Performance of cache placement using supervised learning techniques in mobile edge networks

Lubna Mohammed | Alagan Anpalagan | Ahmed S. Khwaja | Muhammad Jaseemuddin

Department of Electrical, Computer and Biomedical, Engineering, Ryerson University, Toronto, Canada

Correspondence
Lubna Mohammed, Department of Electrical, Computer and Biomedical, Engineering, Ryerson University, Toronto M5B 2K3, Canada.
Email: lubnabadri@ryerson.ca

Funding Information
Natural Sciences and Engineering Research Council of Canada

Abstract
With the growth of mobile data traffic in wireless networks, caches are used to bring data closer to mobile users and to minimise the traffic load on macro base station (MBS). Storing data in caches on user terminals (UTs) and small base stations (SBSs) faces challenges with respect to the decision of cache contents. Here, a multi-objective cache content strategy that aims to maximise the cache hit rate of SBSs in mobile edge networks (MENs) is proposed. The multi-objective cache placement optimisation is formulated as a classification problem. Unlike previous work, mobility input attributes such as user locations, contact duration, communication ranges, contact probability between UTs and SBSs, etc. as well as content popularity and the correlation between these input attributes separating the decision space into two regions of cache and not cache are used. Stochastic gradient descent algorithm is used for the training of three supervised machine learning techniques: artificial neural network ANN, support vector machine (SVM), and logistic regression LR to define the hyperplane that separates the cache content decision space. Simulation results show that compared with the weighted-sum approach, the SBSs cache hit rates increase on the average by 18.58%, 18.52%, and 18.2%, and the total energy consumption values decrease on the average by 33.49%, 53.19%, and 49.9% for ANN, SVM, and LR, respectively.

1 | INTRODUCTION

Worldwide, smart mobile devices have been experiencing an exponential growth recently and their number is expected to reach billions in the coming years. With the increased applications and services running on these devices, data and mobile traffic continue to increase, which challenges the design of future wireless networks [1]. The increase in mobile data traffic mostly comes from repetitive downloads of popular contents [2]. By knowing the content popularity distribution, most popular contents can be placed proactively closer to the users by deploying caches at wireless network edges such as in user terminals (UTs) and small base stations (SBSs), termed as edge caches. This placement will significantly minimise the transmission load at both backhaul and core networks, and also improve users quality of service by minimising the latency for downloading the contents from near by caches [3]. The mobile edge network (MEN) architecture provides storage and computation resources at the network edge, which reduces the delay for delivering the contents from macro base station (MBS) to UTs and support computation-intensive mobile applications. The most popular contents can be stored at MBS, SBSs, and UTs caches during off-peak hours.

Cache placement techniques are used to improve network performance by maximising the cache hit rate, which means maximising the amount of content downloaded from caches at the edge of the network. A continuous change in content popularity is not the only challenge that is faced in designing cache placement techniques. There are several other challenges due to continuous changes in user mobility, number of users within each network, different paths the users take while moving, and the changes in the contact probability between a UT and other UTs and between a UT and SBSs. In addition to these challenges, many applications require transfer data within power and delivery time constraints that need to be considered when designing caches at MENs.

This is an open access article under the terms of the Creative Commons Attribution License, which permits use, distribution and reproduction in any medium, provided the original work is properly cited.

© 2021 The Authors. IET Networks published by John Wiley & Sons Ltd on behalf of The Institution of Engineering and Technology.
Various cache placement schemes have been designed for edge caching in wireless networks. Most existing literature on cache content placement algorithms aim to maximise the cache hit rate by placing contents based on the change in content popularity. This is because a large amount of content requests are caused by a few popular files. Using popular files, cache placement may result in replication of the same file in many caches in one cell. In [4], the authors presented cache placement algorithm where the decision of cache or not cache (1 or 0) is converted into a convex optimisation problem where the decision is relaxed to any value in [0,1]. Using an optimisation technique, a non-integer solution is found and converted into an integer solution. In this algorithm, authors took content popularity distribution as an input to the optimisation problem and specified the constraints of the problem based on the network topology.

A deterministic caching approach was used for cache placement based on computational greedy algorithms. In [5], an optimal greedy algorithm was proposed for cache content placement. The authors exploited the search for optimal contents by balancing between the content diversity in caches within the wireless network cell and the channel diversity gain. The authors in [6] designed the cache placement with the objective to minimise the delay of delivering the file to the end users. They proved that the objective function was a monotone non-increasing super-modular function that could be solved by a greedy algorithm to obtain a locally optimal solution. Deterministic caching approaches can improve network performance only with the assumption that UTs stay at the same locations for several hours [6].

To enlarge the set of cached files and exploit caching diversity, cooperative caching and device-to-device (D2D) caching approach was presented in [7–9]. The D2D approach utilises UT caches to improve network performance. The D2D cache placement algorithms considered content popularity, distances between users in one cell, and social association between users [10]. An asymptotically optimised online learning content placement and content delivery algorithm was proposed in [10]. The algorithm used a distributed on-line learning approach at individual edge servers based on stochastic gradient descent (SGD). Studies that use multi-winner auction theory for cache placement-based D2D communication showed that the network throughput increased linearly with the number of UTs in one cell [11].

A learning-based proactive caching was investigated to predict the content demand of users. In [12], the authors used singular value decomposition (SVD) to predict content preferences. In [13], users-to-content association was used to predict content popularity using non-negative matrix factorisation (NMF) machine learning. This is a linear model that learns through one layer relationship between the user and the contents. Multi-layer learning was proposed in [14] that used a deep neural network, which provides a non-linear model for the association between users and the content demands. Distributed deep learning (DDL) was proposed in [15] in which deep learning was used to predict users demand at the edge of the network. Then, the trained models were collected by the central server from the MENs and the global model was updated accordingly.

The cache placement problem was studied from the perspective of user mobility. In [16], the authors presented the effects of exploiting the statistical traffic pattern and users’ context information, such as file popularity, user location, and mobility pattern on the amount of resources needed and at which network location the content should be pre-cached. Context-aware proactive caching was also considered in the work proposed in [17]. The authors formulated a multi-armed bandit optimisation problem. They considered user mobility by computing connected users’ preferences and users’ service priorities in a given time slot. The content popularity was considered to change over time by taking into account users’ movements and connections between new sets of users. In [18], the authors addressed realistic features of cache placement by describing user mobility via the set of highly visited locations within the cell where each location was covered by one or more SBSs. The impact of mobility-awareness in cache placement algorithm was discussed in [19]. The authors formulated the problem of caching of coded segments at base stations (BSs) and UTs taking into account user mobility and the content amount per transmission. User mobility was presented as a peer-to-peer connectivity model with contact duration and contact frequency perspectives. The problem was formulated as an integer programming problem and solved by sub-modular optimisation.

In most previous works, such as those in [16–19], the impact of user mobility on caching strategy was considered from one of the following perspectives: either from users’ content preferences, highly visited places by the users, user social context, content segmentation and multi-casting of segments to multiple cache locations, or users’ contact probability. However, improving the quality of service by increasing the cache hit rate that leads to minimising the latency of downloading the contents and minimising the total energy consumption require the utilisation of learning techniques in the modelling of cache placement. Learning technique can be used to exploit different user mobility attributes and the relationship between these attributes that can affect cache placement decisions. This is a challenging problem and motivates for further investigations. Future wireless network should be more adaptive to the changes in user mobility, data rate, delay, number of SBSs and user terminals, and many other network information and parameters. To improve the adaptiveness to the dynamic changes in wireless networks, the network intelligence should be used to model network status and update future predictions. This is an open research challenge requiring the design of edge caching strategies which can automatically adapt the wireless network changes [20].

In this paper, a new formulation of cache placement problem is investigated. This is based on our formulation in [21] of a multi-objective function that aims to maximise the cache hit rate by maximising the file content popularity, maximising the contact duration between the mobile user and the cache that stores the required contents, minimising the communication range between mobile user and the cache, and
maximising the UT contact probability. In the above mentioned formulation and solution, we assumed equal weights for all input parameters. In this paper, we focus on the values of weighted-sum (WS) cache placement input attributes to model the relationship between these values and the hit rate when the users are moving. Weights are assigned to each input to assess the contribution of input attributes on cache placement decision. The multi-objective cache placement optimisation is formulated as a classification problem. The formulation of cache placement as a binary classification was investigated in [22]. The authors considered the input propriety of video contents and user context requesting the video contents are the input attributes that defined the space of cache decision. The input attributes included information such as the number of views, number of likes, shares, comments, language, user subscription, and user past views. Unlike previous work, we explore mobility input attributes such as user locations, contact duration, communication ranges, contact probability between UTs and SBSs, as well as content popularity and the correlation between these input attributes that can separate the decision space into two regions for cache and not to cache. The main contributions of this work are summarised as follows:

- Unlike existing techniques, we propose a new cache placement algorithm formulated as a binary classification problem (to cache and not to cache) based on user locations, contact probability, communication range, contact duration, and content popularity. Artificial neural network (ANN), support vector machine (SVM), and logistic regression (LR) are used to model cache placement decisions.
- We investigate the characteristics of the input features (attributes) and the properties of these characteristics that affect supervised machine learning approaches.
- We investigate the performance of new cache placement models using supervised learning techniques by placing the proposed models in work with different system parameters. These parameters are varied to study the sensitivity of classification decisions with the change of system parameters.

The remainder of the paper is organised as follows: Section 2 describes the system model and assumptions. In Section 3, we present cache placement problem formulation using binary classification approach. Section 4 discusses hyperplane parameters estimation based on the learning techniques. In Section 5, we provide the simulation results and discussion followed by conclusion in Section 6.

2 | SYSTEM MODEL

As illustrated in Figure 1, we consider a MEN with multiple SBSs and UTs. Each macrocell consists of one MBS connected to a gateway of the core network via a high-speed interface, N SBSs that are connected to the MBS through backhaul links, and M UTs connected to neighbouring devices through D2D communication and to SBSs. The sets of SBSs and UTs are denoted by \( s = \{s_1, s_2, \ldots, s_N\} \) and \( u = \{u_1, u_2, \ldots, u_M\} \), respectively. There are cache storage units in each MBS, SBS, and UT with different storage capacities. Within one macrocell, cache storage capacities can be defined by two sets \( c_s = \{c_{s_1}, c_{s_2}, \ldots, c_{s_N}\} \) and \( c_u = \{c_{u_1}, c_{u_2}, \ldots, c_{u_M}\} \) for SBSs and UTs caches, respectively. It is assumed that the MBS has enough cache capacity to store \( F \) files defined by the set \( f = \{f_1, f_2, \ldots, f_F\} \).

There can be four types of cache content delivery approaches, as shown in Figure 1. Out of these approaches, we consider the following approaches: local cache, SBS cache, and MBS cache.

The content library consists of \( F \) files stored in the MBS cache. Each file is denoted by \( f_z \) for \( z = 1, \ldots, F \). The files are requested from the main library based on their popularity distribution. The popularity of the \( F \) files are denoted by the set \( \Lambda \), where \( \Lambda = \{\Lambda_1, \Lambda_2, \ldots, \Lambda_F\} \). The set \( \Lambda \) can be characterised by the Zipf popularity distribution [23].

It is assumed that the connectivity of users in the D2D communication and in the small cell network (communication between mobile UT and SBS) is a peer-to-peer connectivity model [19]. The user mobility is modelled based on spatial and temporal properties. The spatial properties provide physical location information of user mobility patterns, while the temporal properties provide time-related information [24]. The user mobility can be modelled by assuming a pairwise contact process that follows an independent Poisson process.

3 | MULTI-OBJECTIVE CACHE PLACEMENT

A cache placement algorithm provides decisions on which contents to cache and where to cache them. After completing the cache placement phase, users start sending their requests for contents during cache access and delivery phase. If the content is downloaded from another UT and/or from SBS caches, it is considered a hit. If the content is downloaded from the MBS, it is considered a miss. The cache placement policy aims to maximise the cache hit rate that minimises the content delivery latency. The weighted-sum cache placement approach provides an insight into the input data attributes. A cache placement approach that is able to provide an insight into which input attributes can maximise the hit rate is useful for implementing efficient content delivery to UTs.

For \( M \) UTs, \( N \) SBSs, \( i, k = 1, 2, \ldots, M \), and \( j = 1, 2, \ldots, N \), the contact durations between all UTs and other UTs and SBSs within one cell are represented by \((T^{UT.SBS})_{M \times (M+N)}\). Similarly, a matrix \((D^{UT.SBS})_{M \times (M+N)}\) represents the communication ranges at time \( t \), when both UT \( u_i \) and UT \( u_k \) are within the D2D communication range \( d^{D2D} \), and UT \( u_i \) and SBS \( s_j \) are within the communication range \( d^{SBS} \), respectively. The probability of a UT to contact other UTs or SBSs \( p^{UT.SBS}_{ut} \) is a matrix of size \((M \times (M+N))\), which represents the probability of UT \( u_i \) to contact with UT \( u_k \) and for the probability of UT \( u_i \) to contact with SBS \( s_j \).
Assume a matrix $A = [A^{UT} : A^{SBS}]_{(M+N) \times F}$ as the cache strategy matrix that needs to be determined by the cache placement algorithm. The contents of matrix $A$ are either one or zero (cached or not cached). Let the matrix $(A^{UT})_{M \times F}$ represent the cache strategy of file $f_x$ that is cached in UTs, where $a_{k,x} \in A^{UT}$ represents file $f_x$ that can be served by UT $u_k$, $(k = 1, \ldots, M)$. Then, the matrix $(A^{SBS})_{N \times F}$ represents the cache strategy of the files cached in SBSs, where $d_{j,x} \in A^{SBS}$ represents file $f_x$ that can be served by SBS $S_j$ $\forall j \in \{1, \ldots, N\}$. The objective function for latency-efficient cache placement algorithm maximises the probability of UT $u_i$ to download the requested files from nearby UTs and SBSs within the contact durations [21], and is written as follows:

$$\begin{align*}
\text{maximize} & \quad \sum_{x=1}^{F} g^*(A, w_x) \\
\text{subject to} & \quad \sum_{x=1}^{F} a_{i,x} f_x \leq C_i, \forall i \in \{1, \ldots, M\}, \quad (1b) \\
& \quad \sum_{x=1}^{F} d_{j,x} f_x \leq C_j, \forall j \in \{1, \ldots, N\}, \quad (1c)
\end{align*}$$

and the weighted-sum of the normalised objective values is

$$g^*(A, w_x) = w_1 \sum_{i=1}^{M+N} A(i,:) \times A^T \forall x \in \{1, \ldots, F\},$$

$$+ w_2 \sum_{i=1}^{M} \sum_{x=1}^{F} T^{UT,SBS}(i,:) \times A(:,z)$$

$$- w_3 \sum_{i=1}^{M} \sum_{x=1}^{F} D^{UT,SBS}(i,:) \times A(:,z)$$

$$+ w_4 \sum_{i=1}^{M} \sum_{x=1}^{F} p^{UT,SBS}(i,:) \times A(:,z),$$

where $a_{k,x} \in [0,1], \forall k \in \{1, \ldots, M\},$ $d_{j,x} \in [0,1], \forall j \in \{1, \ldots, N\},$ $w_1, w_2, w_3,$ and $w_4 > 0$. 

\[\text{FIGURE 1}\] An illustration of the network model showing the four cache content delivery approaches: (a) local cache, (b) D2D cache, (c) small base stations cache, and (d) macro base station cache.
where $w_1$, $w_2$, $w_3$, and $w_4$ are the weights of each objective function. The file popularity probability, normalised contact duration, normalised communication range, and contact probability are represented by $\lambda^T_{m}, T^{UT.SBS}, D^{UT.SBS},$ and $P^m_{UT.SBS}$, respectively. In the multi-objective optimisation problem, weights are considered as general gauges of relative importance for each objective function. The importance of one objective function over the other objective functions to maximise the cache hit rate requires selecting the set of weights that reflects these preferences. The final solution may not accurately reflect initial weight preferences of each objective function [25]. In the cache placement problem, objectives may conflict with one another and optimising one solution with respect to a single objective function may result in an unacceptable solution with respect to other objective functions. For example, placing files in caches depending on maximising the content popularity and neglecting other objective functions like minimising the distances between UT and other UTs and SBSs may result in an unreasonable solution. A set of solutions need to be investigated such that each solution satisfies all objectives at an acceptable level.

### 3.1 Input attributes influence weights

The cache placement decision is a binary decision (place or not to place the contents) that results in one of two cases: either the content is found (Hit = 1) or the content is not found (Hit = 0) in UT and SBS caches when the user requests the content. We will use the binary classification conditional probabilities presented in [26]. As mentioned before, in cache placement, we have four input attributes: file popularity probability ($\lambda_2$), normalised contact duration ($T^{UT.SBS}$), normalised communication range ($D^{UT.SBS}$), and contact probability ($P^m_{UT.SBS}$). Each input attribute has its influence on the cache placement decision, which is represented by the weights. The combination of these influence weights gives an influence score for cache placement decision.

To formulate the influence of weights, we assume the influence weight for the cache placement decision for each input attribute to be between $-1$ and $+1$. That means, the certainty of the decision (place) produces a weight of $+1$ and the certainty of the decision (not place) produces a weight of $-1$. Thus, the influence weight of the input attribute value represents the influence weight for an input attribute on cache placement decision relative to its opposite. For any input attribute, the sum of the conditional probabilities for each possible result is 1 as the events are mutually exclusive, shown as follows:

$$Pr(\text{Hit} \mid I_i) + Pr(\text{Miss} \mid I_i) = 1,$$

where $I_i$ is the value of the input attribute $\forall i \in \{1, \ldots, 4\}$. Consider that the cache hit is positive and the probabilities can be mapped between the range $-1$ to 0. The range of mapped probabilities for any input attribute is between $-1$ and $+1$. Since each weight represents its influence on both probabilities, we can compute the influence weight for an input attribute for both cache states as follows:

$$w_i = Pr(\text{Hit} \mid I_i) + Pr(\text{Miss} \mid I_i).$$

Assume for a given period of time, there are $V$ number of user requests. Each request results in a hit or a miss, then the hit rate ($\hat{b}$) can be computed as follows:

$$\hat{b} = \frac{\text{Total Number of Hits}}{\text{Total Number of Requests}(V)}.$$

To demonstrate the calculations for influence weights, assume $Pr(\text{Hit} \mid \lambda_2) = 0.6$, $Pr(\text{Hit} \mid T^{UT.SBS}) = 0.8$, $Pr(\text{Hit} \mid D^{UT.SBS}) = 0.9$, and $Pr(\text{Hit} \mid P^m_{UT.SBS}) = 0.4$. Then, $Pr(\text{Miss} \mid \lambda_2) = -0.4$, $Pr(\text{Miss} \mid T^{UT.SBS}) = -0.2$, $Pr(\text{Miss} \mid D^{UT.SBS}) = -0.1$, and $Pr(\text{Miss} \mid P^m_{UT.SBS}) = -0.6$ which are mapped between 0 and $-1$. According to (4), the file popularity probability influence weight ($w_1$), normalised contact duration influence weight ($w_2$), normalised communication range influence weight ($w_3$), and contact probability influence weight ($w_4$) are equal to 0.2, 0.6, 0.8, and $-0.2$, respectively. Figure 2 shows an example of input attribute influence weights on the two classes of the cache placement problem.

### 3.2 Classification-based influence scores

For each user request, we can compute a single score for each influence weight of the input attributes. If we calculate the summation and averaging of the input attribute influence weight $w_i$ for all user requests for input attribute $I_a$, then we can compute a score that can be used as an indicator of the influence weight of the input attribute. For our cache placement input attributes, the influence weight input attribute scores is

$$S_a = \frac{1}{V} \sum_{j=1}^{V} w_{a,j},$$

where $a = \{1, 2, 3, 4\}$ and $S_1$, $S_2$, $S_3$, and $S_4$ represent the influence weight input attribute scores for ($\lambda_2$), ($T^{UT.SBS}$), ($D^{UT.SBS}$), and ($P^m_{UT.SBS}$), respectively $w_{a,j}$ is the influence weight for the $a$th input attributes and $j$th user request.

According to (6), the influence score can be used as an evidence that the corresponding example results in one class and not the other class. The samples that represent previous cache placement decisions for $V$ user requests are considered as a training data set. From the training data sets, we can search for cases that result in correct classification and misclassification.
Classification is the task of predicting the class for which the set of input attributes belongs. The classifier \( k(I, w) \) aims to define a decision surface \( k(I, w) = 0 \) such that for a given set of input attributes \( I \), the classifier can predict to which class the input attributes belong. Then, given the value of \( I \), its class label \( y \) is predicted according to the following:

\[
y' = k(I, w),
\]

where \( k(I, w) \) is used to decide on which side of the decision surface \( k(I, w) = 0 \) lies the input attributes \( I \) [27]. For a given training set with \( V \) samples that result in either \( y = -1 \) in a miss or \( y = 1 \) in a hit, the goal is to design a hyperplane that separates these two classes.

\[
k(I, w) = f(I, w) = f(I, w_0, w_1, w_2, w_3, w_4) = 0,
\]

where \( w_0 \) is the bias and \( f \) is the classifier function, which can be either linear or non-linear. We need to build a classifier that is able to place points with negative score values on one side of the hyperplane and place points with positive score values on the other side of the hyperplane. Figure 3 shows an example scenario, where the hyperplane is separating the 4-dimensional space into two regions. For each row in the training set, we have the values of the input attributes and the value of the label \( y \). Label \( y \) is the class that this row belongs to, so it is either \( y = -1 \) or \( y = +1 \). The influence weights \( w_i \) for \( i = 1, \ldots, 4 \) of input attributes and the value of the bias \( w_0 \) can be computed by minimising the mean square error (MSE) cost function \( J(w) \) as follows [28].

\[
\text{minimize } J(w) = \frac{1}{V} \sum_{j=1}^{V} (y_j - y'_j)^2,
\]

where \( y'_j \) is the predicted class for the given values of influence weights and input attributes, and \( y \) is the actual class value for the 4th sample. The MSE should be minimised such that the predicted responses would be as close as possible to the actual values of the class. The weights and bias are estimated to formulate the optimal model for the classifier. A learning technique is presented in the following section that can be used to estimate the values of influence weights and build the cache placement model.

4 | HYPERPLANE PARAMETERS ESTIMATION BASED ON LEARNING TECHNIQUES

The hyperplane parameter estimation can be implemented using supervised machine learning classifiers. Supervised classifier requires a dataset consisting of input/output pairs. Each row in the data set represents the observations (input), or what is referred to as the vector of features and the corresponding class \( y \). The classification function aims to build a model by training the given dataset, and then the model will be able to estimate the output \( y' \) for new input features. The classifier learns from the previous datasets to predict the output when given new input features. The output of the classifier is the decision of either placing the content (cache) or not placing the contents (not cache). The objective function of the learning algorithm is to minimise the cost function given in Equation (9). The algorithm used for optimising the objective function is the stochastic gradient descent (SGD) algorithm, which learns the statistics iteratively from the training dataset.

Three supervised machine learning (ML) techniques are used as binary classifiers for the cache content placement algorithm, as presented in the following subsections. We have adopted SGD algorithm in the three learning techniques, because in the literature it has been shown that SGD is able to minimise the loss function evaluated over a given training set and to find the right set for the estimated hyperparameters. It has the ability to avoid the overfitting problem by stopping the iterations of the optimisation routine early before it converges [29].

Consider the training data set, also called the observations, that consists of \( n \) input features \( (I_1, I_2, \ldots, I_n) \) and the corresponding class \( y \) for \( V \) the number of observations. The classifier output \( y \) can be 1 that means the observation belongs to the class ‘cache’ or 0 which means the observation belongs to the class ‘not cache’.

4.1 | Artificial neural network (ANN) for cache placement

Artificial Neural Network (ANN) is first adapted for binary classification of cache placement problem. The adaptive momentum (Adam) gradient descent-based optimiser is used for estimating the cache content. The Adam algorithm is a combination of RMSprop and SGD with momentum. It uses the square gradient to scale the learning and uses moving average of
the gradient instead of gradient with momentum [30]. The Adam algorithm is utilised in machine learning problems with high-
dimensional input parameter space and huge datasets. The algo-
algorithm calculates the adaptive learning rate individually for
various parameters involved in the training of gradients [30].

We consider an ANN that has three layers: one input layer,
one hidden layer, and one output layer. The input layer is the first
layer of the ANN, and it does not have any weights associated
with it. The input layer consists of a set of \( n \) neurons representing
the number of input features. Each neuron in the hidden layer
transforms the values from the previous layer with a weighted
linear summation followed by a sigmoid non-linear activation
function. The output layer receives the output from the hidden
layer and computes the output value \( y \). The ANN estimates a
non-linear classifier that separates the two classes. This classifier
can be written as \( k(I, w) = \hat{f}_\Omega(I, w) \), where \( \hat{f}_\Omega(I, w) \) is a non-linear
function consisting of the weights of the neurons in the hidden
and output layer, as well as the activation function.

\[
\Delta \Omega_t = -\alpha \frac{m'_t}{\sqrt{v'_t} + \epsilon} \\
\Omega_{t+1} = \Omega_t + \Delta \Omega_t
\] (10) (11)

Algorithm 1 Artificial Neural Network Classifier for
Cache Placement

1: **Input**: Dataset with input features \( I_j \) and
one output label \( y \).
2: **Output**: Classifier model \( k(I, w) \) given
in (7).
3: **Initialise**: bias and weights \( \Omega = \text{random} \)
value \( \in [0, 1], \beta_1 = 0.9, \beta_2 = 0.999, \) moving
averages \( m_{t-1} = v_{t-1} = 0, \varepsilon = 10^{-8} \) and
learning rate \( \alpha = 0.001 \).
4: Split dataset into 70% for training and
30% for testing.
5: **Construct** artificial neural network.
   **Training the model:**
   6: **for** \( i = 1 \): Number of epochs **do**
      7: \( \delta_z = \nabla_\Omega J(\Omega_{t-1}) \)
      8: \( m_t = \beta_1 m_{t-1} + 1 + (1 - \beta_1) \times \delta_z \)
      9: \( v_t = \beta_2 v_{t-1} + 1 + (1 - \beta_2) \times \delta_z^2 \)
     10: \( m'_t = \frac{m_t}{1 - \beta_1^t} \)
     11: \( v'_t = \frac{v_t}{1 - \beta_2^t} \)
     12: \( \Omega_{t+1} = \Omega_t - \alpha \frac{m'_t}{\sqrt{v'_t + \epsilon}} \)
   **end for**
   **Return** model structure with updated bias
and weights \( \Omega_j \).
   **Testing the model:**
   15: **for** each instance in the testing
dataset **do**
   16: **Read** input attributes \( I_j \)
   17: **Compute** \( y' \) using the resulting model
from step 14.
18: **end for**

- \( m \) the exponential average of gradient along \( \Omega \).
- \( v \) the exponential average of squares of gradient along \( w \).
- \( m'_t \) the bias-corrected first moment estimate.
- \( v'_t \) the bias-corrected second raw moment estimate.
- \( \Omega_t \) the model weights at time \( t \).
features) into two classes. Given a training dataset \(\{\{I_i, y_i\}\}_{i=1}^V\) with vectors \(I_i \in \mathbb{R}\) and labels \(y_i \in \{+1, -1\}\), the aim is to design a linear classifier \(k(I, w)\) that satisfies the following [31]:

\[
\begin{align*}
w^T I_i + w_0 & \geq +1, \text{ if } y_i = +1, \\
w^T I_i + w_0 & \leq -1, \text{ if } y_i = -1.
\end{align*}
\]

(12)

This is equivalent to

\[
y_i[w^T \phi(I_i) + w_0] \geq 1, \text{ for } i = 1, \ldots, V.
\]

(13)

The non-linear function \(\phi(.)\) maps the input space to a high dimensional feature space. This means the SVM classifier objective function can be defined as [31].

\[
\min_w J(w) = \frac{1}{2} w^T w + C \sum_{i=1}^V \max(0, 1 - y_i w^T I_i),
\]

(14)

where \(\min \frac{1}{2} w^T w\) is the regularisation term that maximises the margin and imposes a preference over the hypothesis space to achieve better generalisation. The term \(\sum_{i=1}^V \max(0, 1 - y_i w^T I_i)\) is called the hinge loss (empirical risk loss) and is used to penalise weight vectors that make mistakes. The penalty parameter \(C (C > 0)\) is used to control the trade-off between a large margin and a small hinge-loss.

As shown in Equation (14), if the sign of the product between the true label \(y\) and the predicted value \(y' = w^T I\) is positive and larger than 1, the loss is zero. If not, the loss will increase linearly. The stochastic gradient descent algorithm for SVM performs gradient descent with respect to the objective function is given in Equation (14). The updating rule for the weights is then [32].

\[
w_{t+1} = w_t - \alpha \nabla_w \left[ \frac{1}{2} w^T w + C \sum_{i=1}^V \max(0, 1 - y_i w^T I_i) \right],
\]

(15)

where \(\alpha\) is the learning rate and \(\nabla_w\) stands for a subgradient with respect to \(w\). The algorithm of SVM for cache placement is shown in Algorithm 2.

### Algorithm 2 Support Vector Machine Classifier for Cache Placement

1: **Input:** Dataset with input features \(I_j\) and one output label \(y\).
2: **Output:** Classifier model \(k(I, w)\) given in (7).
3: **Initialise** bias \(w_0 = 0\), weights \(w_j = 0\), and learning rate \(\alpha = 0.001\).
4: **Split** dataset into 70% for training and 30% for testing.
5: **Training the model:**
6: \(\text{for } i = 1: \text{Number of epochs do}\)
7: \(\text{Pick a random sample } (I_i, y_i) \text{ from the training dataset.}\)
8: \(\text{Compute } \nabla_w \text{ of } J(w) \text{ (14)}\)
9: \(w_{t+1} = w_t - \alpha \nabla_w J(w_t)\)
10: **end for**
11: **Return** model structure with updated bias \(w_0\) and weights \(w_j\).
12: **Testing the model:**
13: \(\text{for each instance in the testing dataset do}\)
14: \(\text{Read input attributes } I_j\)
15: \(\text{Compute } y' \text{ using the resulting model from step 10.}\)
16: **end for**

### 4.3 Logistic regression (LR) for cache placement

The aim of binary logistic regression (LR) algorithm is to train a classifier to make a binary decision about a class of a new input observations. The LR model estimates the probability \(P(y = 1|I)\) that this observations belongs to the class ‘cache’ or the probability \(P(y = 0|I)\) that it belongs to the class ‘not cache’. The LR learns from the training set by updating the weights \(w_1, w_2, \ldots, w_k\) and the bias \(w_0\). As discussed in section 3.1, the weights indicate the importance of each input feature to the classification problem. The LR learns by updating the values of the weights and bias from a training set. Each weight \(w_i\) is associated with one input attribute \(I_i\). The weight represents the importance of the input attribute to the classification decision. The bias terms are added to the weighted inputs.

According to Bayesian statistics, the posterior probability of a random event or an uncertain proposition is the conditional probability assigned after the relevant evidence or the background is taken into account. Let us consider logistic regression model for a two-class \((y_1, y_2)\)
classification task, the posterior probabilities are modelled as [33].

\[ P_r(y_1 | I) = \frac{1}{1 + \exp(-I w)} \]  

(16)

and

\[ P_r(y_2 | I) = 1 - P(y_1 | I) \]  

(17)

After complete training of the weights and bias, the classifier multiplies each input \( I_j \) by its updated weight \( w_j \), sums up the weighted features and adds the updated bias term \( w_0 \) as shown in Algorithm 3.

### Algorithm 3 Logistic Regression Classifier for Cache Placement

1: **Input:** Dataset with input features \( I_j \) and one output label \( y \).
2: **Output:** Classifier model \( k(I, w) \) given in (7).
3: **Initialise** bias \( w_0 = 0 \), weights \( w_j = 0 \), and learning rate \( \alpha = 0.3 \).
4: **Split** dataset into 70\% for training and 30\% for testing.

**Training the model:**

5: **for** \( i = 1 \): Number of epochs do
6: **for** each training instance in training dataset do
7: \hspace{1em} **Compute** \( y' = P_r(y | I) = \frac{1}{1 + \exp(-I w_j)} \)
8: \hspace{1em} **Update** bias and weights
9: \hspace{2em} \( w_0^{t+1} = w_0^t + \alpha(y - y')y'(1 - y') \)
10: \hspace{2em} \( w_j^{t+1} = w_j^t + \alpha(y - y')y'(1 - y')I_j \)
11: **end for**
12: **end for**
13: **Return** model structure with updated bias and weights.

**Testing the model:**

14: **for** each instance in the testing dataset do
15: \hspace{1em} **Read** input attributes \( I_j \)
16: \hspace{2em} **Compute** \( y' = P_r(y | I) = \frac{1}{1 + \exp(-I w_j)} \)
17: \hspace{2em} **if** \( y' \geq 0.5 \) **then**
18: \hspace{3em} **The decision is cache**
19: \hspace{2em} **else if** \( y' < 0.5 \) **then**
20: \hspace{3em} **The decision is not cache**
21: **end if**
22: **end for**

The classifier models described by algorithms 3 define the contents of SBS caches. This means these classifiers define the contents of the solution matrix \( A \) presented in the multi-objective function given in Equation (1).

### 5 | EXPERIMENTAL RESULTS

In this section, we evaluate the performance of the classifier-based supervised machine learning algorithms for cache placement and access via simulations. We compared the performance of the proposed cache placement approaches with popularity, random, weighted sum cache placement algorithms.

#### 5.1 Simulation Setup

Our simulation is implemented on a certain sub area with the area dimension and number of SBSSs described in [34]. We assume that the macrocell includes one MBS, 15 SBSSs, and there are three different paths that users pass through with 11 locations in each path (Figure 7). The results are computed for 10000 randomly generated user locations and 10000 different user requests within one wireless network cell. Each request consists of the requested file ID. The requests of files follow Zipf popularity distribution. A number of experiments are implemented to compare the impact of system parameters on the performance of different cache placement algorithms. During each experiment, one of the parameters is varied and the remaining are fixed. The parameters are illustrated in Table 1. For the weighted-sum approach, we assumed all the parameters have equal weights and the threshold is 0.5 to take the 50\% of the effect of the input attributes.

We assume that the user mobility parameters follow the work in [19]. It is defined as an independent Poisson process, the pairwise contact duration between mobile UT and SBS follows the exponential distribution with parameter \( \lambda^{\text{SBS}} \), such that \( \Lambda(10, 1/100) \) represents the contact rate between mobile UT and SBS. The dataset is divided into 70\% data and 30\% data used for training and testing, respectively. The simulation was implemented using Python on a Jupyter notebook installed on Anaconda environment [35].

#### 5.2 Cache placement algorithms

To evaluate the performance of the proposed classifier-based cache placement algorithm and investigate the relationship

| TABLE 1 Simulation parameters |
|-------------------------------|
| Parameter          | Value                             |
| Number of files in MBS     | 100                               |
| SBSs cache size           | (10\%-100\%) of MBS library       |
| File size                | (1–8) MB                          |
| SBS data rate             | (2–16) Mbps                       |
| SBS communication range   | (50–200) m                        |
| User speed               | (10–30) m/sec                     |
| Number of user requests   | 10,000                            |

Abbreviations: MBS, macro base station; SBSs, small base stations.
among various input attributes, we compare the impact of different variables on six caching techniques. The cache placement techniques are:

5.2.1 | Popularity caching [19]

It assumes that most popular contents are placed in each SBS cache.

5.2.2 | Random caching

It assumes that contents are placed randomly in each SBS cache.

5.2.3 | Weighted-sum approach caching [21]

In our earlier work, we assumed that the contents are placed in SBS caches depending on input attributes such as file popularity, contact duration between UT and SBSs, communication ranges between UT and SBSs, and contact probability between UTs and SBSs.

5.2.4 | Artificial neural network with SGD learning

An ANN is trained on datasets of previous user input attributes, user requests, and the resulted hit or miss to extract information from input attributes and predict cache contents.

5.2.5 | Logistic regression (LR) with SGD learning

It uses optimisation algorithm to train the cache placement model by minimising the loss function in (9). The trained model can be used to predict cache contents.

5.2.6 | Support vector machine (SVM) with SGD learning

It uses SVM to find the hyperplane separating the decision space into two classes as to place or not place the contents that results in cache hit or cache miss. The iterative learning process is implemented using SGD algorithm that aims to minimise the cost function in (9).

5.3 | Learning technique classification models

In this section, the results are evaluated for various learning techniques that are used in this paper: ANN, SVM, and LR. The performance of the classification models is explored for the same dataset using a confusion matrix, that describes the classification model performance. It examines the testing dataset to find if the classification outcome is correct or not. The model accuracy scores are calculated to identify model errors [36].

Figures 4–6 show the confusion matrices for ANN, SVM, and LR classification models, respectively. These figures summarise the percentage of correct and incorrect predictions for each class and the types of errors that are made by the classifiers. The correct estimation is organised along the diagonal direction from top-left to bottom-right of the matrices. There are 23.13%, 21.27%, and 23.09% of the sample cases that belong to not cached category and they are predicted not cached by the ANN, SVM, and LR models, respectively. These samples are true negatives (TN), which means they are predicted to not cache and the correct decision is not to cache. The other correct estimation is the true positive (TP) case, there are 75.35%, 74.64%, and 74.42% of the samples, in which the actual decision was to cache and they are predicted to be cached by the ANN, SVM, and LR models, respectively. This refers to the cases where there were users who requested the files and the files were stored in the caches.

FIGURE 4 Confusion matrix of artificial neural network model

FIGURE 5 Confusion matrix of support vector machine model
By looking at the classification errors, false positive (FP) gives 0.0%, 0.04%, and 1.86% for the ANN, SVM, and LR models, respectively. This means there are some percentage of samples in SVM and LR models that are identified as cache, and they should not actually be cached. From the cache placement perspective, this can cause energy loss and more latency consumed to store contents in some SBS caches that will not be requested from these SBSs and/or will not result in successful transmission to the requested UTs. From this point, ANN can be considered as the best model for estimating cache contents compared with other cache placements algorithms used in this work.

Table 2 describes in more depth the performance of our estimators by applying the proposed ANN, SVM, and LR algorithms on the datasets and comparing the accuracy, precision, recall, and F1 score of the results. The accuracy gives the ratio of the correctly predicted observation to the total observations. Accuracy is defined as (TP + TN)/(TP + FP + FN + TN). Accuracy alone will not give an indication of the performance of the estimator. Analysing other performance metrics will give us more understanding of our model before using it in the actual environment with different cases. The precision indicates how precise our model is out of those predicted positive (cache), how many of these samples are actually positive. Hence, Precision = TP/(TP + FP) and the sensitivity or recall performance index measures the ratio of the correctly predicted positive observation (cache) to the all observations in the actual values. That is, Recall = TP/(TP + FN)

Then, we compute the weighted average of precision and recall to compute the F1 score performance index. F1 score helps to understand the model accuracy better than the accuracy performance index when we have uneven class distribution, where F1 score is defined as

\[
F1 \text{ Score} = \frac{2 \times (\text{Recall} \times \text{Precision})}{(\text{Recall} + \text{Precision})}
\]

**TABLE 2** Performance evaluation for classification models

| Classification performance | Cache hit rate | ANN | SVM | LR | Best score |
|---------------------------|---------------|-----|-----|----|------------|
|                          |               | 0.98| 0.79| 0.81| ANN        |
| Accuracy                  |               | 0.99| 0.98| 0.96| ANN        |
| Precision                 |               | 0.98| 0.98| 0.96| ANN, SVM   |
| Recall                    |               | 0.99| 0.98| 0.96| ANN        |
| F1 score                  |               | 0.99| 0.98| 0.96| ANN        |

Abbreviations: ANN, artificial neural network; LR, logistic regression; SVM, support vector machine.

![Figure 6](image-url) Confusion matrix of logistic regression model

![Figure 7](image-url) A 500 m x 500 m macro cell with one macro base station (MBS), 15 small base stations (SBSs), and three paths
Based on the results provided by Table 2, among the three classification models, ANN has performed best and has achieved the highest score as compared with the other algorithms. However, all the remaining algorithms appear to demonstrate a good performance as well when compared with our previous work, weighted-sum approach.

5.4 | Performance evaluation

In this paper, we show only the results of caching in SBS caches. The six strategies are evaluated and compared by computing the cache hit rate for different experiments. User mobility is considered by taking into account that the users may request files while moving from one location to another location within the wireless network. Users can download files from nearby SBSs without the need to use backhaul link. This will cause cache hit. The cache hit rate indicates the number of cache hits divided by the total number of user requests.

5.4.1 | Impact of cache size

To study the impact of SBS cache size on the cache hit rate, we consider SBS cache size ranges from 10% to 100% of the entire file library size. Figure 8 shows the results of changing SBS cache sizes on the SBS hit rate. In all the cache placement algorithms, increasing the cache size results in an increase of the cache hit rate. This is because, increasing cache size means more files can be cached in SBS caches, which increases the possibility of finding the requested contents. In popularity and random caching, the cache hit rate increases almost linearly with the increase of SBS cache size. The WS performs better than popularity and random caching with near linear increase in the cache hit rate with the increase of SBS cache size.

The performance of the learning techniques ANN, SVM, and LR are very similar and better than that of the other techniques. Their results show that although increasing cache size results in an increase of the cache hit rate, the hit rate is 0.92 in ANN, 0.921 in SVM, and 0.91 in LR even for 10% SBS cache size. This means that the trained models in the three learning techniques were able to extract information from the dataset and give correct prediction for the cache content placement regardless of the cache size.

5.4.2 | Impact of user speed

Figure 9 shows the impact of SBS cache size on hit rate for different user speeds. The figure shows the results for 10 m/sec and 25 m/sec for four cache placement techniques: WS, ANN, SVM, and LR. We observed that at low user speed (10 m/sec), the SVM caching technique outperforms ANN, LR, and WS. When increasing user speed up to 25 m/sec, the ANN gives higher hit rate than the SVM, LR, and WS.
5.4.3 | Impact of SBSs communication range

We explore the impact of changing the SBS communication range from 75 to 200m on cache placement performances shown in Figure 10. As the results show, when the SBSs communication range increases, the SBSs cache hit rates increase in all the cache placement algorithms. This is because as the SBS communication range increases, the number of SBSs that are within the communication range of the moving UT increases. Thus, the possibility of finding the requested contents within the SBSs caches is higher. At lower SBS communication range (75m), the LR approach gives higher hit rate than that of ANN, SVM, and WS caching techniques. After increasing the SBS communication range up to 200m, the ANN outperforms SVM, LR, and WS.

5.4.4 | Impact of SBS Data rate

Figure 11 presents the influence of changing SBS data rate on the cache hit rate. As the results show, for all cache placement techniques, increasing the SBS data rate increases the hit rate, since SBSs can transfer more data during the contact with the UT. Increasing the SBS data rate allows the user to complete downloading the contents within the available time while on the move from one location to another location. When the SBS data rate is 4Mbps, the hit rate in SVM technique outperforms the ANN, LR, and WS caching algorithms. While for higher data rates (16 Mbps), the results of LR show higher hit rate than the ANN, SVM, and WS.

5.4.5 | Impact of file size

Figure 12 shows the impact of changing the file size from 1 to 8 MB on the SBS cache hit rate. As expected, the hit rate decreases when the file size increases. This is because the users are downloading the files while moving from one location to the next along the path. When the file size increases, the users will not be able to complete downloading the whole file during the required time which results in a miss. When the file size is small (1 MB), the ANN gives a higher hit rate than the SVM, LR, and WS caching techniques. After increasing the file size up to 8 MB, the ANN again outperforms SVM, LR, and WS.

Table 3 compares cache hit rate for four cache placement algorithms while changing some system parameters between low and high values. The results show that changing the SBS cache size from low to high slightly increases the cache hit rates for ANN, SVM, and LR models and the ANN results in the highest score. The WS cache placement algorithm is improved when we increase the cache size. This is because we trained the model to place the contents based on the input features by defining the hyperplane parameters that separate the two classes. On the other hand, in WS, all the input attributes are assigned in equal weights in the cache placement model.
**FIGURE 10** Impact of cache size on the hit rate with different values of small base stations communication range

**FIGURE 11** Impact of cache size on the hit rate with different values of small base stations data rate
Changing user speed from low to high impacts all algorithms. Knowing that in our simulation, the speed is only increased to the point that the UT is still able to download the contents from SBS while the user is on the move. The LR is the least sensitive to the change of user speed and results in a higher hit rate. For SBS data rate, changing the rate from low to high values affects the cache hit rates for all algorithms, since with higher data rate, the amount of contents transmitted to the UT increases. The SVM shows least sensitivity to the changes in data rate and the difference between low and high data rates is marginal. In the last experiment, we changed the stored file size and computed the effect of this parameter on the cache placement algorithms. In this case, the ANN has the highest hit rate compared with the remaining techniques as shown in 12.

### 5.4.6 Impact on SBS and MBS energy consumption

Figure 13 shows the impact of changing cache size and cache placement technique on the normalised total energy consumption in SBSs and MBS required to download the files. As the figure shows, the normalised total energy consumption decreases with the increase in the cache size. This is because as the SBSs cache size increases, the cache hit rate increases, which means the contents are mostly downloaded from the SBSs. The increase in energy consumption for transmitting the contents from SBSs to UTs means decreasing the energy consumption for downloading the same contents from the MBS. When comparing the normalised total energy consumption required transferring the contents from SBSs and MBS to UTs between different cache placement algorithms, we can see that the SVM and LR result in lower energy consumption. On the other hand, the WS technique results in higher total energy consumption compared with the learning techniques.
6 | DISCUSSION

In this study, we proposed a new cache placement using supervised learning techniques in MENs. To the best of our knowledge, this is the first work where cache placement is formulated as a binary classification based on multi-objectives optimisation problem to improve the cache hit rates that leads to minimising the total latency of downloading the contents to the end users. Also, the proposed cache placement-based learning techniques minimise the total energy consumption required to download the contents. We analysed three proposed learning technique for the prediction of cache contents, ANN, SVM, and LR. The first part of the experiments measures the accuracy and prediction precision among these three algorithms. For the same datasets, the three algorithms show competitive performance in their computed precision, recall, and F1 score. The accuracy of prediction is higher in ANN compared with SVM and LR. There is no unique classifier that performs best for all situations of cache placement system in MENs. To select one of the three algorithms to model the cache placement automatically for a given datasets, number of input features, and computation resources, we should consider the following:

- If the collected dataset is not large, SVM or LR should be selected, otherwise ANN is selected.
- If the input features increase, consequently increasing the hyperplane dimension, ANN or SVM should be selected, otherwise LR can be selected.
- If the computation resources are limited, LR is selected which can be implemented faster and easier. Otherwise, ANN or SVM is selected.

Further selection criteria can be considered after analysing the results from the second part of the experiments that measures the cache hit rate in MENs while changing the system parameters. Table 3 can assist in the selection of an appropriate learning algorithm based on the changes in SBS cache size, user speed, SBS communication ranges, and file size, as follows:

- If the SBS cache size is low, SVM should be selected, otherwise ANN should be selected.
- If it is known that most of the users are moving at a low speed, ANN should be used, otherwise LR should be used at higher user speed.
- If the SBS communication ranges are low, LR should be used, otherwise ANN should be used for high SBS communication ranges.
- If the SBS data rates are low, SVM should be used, otherwise LR should be used.
- If the file size is large, ANN should be used.

Continuous dataset accumulation is suggested to more understand the effect of classification decisions and the change in system parameters on the latency and the total consumption of the energy required to download the contents in mobile edge networks. Increasing the orders of

![FIGURE 13 Impact of cache size on the normalised total energy consumption with different cache placement algorithms](image-url)
magnitude of training dataset requires adopting new technology such as big data and the development of the proposed algorithms to deep learning-based prediction algorithms. However, there is an implication from using supervised learning techniques, that in some practical applications, collecting datasets from moving users and creating such large dataset while continuous changing the system parameters requires a considerable amount of resources, time, and effort. These resources may not be available in many practical cases which limits the use of deep learning methods. In addition, the increase in the training samples may cause supervised learning technique to fit itself to the statistical noise in the training sample and result in overfitting problem. To overcome these problems, the use of semi-supervised learning is suggested that leverages both labelled and unlabelled data for learning.

7 | CONCLUSION

This work focussed on formulating the cache placement as a classification problem that can be solved using machine learning techniques. Different learning techniques were investigated for understanding the problem and the input attributes that were correlated to the classification decisions. We analysed the characteristics of the input features (attributes) that were related to the decision of cache content placement objectives and their properties. These input attributes were important to form the hyperplane that separated the multi-dimensional space into two decision regions (cache contents and not cache contents). The performance comparison of different machine learning models was carried out with the same datasets. Finally, the cache placement models using artificial neural network, support vector machine, logistic regression, and weighted-sum algorithms were tested with actual data sets of user requests taken from published literature. The test was made by changing one of the system parameters and fixing the other parameters and computing the hit rate to investigate the sensitivity of the classification by the changes in the environment parameters.

It has been noticed that with the changes in system parameters, there were a few limitations and weaknesses in each classifier model used in this research. Generating an automatic algorithm selection that decides which algorithm is executed in each wireless network cell depending on the size of the dataset, the number of input features, the available resources, and values of system parameters is suggested for future work. The automatic selection of the algorithm can be a rule-based technique aiming to combine different learning algorithms.

This work yielded promising results for the formulation of cache placement as a classification problem, but continuous dataset accumulation and adoption of semi-supervised deep learning techniques are also suggested to understand the effect of classification and the change in system parameters on the latency and energy consumption caused by downloading the contents by requested users.

ORCID

Alagan Anpalagan DOI: https://orcid.org/0000-0002-6646-6052
Ahmed S. Khwaja DOI: https://orcid.org/0000-0002-9012-989X

REFERENCES

1. Global mobile data traffic forecast update, 2015–2020. Cisco white paper. (2016)
2. Chen, T, et al.: Multi-objective learning for efficient content caching for mobile edge networks. IEEE International Conference on Computing, Networking and Communications (ICNC), 543–547 (2020)
3. Mohammed, L., Anpalagan, A., Jaseemuddin, M.: Energy and latency efficient caching in mobile edge networks Survey solutions and challenges (2021). TechRxiv. Preprint. https://doi.org/10.36227/techrxiv.14461760.v1
4. Krishnendu, S., Bharath, B.N., Vimal, B.: Cache enabled cellular network: algorithm for cache placement and guarantees. IEEE Wireless Commun. Lett. 8(6), 1550–1554 (2019)
5. Jaeyoung, S., Hojin, S., Wan, C.: Optimal content placement for wireless femto-caching network. IEEE Trans Wireless Commun. 16(7), 4433–4444 (2017)
6. Ramy, A., Majid, M.B., Nicola, M.: Caching at the edge in low latency wireless networks. Wireless Automation as an Enabler for the Next Industrial Revolution, 209–240 (2020)
7. Xihua, L., et al.: Hierarchical edge caching in device-to-device enabled mobile networks: modeling, optimisation, and design. IEEE J Sel Area Commun. 36(8), 1768–1785 (2018)
8. Yecheng, W., et al.: Challenges of mobile edge device caching. IEEE Access, 4, 8938–8947 (2016)
9. Jaeyoung, S., Wan, C.: Mobility-aware content placement for device-to-device caching systems. IEEE Trans Wireless Commun. 18, 3658–3668 (2019)
10. Zhang, R., et al.: Deep reinforcement learning (DRL)-based device-to-device (D2D) caching with blockchain and mobile edge computing. IEEE Trans Wireless Commun. 19, 6469–6485 (2020)
11. Tiankui, Z., et al.: D2D-enabled mobile user edge caching: a multi-winner auction approach. IEEE Trans Veh Technol. 68(12), 12314–12328 (2019)
12. Engin, Z., et al.: Big data caching for networking: moving from cloud to edge. IEEE Commun Mag. 54(9), 36–42 (2016)
13. Junsoo, A., Seung, H.J., Hong-Sik, P.: A novel proactive caching strategy with community-aware learning in CoMP-enabled small-cell networks. IEEE Commun Lett. 22, 1918–1921 (2018)
14. Shuai, Z., et al.: Deep learning based recommender system: a survey and new perspectives. ACM Comput Surv. 52(1), 1–38 (2019)
15. Yuris Mulya, S., et al.: Distributed deep learning at the edge: a novel proactive and cooperative caching framework for mobile edge networks. IEEE Wireless Commun. Lett. 8, 1220–1223 (2019)
16. Ejder, B., Mehdi, B., Mérouane, D.: Living on the edge: the role of proactive caching in 5G wireless networks’. IEEE Commun Mag. 52, 82–89 (2014)
17. Sabrina, M., et al.: Context-aware proactive content caching with service differentiation in wireless networks. IEEE Trans Wireless Commun. 16(2), 1024–1036 (2016)
18. Konstantinos, P, Leandros, T.: Code, cache and deliver on the move: a novel caching paradigm in hyper-dense small-cell networks. IEEE Trans Mobile Comput. 16(3), 675–687 (2016)
19. Min, G., et al.: Green and mobility-aware caching in 5G networks. IEEE Trans Wireless Commun. 16, 8347–8361 (2017)
20. Piao, Z., et al.: Recent advances of edge cache in radio access networks for internet of things: techniques, performances, and challenges. IEEE Internet of Things J. 6(1), 1010–1028 (2018)
21. Mohammed, L., et al.: Mobility-aware latencys-efficient cache placement in mobile edge networks. 22nd IEEE International Conference on High Performance Computing and Communications, HPCC, pp. 778–785. Fiji (2020)
22. Shatarupa, D., Sahu, B., Navrati, S.: Social network aware caching for 5G radio access network. IETE Tech Rev. vol. 34, pp. 52–61. sup1 (2017)

23. Yuanyuan, T., et al.: Femtocaching in wireless video networks: distributed framework based on exact potential game. IEEE/CIC International Conference on Communications in China (ICCC), pp. 1–6. (2016)

24. Rui, W., et al.: Mobility-aware caching for content-centric wireless networks: modelling and methodology. IEEE Commun Mag. 54(8), 77–83 (2016)

25. Timothy M.R., Jasbir, S.A.: The weighted sum method for multi-objective optimisation: new insights. Struct Multidiscip Optimisation. 41(6), 853–862 (2010)

26. Anthony, Q., Andrew, S.J.Y.: Classification for accuracy and insight: a weighted sum approach. Proceedings of the sixth Australasian Conference on Data Mining and Analytics. 70 (2007)

27. Theodoridis, S.: Machine learning: a Bayesian and optimisation perspective. Academic press (2015)

28. Casella, G., Fienberg, S., Olkin, I.: Probability for statistics and machine learning. Springer Texts in Statistics (2002)

29. Dustin, T., Panos, T., Edoardo, M.A.: Stochastic gradient descent methods for estimation with large data sets. arXiv. (2015)

30. Diederik, P.K., Jimmy Ba.: Adam: a method for stochastic optimisation. arXiv. (2014)

31. Tony, V.G., et al.: Benchmarking least squares support vector machine classifiers. Mach Learn. 54(1), 5–32 (2004)

32. Constantinou, P., Petroula, T.: The stochastic gradient descent for the primal l1-SVM optimisation revisited. Joint European Conference on Machine Learning and Knowledge Discovery in Databases, pp. 65–80. Springer, Berlin (2013)

33. Tanya, K., Samuel, B.: Supervised machine learning: Optimisation Framework and applications with SAS and R. CRC Press (2020)

34. Marvin, M., Geoffrey, M.V.: Access and mobility of wireless PDA users. ACM SIGMOB - Mob Comput Commun Rev. 9(2), 40–55 (2005)

35. Fabian, P., et al.: Scikit-learn: machine learning in Python. J. Machine Learning Res. 12, 2825–2830 (2011)

36. Ramalingaswamy, C., Da modar, R.E., Venkataareshhabu, K.: Soft Computing Techniques for Type-2 Diabetes Data Classification. CRC Press (2020)

How to cite this article: Mohammed L, Anpalagan A, Khwaja AS, Jaseemuddin M. Performance of cache placement using supervised learning techniques in mobile edge networks. IET Netw. 2021;1–18. https://doi.org/10.1049/ntw2.12029