that can be found using grid search. The model used for hyperparameter turning and feature selection is Casper, which is a constructive neutron network that can solve tasks ranging from simple to complex. The specific task the study is going to solve is the prediction of user satisfaction with mobile web searches based on the layout feature of the result pages, user behaviour features, and users' eye-tracking features. Through this study, it was found that the GA can indeed save a lot of computation time if we only aim to find one of the top 10 hyperparameter combinations or feature subsets within the search space. However, it will still require similar computations as the grid search to find the best one within the search space. Even though, using one of the top 10 hyperparameter combinations or feature selections should let the model perform reasonably well.

Keywords: web search, satisfaction, Casper, neuron network, genetic algorithm

1 Introduction

In the digital age, web searches have woven themselves into the very fabric of our daily routines. For search engine providers, ensuring that users have a satisfactory experience is essential. Traditionally, satisfaction can be obtained through user surveys, but this may face some challenges, such as prolonged feedback and dishonest answers from the participants. In addition to directly obtaining satisfaction through the user's opinion, satisfaction may also be estimated through correlated features, such as the layout of the result page and the user's behaviours. For example, Kim et al.'s study has found that having long snippets[1] increases users' search time but without increasing search accuracy when users are doing informational tasks[2], leading to worse user satisfaction [1]. Therefore, a regression model can be built that characterises the results page layout and user behaviour and outputs an estimated satisfaction number. One of the options for this regression model is Casper.

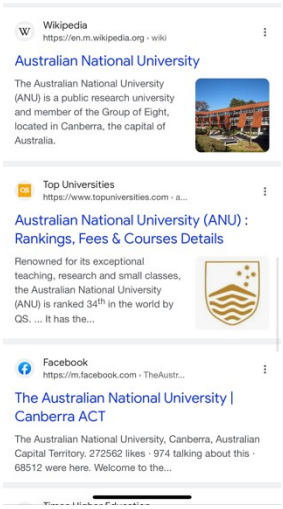
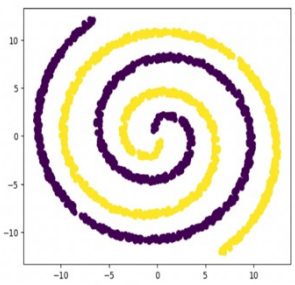


Fig. 1. Example of snippets on the result page, the last snippets are shorter than the first two.



Casper, proposed by Treadgold and Gedeon (1997) [2], is a constructive neuron network in which size and structure grow during the training process. This makes Casper a powerful tool for solving tasks varying from simple to complex because it can dynamically adjust the size of the network based on their complexity. One of the remarkable achievements of Casper is that it can solve the Two Spiral Problems, shown in Fig.2 to a high standard, with only 13 hidden neurons and 2437 epochs on average, which is difficult to achieve for general neuron networks using backpropagation. To train the Capser

1 The text used to summarise a retrieved webpage located under the title of each website on the result page, shown in Fig. 1

2 The search task to finds the information of something, e.g., "What are the side effects of Panadol?" [1]

model, the dataset from Kim et al.'s study is used, which contains 262 observations of web search activities. For each observation, the snippet size, user behaviour data, user eye-tracking data, and most importantly, user satisfaction are recorded. Therefore, the model can then be learned to predict user satisfaction using the other given features as inputs.

Fig. 2. Two Spirals Problem: the model should determine which spiral a point is in

During the training and validating process, hyperparameter tuning and feature selection are the two key steps to improve model quality. The hyperparameters control model behaviours and a model can perform well only if the hyperparameters are properly chosen. When given a dataset, some of the features may not be useful and they will potentially distract the model when it is learning. Therefore, feature selection is about finding the subset of features as the model input that leads to an optimal model performance. The genetic algorithm (GA) is one of the search techniques that can be used in both hyperparameter tuning and feature selection. It is an evolutionary algorithm widely used in optimisation tasks. Although its property of randomness means it cannot guarantee the best solution, it is often able to provide a satisfactory result in an acceptable time, which provides another option other than exhaustive search and random search when a problem-specific algorithm is not found.

This study will evaluate the performance of GA, mainly its efficiency, by applying it to search best hyperparameter combination and feature subset inputs for the Casper model used in web search satisfaction prediction. It will also compare GA with grid search and random search in terms of number After using GA, grid search, and random search to select the best hyperparameter combination and feature subset for the model. It will train the model 2000 times with different testing and training set split and weight initialisation. After that calculate the average performance of the model in terms of L1 loss and if the model predictions are aligned with the findings in the dataset paper. Section Two provides further background for the dataset, Casper, and GA. Section Three unfolds the methodology and implementation. Section Four summarises the experiment results and Section Six concludes the paper.

This study evaluates the performance of the Genetic Algorithm (GA), with a focus on its efficiency, by applying it to search for the optimal hyperparameter combination and feature subset inputs for the Casper model used in predicting web search satisfaction. The study further contrasts the GA with grid search and random search in terms of the computational cost[3]. Upon employing GA, grid search, and random search to determine the best hyperparameter combination and feature subset, the model undergoes training 2,000 times using different testing and training set splits and weight initializations. Subsequent analysis computes the average performance in terms of L1 loss and assesses the alignment of model predictions with findings from the original dataset paper. Section Two delves into the background of the dataset, the Casper model, and GA. Section Three outlines the goals and methodology. Section Four records the implementation. Section Five presents a summary of the experimental results, and Section Six offers conclusions drawn from the study.

2 Background

2.1 Dataset

The dataset collected by Kim et al. [1] provides many useful features for training the model. The primary objective of the paper is to investigate the influence of different snippet sizes on several aspects, including users' satisfaction, confidence, and performance when doing web searches. Even though, the dataset can be viewed from many different angles, such as how users' confidence and performance and snippet sizes affect users' satisfaction. Therefore, it is a suitable dataset to train a model to predict users' satisfaction with mobile web searches. The satisfaction is recorded as an integer ranging from 1 to 7. The snippet size varies from: short, medium, and long. The dataset was collected

[3] The number of times the search algorithm asks validation function for a score of the model trained by a specific hyperparameter combination or a subset of features.

by having 24 participants each complete 12 different web search tasks. The 12 tasks differ by their task type (either informational or navigational tasks) and snippet size shown on the result page.

During each task, the users' normal behaviours, such as the time it takes to read the search result, and the eye-tracking features are also recorded. As shown in Fig. 3, among all the columns in the dataset, Columns [B] and [C] are task-related features, columns [E] to [L] are users' behavioural features (where [I] will be used as the label for training), and columns [M] to [Z] are eye-tracking features.

```
[A] Subject      [B] Task_type     [C] Snippet_length   [D] Task_num     [E] Shown_T_num       [F] Time to first click
[G] Log(TTF)     [H] Accuracy      [I] Satisfaction     [J] scroll       [K] viewport          [L] Clicked_rank
[M] StartTime    [N] Endtime_SERP  [O] Fixation _title  [P] Fixation_URL [Q] Fixation_snippet  [R] Fixation_total
[S] Log(F_title) [T] Log(F_URL)    [U] Log(F_Snippet)   [V] Log(F_total) [W] Original_scanpath_value
[X] Compressed value               [Y] Minimal value    [Z] comp-mini
```

Fig. 3. All the columns in the dataset

2.2 Casper

The model used for predicting user satisfaction is Casper. Casper is a constructive neuron network that gradually increases its number of hidden neurons during training. The algorithm for training a Casper is shown in the following.

- The network initialises a minimal structure in which there is only an input layer connecting to an output layer.
- Train the network until the difference between the current error and the error from the previous epoch is less than 1%. This condition should persist for a specified number of epochs, determined by the formula in (1), where P is defined by programmers and N represents the number of currently installed neurons."
- Add a new neuron connecting to all input neurons and hidden neurons in lower layers and the output neuron, as shown in Figure 2. The newly added weights should be initialised but the old weights in the network remain unchanged.
- Repeat 2 until the training result is satisfied.

$$Number\ \ of\ \ epochs\ \ required\ = 15 + P * N \quad \textbf{(1).}$$

When updating the weights given the loss function and its gradient, a modified version of resilient backpropagation (R-prop) is used. R-prop is proposed by Riedmiller and Braun in 1993 [3]. Compared to normal backpropagation that updates weights based on their derivatives, weights are updated based on their signs and dynamic step size. As shown in (2), the sign of a weight update Δ_{ij} depends on the current derivative of the error function with respect to the weight, the amount of Δ_{ij} depends on the current derivative and the previous derivative, i.e., if the previous derivative and the current derivative have the same sign, increase the amount of Δ_{ij} and vice versa.

$$\Delta_{ij}^{(t)} = \begin{cases} \eta^+ \cdot \Delta_{ij}^{(t-1)}, & \text{if } \frac{\partial E^{(t)}}{\partial w_{ij}} \cdot \frac{\partial E^{(t-1)}}{\partial w_{ij}} > 0 \\ \eta^- \cdot \Delta_{ij}^{(t-1)}, & \text{if } \frac{\partial E^{(t)}}{\partial w_{ij}} \cdot \frac{\partial E^{(t-1)}}{\partial w_{ij}} < 0 \\ \Delta_{ij}^{(t-1)}, & \text{otherwise} \end{cases} \qquad w_{ij}^{(t)} = \begin{cases} -\Delta_{ij}^{(t)}, & \text{if } \frac{\partial E^{(t)}}{\partial w_{ij}} > 0 \\ +\Delta_{ij}^{(t)}, & \text{if } \frac{\partial E^{(t)}}{\partial w_{ij}} < 0 \\ 0, & \text{otherwise} \end{cases} \quad \textbf{(3).}$$

Casper also makes use of the regularisation technique, Simulated Annealing, to improve the generalisation of the network. As shown (4), Simulated Annealing makes the amount of weight decay proportional to its squares [4], where Hepoch is the number of epochs passed since the addition of the last hidden neuron.

$$\frac{\delta E}{\delta w_{ij}} = \frac{\delta E}{\delta w_{ij}} - k \cdot sign\left(w_{ij}\right) \cdot {w_{ij}}^2 \cdot 2^{-0.01 * Hepoch} \quad \textbf{(4).}$$

In addition to R-prop and weight decay, the Casper optimiser also applies different learning rates to 3 different parts of the network. As shown in Figure 2, L1, the weights connecting the last hidden neuron and input neurons to the new hidden neurons should learn the fastest, followed by L2, the weights connecting the last hidden neurons to the output neurons, and the rest of the weights, L3.

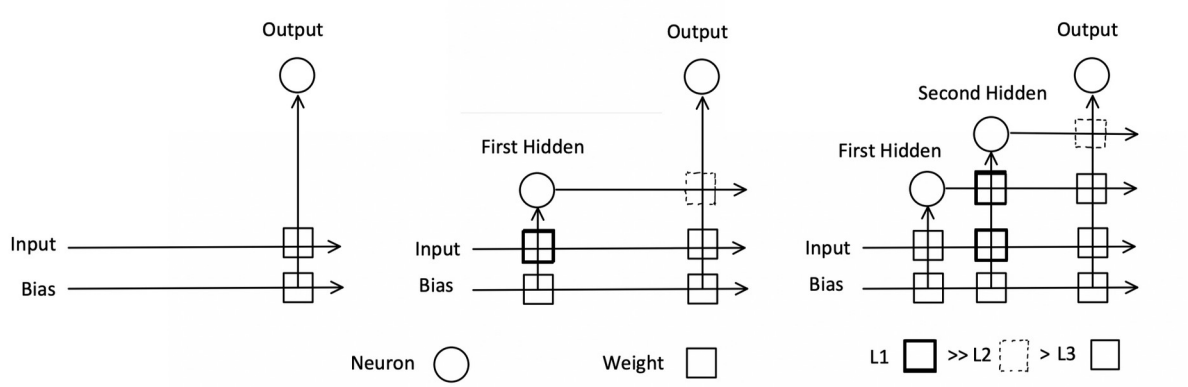


Fig. 4. Initial network, network after one hidden neuron is added, and network after two hidden neurons are added.

2.3 Genetic Algorithm

The genetic algorithm is a type of evolutionary algorithm that can be used to solve optimisation problems, such as feature selection and hyperparameter tuning [5]. In a genetic algorithm, individuals in a search space are encoded as chromosomes, which can be a string of characters/integers/float/bits. For example, if there are 6 features in a dataset and binary chromosomes are used to encode every possible subset, then a chromosome “0001001” can indicate a subset that only contains the fourth and the sixth column. For hyperparameter tuning, if there are 4 hyperparameters to fine-tune and each hyperparameter contains 8 candidates, a 12-bit chromosome can be used to represent an individual hyperparameter combination, where each hyperparameter uses 3 bits to store. In addition, an integer or float chromosome can also be used to represent a larger search space, and in this case, one gene should be enough to represent a hyperparameter within an infinite search space. After encoding information into chromosomes, the GA begins with the following steps[4].

1. Randomly initialise the first generation.
2. Calculate a fitness table that maps each individual to a fitness score. In this study, the fitness score is the mean squared error (MSE) of the model trained using the selected feature subset or hyperparameter combination. The lower the better the chromosome.
3. Based on the fitness table, find out the top chromosomes and directly copy them into the next generation without crossover and mutation.
4. Select two parents from the population using a selection method. In this study, tournament selection is used as the MSE of individual are close. In tournament selection, a number of individuals are randomly chosen from the population and best one (the one with the lowest MSE) will return.
5. Perform crossover on the parents' chromosomes to produce offspring. In this study, single-point crossover is utilized, and two parents typically produce two children. However, there is a possibility that the two parents may not undergo crossover, resulting in offspring where one child is identical to one parent, and the other child is identical to the other parent.
6. A child may mutate based on some rate. In this study, mutation is simply flipping 0 and 1 bits in a chromosome.
7. Keep selecting parents and spawning children until the new population size is satisfied.
8. Replace the current population set with the new population set. Go back to 2. for the next generation.
9. Stop until a satisfactory result is found or the maximum number of generations is reached.

[4] The algorithm may vary.

3 Goals and Methodology

3.1 Goals

Goal 1: Assess the efficiency of the Genetic Algorithm (GA) in hyperparameter tuning.

Method: This will be achieved by determining the average number of generations required to identify one of the top 10 hyperparameter combinations and the best hyperparameter combinations within GA's search space. When accessing Goal 1, all features in the pre-processed dataset will be used. To illustrate, consider a search space in GA tuning 4 hyperparameters, each with 8 potential values. This results in a search space size of 8^4 = 2^12 = 4096, which can be represented using a 12-bit chromosome. Let's assume the ground true optimal chromosome is represented as "000000000011" within the total 2^12 chromosomes and the top 10 chromosomes are catalogued in a list named "top_10". The aim is to find out the average number of generations GA takes to yield a chromosome present in the "top_10" list and the number of generations it takes to output the true optimal chromosome "000000000011".

Goal 2: Assess the efficiency of the Genetic Algorithm (GA) in feature selection.

Method: Similarly, this will be achieved by determining the average number of generations required to identify one of the top 10 feature subsets and the best feature subset within GA's search space. When accessing Goal 2, the best hyperparameter combination found during Goal 1 will be used and held constant.

To achieve Goals 1 and 2, the ground truth about the best and the top 10 hyperparameters combination or feature subsets should be obtained before accessing the efficiency of GA. Therefore, grid search will be used first to compute the scores (MSEs) of the models trained using all possible hyperparameter combinations. This will generate a fitness table with 2^12 = 4096 entries (assuming we are tuning 4 hyperparameters, each with 8 candidates) where each entry is a mapping between a chromosome string and its MSE. Similarly, grid search will compute all 2^15 = 32768 feature subsets (assuming there are 15 feature columns) and their corresponding MSEs and record it in another table.

By sorting the tables, the best and the top 10 chromosomes can be easily obtained for both hyperparameter tuning and feature selection, which serve as the ground truth for the GA efficiency assessment. Although building up these two tables can take a significant amount of time, after the tables are obtained, the GA can easily get the fitness for any chromosome by directly accessing the table built by grid search instead of calling the Casper function to obtain an MSE. This is because grid search has pre-calculated and stored all the possible chromosomes and scores within the search space of the GA. Therefore, the GA can run very fast and hence can run many times to obtain a better average result of the number of generations needed to find the top 10 and the best chromosomes.

Goal 3: Evaluate the model's efficacy post hyperparameter optimization and feature selection using Mean Squared Error (MSE) and Mean Absolute Error (MAE), also known as Mean L1 Error from the model's predictions.

Method: To minimise the effect of the randomness of the training and testing split and the model's weights initialisation on the model performance, the dataset splitting, training, and testing are repeated multiple times. Suppose that the model is trained and tested three times on different training and testing dataset split and three MSEs, 1.13, 1.14, and 1.15, are obtained. The final MSE of the model will be the average of them, which is 1.14.

Goal 4: Evaluate the model's efficacy by verifying that the behaviour of the model is aligned with the findings in the dataset data. The dataset paper has found that the longer the snippet size, the more time it takes the user to read, and the lower the user satisfaction. Therefore, the model should also be trained to assign low satisfaction scores for samples that have long snippet sizes.

Method: First split the dataset into training and testing sets. Then train the model on the training set. After, use the model to predict satisfactory scores for samples in the test set. Loop through all the samples, if its snippet size is long, add its predicted satisfactory score into the set "predicted_score_for_long_snippets", if its snippet size is median, add its predicted satisfactory score into the set "predicted_score_for_median_snippets", similarly for samples with short snippets. After this process, 6 sets are obtained (3 for predicted and 3 for ground truth).

Since the dataset paper claims that long snippet size leads to low satisfaction. The overall satisfactory score for median snippets and short snippets should be statistically higher than the overall satisfactory score for median snippets. This means if running p-tests between "predicted_score_for_long_snippets" and "predicted_score_for_median_snippets" and between "predicted_score_for_long_snippets" and "predicted_score_for_short_snippets" also result in very low p scores, then the model behaviour is aligned with the finding of the dataset paper and the model is effective.

3.2 Overall Experimental Procedure

1. Pre-process the original dataset.
2. Implement Casper in Pytorch 2.0
3. Implement an n-repeated k-fold validator for Casper that performs dataset splitting, training, testing, calculating MSE and MAE (n * k) times and outputs the average of the (n * k) MSEs and MAEs as the score of the model that is trained by a specific hyperparameter combination and feature subset.
4. Find the best hyperparameter combination using grid search.
5. Find the number of computations required for GA to yield the top 10 and the best hyperparameter found by grid search. **Complete Goal 1**.
6. Compare GA, grid search, and random search in terms of the number of computations required to find the best one and the top 10s.
7. Fix the hyperparameters of the model to the best hyperparameter found in the previous hyperparameter tuning steps during the following feature selection processes.
8. Find the best feature subset using grid search.
9. Find the number of computations required for GA to yield the top 10 and the best hyperparameter found by grid search. **Complete Goal 2**.
10. Compare GA, grid search, and random search in terms of the number of computations required to find the best one and the top 10s.
11. Choose a final model (a model that is configured to use the best hyperparameters and one of the top 10 feature subsets) and test it on a number of different training and testing splits and obtain the distribution of squared errors and absolute errors.
12. Evaluate the model based on the squared errors and absolute errors. **Complete Goal 3**
13. Verify that the model is able to give statistically lower satisfactory scores for inputs with long snippet sizes than inputs with short and median snippet sizes. **Complete Goal 4**.

4 Implementation and Results

All implementations mentioned in this section can be found in the source code associated with this paper. For example, several Jupyter Notebook files have recorded how data is processed step by step and how the best hyper-parameter of a model is selected Please refer to the readme file for details.

4.1 Data Pre-Processing

Relevant File: data_preprocessing.ipynb, data_preprocess_w_oversampling.ipynb, data_preprocess_wo_oversampling.ipynb, custom_smote() in utililties.py

Several data processes have been done before further model training and feature selection. (1) While the original dataset has provided 26 columns as shown in Fig 3, the columns such as "Subject" and

"Task_num" are highly likely not useful for model prediction and can be removed. (2) There are also redundant columns such as column TTF (Time to first click) and column log (TTF). For column a column that has its logarithmic transformation values in another column, remove and original column and keep the log one as the log transformation helps a single feature of data to become more normally distributed and normally distributed data is preferred by the machine learning model. (3) The categorical variables, such as nav/info in column "Task_Type" are encoded into non-negative integer numbers. (4) Finally, all column except the target column is normalised by subtracting their mean and dividing by their standard deviation. This ensures features with large number values are equally important as the features with small number values. (5) Let the first column be the target, i.e., satisfactory and the rest of the column be the feature. Fig. 5 shows the remaining column after the data pre-processing.

```
[A] Satisfaction  [B] Snippet_length  [C] Shown_T_num    [D] Log(TTF)       [E] Task_type      [F] scroll
[G] Clicked_rank  [H] StartTime       [I] Endtime_SERP   [J] Log(F_title)   [K] Log(F_URL)     [L] Log(F_Snippet)
[M] Original_scanpath_value           [N] Compressed value                  [O] Minimal value  [P] comp-mini
```

Fig. 5. Remaining columns after the dataset pre-processing

Key Finding 1

One of the findings during the data pre-processing is that the dataset is not balanced. Specifically, the number of samples for the 7 satisfactory scores/classes [1, 2, 3, 4, 5, 6, 7] are [2, 15, 34, 40, 77, 70, 23], respectively. The SMOTE oversampling technique has been tried to make the number of each class equal to the number of the biggest class. However, it was found that models trained with oversampling provided statistically higher MSE than models trained without oversampling. This finding is obtained by using a grid search to find the best hyperparameters for both the model with oversampling and the model without oversampling and then obtain the distribution of their squared error through 200 runs with different training and testing splits and model initialisation. The detailed process can be found in "data_preprocessing.ipynb", "data_preprocess_w_oversampling.ipynb".

4.2 Implement Casper in PyTorch 2.0

Relevant File: casper.py, casper_two_spirals_demo.ipynb

In order to have a changeable structure, in the class definition of Casper, the hidden layers are stored in a list. In a forward pass, the input is passed to the first hidden layer in the list and then keeps updating while transferring a list of hidden layers. In the end, the last hidden layer sends the result to the output layer to output. All hidden layers use sigmoid as their activation function. In the Casper model class, there is also a helper function that can add a neuron to the model when it is called.

The training function consists of two loops, the inner loop trains the model through a number of epochs and stops until the loss no longer decreases for some time. The outer loop manages the addition of hidden neurons into the model. After the inner loop is complete, the outer loop adds a neuron to the model and then updates the optimiser function as the number of trainable parameters has changed. The outer loop stops when the number of hidden neurons has exceeded a certain number.

The optimiser is written based on the formulae (3) and (4) in Section 2.2. Casper uses different learning for different parts of the model. Luckily, PyTorch supports making model weight parameters into groups and for each group a specific learning rate is assigned.

To demonstrate the correctness of the implementation of Caspor, the model is run on a classification problem, called two spirals, shown in Fig. 2, which consists of two groups of data points lying on two spirals, respectively. It was found that the model performed quite well on this problem with a high accuracy of 95.46% with a standard deviation of 4.48%, which is close to the benchmark result of 97.80% of the original paper. This suggests that the implementation of Casper is most likely correct.

The loss function used to train and validate the model is MSE. Although the predicted labels are discrete (integers from 1 to 7), the problem is essential a regression tasks, where MSE is used instead of cross-entropy loss. For example, if the prediction is 4 and the ground true is 5, in classification, it is considered totally wrong, but in regression problems, 4 is still somehow close to the ground true and should have a smaller loss than having a prediction of 3.

4.3 Implement Casper Validator

Relevant File: the function rkf_validator() in casper_test.py

The purpose of Casper validator is to assign a score (MSE or MAE) for the Casper model training by a specific hyperparameter combination of feature selection during the validation process. Therefore, whether or not this hyperparameter combination or feature selection is good can be known. It takes a training set and performs n-repeat k-fold validation, that is, the model is trained by testing using (n * k) different training and validating sets split and the average of MSE or MAE among these (n * k) runs are obtained. Since there are only 261 observations in the dataset. Having a separate pre-defined validation set may result in too little training data and the distribution of the validation set may not be similar to the distribution of the training. Therefore, the n-repeat k-fold validation is important.

The rkf_validator() can also be used in the final testing of the model after finding the best hyperparameters and feature subsets. In this case, the rkf_validator() accepts the whole dataset instead of the training set. Then it mimics that the model is trained and tested using different training and testing splits that give a more general result of the model performance that minimises the effect of randomness due to the differences in training and testing splits and weight initialisation.

When accessing the efficiency of the GA, grid search, and random search, the number of computations will be equal to the number of times the search algorithm call the rkf_validator() function.

4.4 Find the best hyperparameter combination using grid searches.

Relevant file: hypers_grid_search.py

The hyperparameters that are going to be searched are the (1) maximum hidden neuron, (2) stopping criterion for the training of newly added neuron P", (3) the magnitude of weight decay, and (4) the learning rate L3. The ratio between the learning rate L1, L2, and L3 will be held constant with L2 = 5 * L3 and L1 = 200 * L3. Therefore, they do not need to be adjusted manually. Fig. 6 shows all the candidates for each hyperparameter. Therefore 4 hyperparamers and each of them has 8 candidates. Therefore, there are 8^4 = 2^12 = 4096 different possible hyperparameter combinations, which can be encoded as a 12-bit binary string used by the GA in the later stage.

```
HYPERPARAMETERS_CANDIDATES = {
    'max_hidden_neurons': [1, 2, 3, 5, 7, 10, 13, 15],
    'P': [0.01, 0.05, 0.1, 0.5, 1, 5, 15, 20],
    'D': [0, 0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1],
    'L3': [0.0001, 0.0003, 0.0005, 0.001, 0.003, 0.005, 0.007, 0.01]
}
```

Fig. 6. All hyperparameter candidates.

After running a grid search, a table that map possible hyperparameter combination to an MSE score is stored in "results/fhitness_table.csv" Fig.7 shows the top 10 hyperparameter combinations, where the first column is their corresponding chromosome encoding and the second column is MSE.

```
The top 10 hyperparameter combinations (chromosomes) and their MSE
000111010101    2.1389299273490905      {'max_hidden_neurons': 1, 'P': 20, 'D': 0.0005, 'lrs': [1.0, 0.025, 0.005]}
000011010111    2.1501518368721007      {'max_hidden_neurons': 1, 'P': 0.5, 'D': 0.0005, 'lrs': [2.0, 0.05, 0.01]}
000110101111    2.157095471024513       {'max_hidden_neurons': 1, 'P': 15, 'D': 0.01, 'lrs': [2.0, 0.05, 0.01]}
000100101001    2.1630828052759172      {'max_hidden_neurons': 1, 'P': 1, 'D': 0.01, 'lrs': [0.06, 0.0014999999999999998, 0.0003]}
000001100111    2.171607992053032       {'max_hidden_neurons': 1, 'P': 0.05, 'D': 0.005, 'lrs': [2.0, 0.05, 0.01]}
000010011110    2.1722268730401995      {'max_hidden_neurons': 1, 'P': 0.1, 'D': 0.001, 'lrs': [1.4000000000000001, 0.035, 0.007]}
000011100101    2.1779274702072144      {'max_hidden_neurons': 1, 'P': 0.5, 'D': 0.005, 'lrs': [1.0, 0.025, 0.005]}
000001010111    2.181641271710396       {'max_hidden_neurons': 1, 'P': 0.05, 'D': 0.0005, 'lrs': [2.0, 0.05, 0.01]}
000100100010    2.18173748254776        {'max_hidden_neurons': 1, 'P': 1, 'D': 0.005, 'lrs': [0.1, 0.0025, 0.0005]}
000011000010    2.1826969593763352      {'max_hidden_neurons': 1, 'P': 0.5, 'D': 0, 'lrs': [0.1, 0.0025, 0.0005]}
```

Fig. 7. The top 10 hyperparameter combinations

4.5 Find the number of computations required for GA to yield the top 10 and the best hyperparameter found by grid search. (Goal 1 Completed)

The script “hypers_GA_search_test.ipynb” runs the GA 2000 times and records the number of generations required to get the top 10s and the best chromosome. It was found that the GA need an average of 5.97 generations to output one of the top 10 chromosomes with a standard deviation of 4.43. And an average of 45.68 generations to output the best chromosome among the 2^12 different possible hyperparameter combinations. Since the population size chosen is 100 and in each generation, GA need to call rkf_validator() for every individual in the population to obtain a fitness score. There is required on average 597 computations to find the top 10s and 4568 computations to find the best one.

4.6 Compare GA, grid search, and random search during the process of hyperparameter tuning.

For grid search, the number of computations to get the top 10s and the best one is fixed, which is 2^12 = 4096.

The script “hypers_random_search_test.ipynb” runs the random search 2000 times and records the number of generations required to get the top 10s and the best chromosome. It was found that it requires 4092.27 computations to find the best one and 415.56 computations to find the top 10s.

Key Finding 2

Based on the experiment results, it was found that the GA search does help reduce the number of computations if we aim only to find the top 10 chromosomes. However, if we want to use the best hyperparameter combination using grid search, it will require on average a similar number of computations. Random search and GA search appear to be on par with each other in this scenario.

4.7 Fix the hyperparameters of the model to the best hyperparameter found in the previous hyperparameter tuning steps during the following feature selection processes.

As the scores for the top 10 hyperparameter combinations are very similar, their scores are recomputed based on a larger number of runs of n-repeat k-fold validation. The details of the process can be found in
“hypers_grid_search_top_10s_compare.py”. After the re-computation of MSEs for the top 10 chromosomes. It was found that the hyperparameters {'max_hidden_neurons': 1, 'P': 0.5, 'D': 0, 'lrs': [0.1, 0.0025, 0.0005]} perform the best and thus it will be used in the following feature selection. As suggested in Y.Saeys et al.’s paper [6], there are 3 main types of feature selection methods, filter, wrapper, and embedded. The filter method separates the hyperparameter tuning and feature selection process. The wrapper searches through feature subset space for each feature subset, further searches through hyperparameter combination. The embedded search through the feature subset and hyperparameter at the same time. In GA, it will then have a chromosome containing the information of both enable and disable of feature columns and hyperparameter. The first method is simple and sable and will be used in this paper. Therefore, we fix the hyperparameters to the best hyperparameter found during the previous process.

4.8 Find the best feature subset using grid search.

The script “features_grid_search.py” calculates the score for all 2^15 different subsets of features and records them in the table “results/feature_selection_fitness.csv”, so that the later GA can directly read the fitness score from the table without actually calling “rkf_validator()” during testing, which can save a significant amount of time. The top 10 feature subsets are shown below. It was found that the top feature subsets have a lot of features discarded, which means even after the data pre-processing, the dataset may still contain redundant information or too much information that cannot make the model generalise well to the test set. It was also found that the log version of the feature “time to first click” is highly important when determining satisfaction. This is likely like because the result page, especially the snippet contains too much information, so it takes longer time for the user to read. As suggested in the paper [1], the longer time it takes, the lower the satisfaction. Therefore, it becomes the most important feature that exists in all top 10 feature subsets.

```
2.01 ['Log(TTF)', 'Log(F_title)', 'Minimal value']
2.02 ['Shown_T_num', 'Log(TTF)', 'StartTime', 'Log(F_title)', 'Log(F_Snippet)']
2.03 ['Log(TTF)', 'StartTime', 'Endtime_SERP', 'Log(F_title)', 'Log(F_Snippet)']
2.03 ['Shown_T_num', 'Log(TTF)', 'Task_type', 'Clicked_rank', 'Log(F_title)', 'Original_scanpath_value']
2.04 ['Shown_T_num', 'Log(TTF)', 'Clicked_rank', 'StartTime', 'Endtime_SERP', 'Log(F_title)', 'Log(F_Snippet)']
2.04 ['Shown_T_num', 'Log(TTF)', 'Task_type', 'Endtime_SERP', 'Log(F_title)', 'Log(F_URL)', 'Log(F_Snippet)']
2.05 ['Shown_T_num', 'Log(TTF)', 'scroll', 'StartTime', 'Endtime_SERP','Log(F_title)']
2.05 ['Snippet_length', 'Log(TTF)', 'Task_type', 'Clicked_rank', 'StartTime','Log(F_title)']
2.06 ['Shown_T_num', 'Log(TTF)', 'scroll', 'StartTime', 'Endtime_SERP', 'Log(F_title)', 'Original_scanpath_value']
2.06 ['Snippet_length', 'Shown_T_num', 'Log(TTF)', 'StartTime', 'Endtime_SERP', 'Log(F_title)', 'Log(F_URL)', 'Log(F_Snippet)']
```

Fig. 8. The top 10 feature subset

4.9 Find the number of computations required for GA to yield the top 10 and the best hyperparameters found by grid search. (Goal 2 Complete)

The script “features_GA_search_test.ipynb” the GA 2000 times and calculates the average generation required to find the best and the top 10 feature subsets. The population size of the GA is also 100. It shows that the average number of generations to get the best chromosome is 211.79 with a standard deviation of 205.8, which means 21179 computations are required. To get the top 10 feature subsets, there are on average 2317.6 computations.

4.10 Compare GA, grid search, and random search in terms of the number of computations required to find the best one and the top 10s.

For grid search, the number of computations to get the top 10s and the best one is fixed, which is 2^15 = 32768

The script “features_random_search_test.ipynb” runs the random search 2000 times showing that it requires 32065.24 computations to find the best one and 3233.26 computations to find the top 10s.

Key Finding 3

The finding of 4.10 is similar to the finding of 4.6. The GA search and random search do help reduce the number of computations if we aim only to find the top 10 chromosomes and their performance is likely similar. However, if we want to use the best hyperparameter combination using grid search, it may still require grid search to provide a more stable result.

4.11 Choose a final model and perform a final test on MSE and MAE

Relevant script: reflect_to_dataset_paper.ipynb

Finally, the final Casper model is chosen from one of the top 10 models during the feature selection processes (4.9 - 4.10). The hyperparameters of these models are also the best hyperparameters chosen from the hyperparameter tuning processes (4.4 - 4.6). Below are the hyperparameters and feature subsets used by the model. The reason for choosing this feature subset instead of the top one is that their score is quite similar and does not have a statistical difference. Also, the chosen feature subset maintains the “Snippet_length” column, which can better reflect the model behaviour to on the findings of the dataset paper.

Hyperparameters = {'max_hidden_neurons': 1, 'P': 0.5, 'D': 0, 'lrs': [0.1, 0.0025, 0.0005]}
Feature Subset = ['Snippet_length', 'Shown_T_num', 'Log(TTF)', 'StartTime', 'Endtime_SERP', 'Log(F_title)', 'Log(F_URL)', 'Log(F_Snippet)']

Fig. 9 shows the distribution of squared errors and absolute errors recorded after training the testing the model 2000 times. Based on the mean absolute error and the distribution graph, the model predicts user satisfaction scores with an average absolute error of 1.127. This implies that, on average, the predicted satisfaction score deviates by 1.127 units from the actual score. This level of accuracy is commendable, especially considering that even humans might lack full confidence when distinguishing between two closely ranked scores.

4.12 Overall judgement on models’ effectiveness (Goal 3 Completed)

Based on the L1 matrices and the distribution graph, it was shown that the model indeed has learned some features from the dataset and is able to use them to accurately predict user satisfaction given some extent of tolerance. The L1 score of the model are both around 1.1, which tells us that on average the model prediction has a 1.1 satisfaction level different from the ground truth. Even though a satisfaction level of 7 is predicted as 6, it may not make a significant difference.

This is because sometimes it is hard for participants to choose between 6 and 7 when they are asked for the confiscation. The maximum of L1 loss across all samples provides some guarantee on the model performance and they are not too far from the mean loss. The standard deviation of L1 loss is very small, which means the loss is very consistent and the performance of the model is stable.

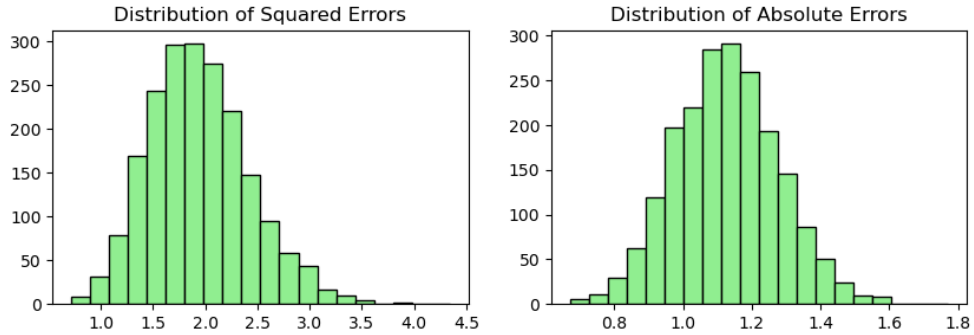


Fig. 9. Distribution: square errors and absolute errors

	Mean	Median	Std
Squared Errors	1.946	1,901	0.485
Absolute Errors	1.131	1.124	0.153

Table 1

4.13 Verify that the model is able to give statistically lower satisfactory scores for inputs with long snippet sizes than inputs with short and median snippet sizes. (Goal 4 Complete)

The script “reflect_to_dataset_paper.ipynb” runs the model 2000 times with different testing and training splits. It gathers all predicted satisfactory scores of the model and groups them into three groups, based on the snippet size of the sample. The purpose of doing this is that the original dataset paper found that a longer snippet size can result in lower user satisfaction. The satisfactory scores in the long snippet groups should be statistically lower than the scores in the other two groups. By performing p-tests between them it was found that the p-value between the median snippet group and long snippet group is 4.03e-6 and the p-value between the short snippet group and long snippet group is 0.0050. Both p-values are smaller than 0.05. Therefore, we can conclude that the model is able to give a lower satisfaction score to a sample with a long snippet size, which is aligned with the finding of the dataset paper.

5 Conclusion

This study has explored the efficiency of the GA in hyperparameter tuning and feature selection of the Capser model when it is applied to the prediction of user satisfaction with mobile web search. It was found that the GA can indeed save a lot of computation time if we only aim to find one of the top 10 hyperparameter combinations or feature subsets within the search space. Since the scores for the top 10 results are quite similar, by considering their standard deviation, it cannot statistically say that one is better than the others. Therefore, choosing one of the top 10s should be enough to make the model perform reasonably well compared with the ground truth. During the hyperparameter tuning and feature selection, it was also found that random search has a similar performance to the GA in terms of the average computation needed to find out one of the top 10s and the best chromosome.

In addition, this study also demonstrates the effectiveness of Casper model in predicting user satisfaction. The model can predict a satisfactory score that is on average 1.131 close to the true satisfactory, which is good enough considering the size of the dataset. It also shows that the mode-predicted results are aligned with the findings of the dataset paper, which can give lower satisfaction scores to those samples with long snippet sizes.

6 Future Work

The conclusions drawn from this research are based on a limited dataset of 261 samples. Future studies could try using a larger data set to develop a more generalised model with higher accuracy and less overfitting.

Additionally, the GA for finding the best hyperparameter and the GA for feature selection can be combined, i.e., encode the feature selection data and hyperparameter data in the same chromosome and find them together. In this study, although the best hyperparameter used during the feature selection is the best hyperparameter found during the hyperparameter turning. However, since the number of features used can change the model behaviour can and the hyperparameter may need to be adjusted again, it may no longer be the best hyperparameter.

References

1. Kim, J Paul, T Sankaranarayna, R Gedeon, T Yoon, HJ 2017, ‘What Snippet Size is Needed in Mobile Web Search?’, *CHIIR '17: Proceedings of the 2017 Conference on Conference Human Information Interaction and Retrieval, March 2017*, <https://dl.acm.org/doi/10.1145/3020165.3020173>.
2. Treadgold, NK, Gedeon, TD 1997, ‘A Cascade Network Algorithm Employing Progressive RPROP’, *Lecture Notes in Computer Science*, vol. 1240, <https://link.springer.com/chapter/10.1007/BFb0032532>
3. Braun, H Riedmiller, M 1993, ‘A direct adaptive method for faster backpropagation learning: the RPROP algorithm’, *IEEE International Conference on Neural Networks*, <https://ieeexplore.ieee.org/document/298623/authors#authors>
4. Kwok, T Yeung, D 2005, Bayesian regularization in constructive neural networks, *Lecture Notes in Computer Science*, vol.1112, <https://link.springer.com/chapter/10.1007/3-540-61510-5_95>

5. Liashchynskyi, P & Liashchynskyi P 2019, ‘Grid Search, Random Search, Genetic Algorithm: A Big Comparison for NAS’, *arXiv:1912.06059*, <https://arxiv.org/abs/1912.06059>
6. Saeys, Y & Inza I & Larranaga P 2007, ‘A review of feature selection techniques in bioinformatics’, Bioinformatics, Volume 23, Issue 19, October 2007, Pages 2507–2517, <https://academic.oup.com/bioinformatics/article/23/19/2507/185254>

Acknowledgement

ChatGPT has been used to polish the writing and perform grammar checks.