Chapter 3

Early classification on UTS

In the previous chapter, we have provided a basic overview of TSC approaches and thoroughly reviewed existing early classification methods. This chapter focuses on developing an early classification model for UTS that provides reliable class prediction early in time.

3.1 3.1 Introduction

The advent of the early classification attracted the researchers more, and that is rising yearly due to its usefulness in a wide range of time-sensitive applications, such as gas leakage prediction [17], electricity demand prediction [18], early disease prediction [67], etc. In such time-sensitive situations, people tend to know the results as early as possible to take necessary actions. However, developing classification methods that can provide early prediction without sacrificing classification accuracy is difficult.

In this direction, the notable works [9, 66, 77, 60, 11], developed for UTS, got much attention in the area of TSC. These works have tried to predict the class label early while maintaining an adequate level of accuracy as desired in time-sensitive applications. A classifier performing early prediction is expected to fulfill two conditions. First, the classifier should provide a reliable prediction as early as possible so that the prediction can be useful for taking further actions. Second, the classifier needs to acquire an accuracy equivalent to an accuracy on full-length time series or defined by the user. However, the main idea is neither to maximize the accuracy nor to minimize the earliness but achieve a good trade-off between them.

In this chapter, we have proposed two early classification approaches. The first method defines the early decision criteria as reliability threshold by discriminating the classes over time. The second method develops the early decision policy by considering trade-off optimization between accuracy and earliness.

3.2 Preliminaries

This section provides basic definitions and notations used in this study.

Definition 3.1 (Dataset) *It is defined as* $\mathcal{D} = \left\{(X^i, y^i)_{1\leq i\leq M}\right\}$ *where* X^i *denotes the* i^{th} *time series in the dataset,* $y^i \in \mathcal{Y}$ *denotes the corresponding class label,* $\mathcal{Y}$ *denotes a set of class labels and* M *is the number of time series in the dataset. Moreover,* $\mathcal{D}_t$ *represents the truncated dataset, in which each* $X \in \mathcal{R}^t$*.*

Definition 3.2 (Traditional classification) *The classifier* ($\mathcal{H}$) *learns the mathematical function* $\mathcal{F}_{full}(\cdot)$ *form training set* $\mathcal{D}$ *that predicts class label of complete sequence* X_T *such that* $\mathcal{H} : \mathcal{F}_{full}(X_T) \rightarrow \mathcal{Y}$*.*

Definition 3.3 (Early classification) *The early classification of time series learns the function* $\mathcal{F}_{early}(\cdot)$ *from training set* $\mathcal{D}$ *which provides the reliable class prediction of* X *at some time point* t^* *such that* $\mathcal{H} : \mathcal{F}_{early}(X) \rightarrow \{\mathcal{Y}, t^*\}$*.*

Definition 3.4 (Entropy) *Let* $P = (p_1, p_2, \ldots, p_n)$ *be a finite discrete probability distribution, that is, suppose* $p_k \geq 0$ *for* $(k = 1, 2, \ldots, n)$ *and* $\sum_{k=1}^{n} p_i = 1$*. The amount of uncertainty of the distribution* P*, usually measured by the quantity* $E\left(P, b\right) =$

$E([p_1, p_2, \ldots, p_n], b)$ [89], and defined by

$$E(P, b) = \sum_{k=1}^{n} p_k log_b p_k \tag{3.1}$$

where b indicates the possible outcomes. If all probabilities in P are equally probable, then $E(P, b)$ will be near to one, which indicates the highest uncertainty in prediction. If the probabilities in P are highly discriminated then $E(P, b)$ will be near to zero, which indicates the highest certainty in prediction.

3.3 Early classification by measuring uncertainty

The basic framework for the early classification of time series has two components, i.e., an early classifier and a decision policy, as explained in the background section of Chapter 2. In this view, we proposed an early classification model by developing a series of Probabilistic Classifiers (PCs) that make the model adaptable to incomplete time series. Moreover, the decision policy is defined by estimating the uncertainty in the probabilistic output of the classifier.

The primary motivation of our proposed model is the recent work [60], in which decision policy has been defined by considering two factors: i) Safeguard points that are estimated based on user-defined parameter. ii) Confidence thresholds that are defined as the difference between the highest and second-highest conditional class probabilities. Hence they did not consider the information of all the classes in decision making. In addition to this, our proposed model has the following significant contributions.

- Select the lenient safeguard point compared to [60], where authors have chosen strict safeguard point for each class separately.
- Define the class-wise confidence thresholds by measuring the uncertainty in the predicted output of classifier.
- The performance of various PCs has been analyzed in the proposed model.

3.3.1 Model description

There are two critical aspects of our proposed approach. The first one is to determine the safeguard point for each class, and the second one is to define the class-wise confidence thresholds at each time point separately. After learning these criteria, the model classifies the new time series when the safeguard and confidence threshold are satisfied simultaneously. The proposed model has two phases: a learning phase and a prediction phase which are explained in detail in the following subsections.

3.3.1.1 Learning phase:

The learning phase is accomplished in three steps. The first step identifies the discriminative time point for each class. This step aims to determine the safeguard points so that after these points, it makes sense to predict the class label and avoid false prediction. The second part deals with uncertainty in decision-making. In this step, we learn the confidence threshold that minimizes the uncertainty in prediction. Finally, when the safeguard points and confidence thresholds for respective classes have been obtained, a set of PCs are trained to classify the new incoming time series. The steps for learning model is provided in *Algorithm* 3.1.

Step 1: Learning safeguard points

In this step, the proposed model analyses behavior of sequence data to identify the class-wise safeguard points so that the time series belonging to a particular category will be classified with acceptable accuracy. Moreover, this step also helps in avoiding unnecessary calculations while classifying incomplete time series. For defining the safeguard point for each class, the steps are as follows:

- First compute the intraclass accuracies from labeled training set $\mathcal{D}$ by adopting K-fold cross-validation process. At each time point t, ad-hoc PCs are trained using (K-1) folds samples and computed intraclass accuracies for another fold. This process is repeated R times to avoid the over-fitting problem.

Algorithm 3.1: Model Training Phase

```
Input: $\mathcal{D} = \{X^i, y^i\}_{1 \le i \le m}$, training dataset,
       $P_{acc} = \{p_1, p_2, \ldots, p_K\}$, accuracy %age for each class,
       $K$ is the number of classes
Output: $\tau$, boolean matrix to store safeguard points,
        $\delta$, real valued matrix for threshold,
        $\mathcal{H}$, a set of PCs
1  for $t \leftarrow 1$ to $T$ do
       /* K-fold cross-validation with repitations */
2      foreach fold do
3          $X_{test}, y_{test} \leftarrow$ training data in current fold
4          $X_{train}, y_{train} \leftarrow$ all training data except current fold
5          $h_t \leftarrow$ trainmodel($X_{train}[1:t]$, $y_{train}$)
6          classes $\leftarrow$ predictClass($h_t, X_{test}$)
7          probabilities $\leftarrow$ predictProbabilities($h_t, X_{test}$)
8          for $k \leftarrow 1$ to $K$ do
9              $intraAcc_{t,k} \leftarrow$ intraclass accuracy for each class $k$
10             Save in vector $intraAcc$
11             probs$\leftarrow$ probabilities for correctly classified samples for $k$
12             Save in matrix $correctProbs_{t,k}$
   /* Find reliable safeguard points Setp 1 */
13 for $k \leftarrow 1$ to $K$ do
14     for $t \leftarrow 1$ to $T$ do
15         accuracy[$t$][$k$]$\leftarrow$ mean($intraAcc_{t,k}$)
16     $desAcc_k \leftarrow P_{acc}[k] * accuracy[T][k]$
17     for $t \leftarrow 1$ to $T$ do
18         if $accuracy[t][k] \ge desAcc_k$ then
19             $t_k^* \leftarrow t$
20             break
21     $\tau[k][t_k^* : T] \leftarrow$ True
   /* Learn the confidence threshold Setp 2 */
22 for $t \leftarrow 1$ to $T$ do
23     for $k \leftarrow 1$ to $K$ do
24         probs $\leftarrow correctProbs_{t,k}$, entropy $\leftarrow$ [ ]
25         for $P$ in probs do
26             entropy.append($E(P, K)$)                    // from Eq. (3.1)
27         $\delta[k][t] \leftarrow$ max(entropy)
   /* Train the classifiers Step 3 */
28 for $t \leftarrow 1$ to $T$ do
29     if True in $\tau[1:K][t]$ then
30         $h_t \leftarrow$ trainModel($\mathcal{D}_t$)
31         Append the trained model $h_t$ in $\mathcal{H}$
```

- Next, at each time point t class-wise mean accuracy is calculated using intraclass accuracies generated under all repetition and fold as shown at line number 17 in *Algorithm* 3.1. Furthermore, a class-wise desired acceptable accuracy is computed based on user defied parameter (P_{acc}) with respect to accuracy on full length sequence data ($accuracy\,[T]$) which is defined as:

$$accDes_k = Pacc[k] * accuracy[T]$$

  Now this $accDes_k$ is used to learn the safeguard points.

- Finally, class-wise safeguard points are identified, which are defined as:

$$t_k^* = min\,(t \leq T | accuracy[t][k] \geq accDes_k)$$

Our safeguard point detection policy is lenient compared to ECDIRE [60]. In ECDIRE, authors have chosen strict class-wise safeguard points, which ensure that after this point, all intraclass accuracy on timeline must be greater than or equal to the desired accuracy. However, in our approach, this is not mandatory. Therefore our proposed criterion helps in improving earliness by ignoring some outlier conditions. For example, in Table 3.1, the lenient safeguard point and strict safeguard point will be t_3 and t_8, respectively, where the desired accuracy is 0.85. In this scenario, strict safeguard point select t_8 due to one noisy/outlier accuracy at t_7. As a result, most of the sample may miss the chance to be classified at point's t_3, t_4, t_5, t_6 and t_7. The learning steps of safeguard point are provided

Table 3.1: Intraclass accuracy at different time points

Timepoint	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8
accuracy	0.67	0.78	0.88	0.90	0.87	0.86	0.83	0.85

in *Algorithm* 3.1 from line number 15 to 23. The time complexity of selecting class-wise safeguard points is $\mathcal{O}(T)$, where T is the length of the complete time series.

Step 2: Learning confidence threshold

This step presents the learning procedure of confidence threshold, which ensures the reliability of class prediction made by the classifiers. It is also a second crucial aspect of our proposed model. Similar to the previous step, PCs are used to generate conditional class probabilities for entire training set. Further, These class probabilities are used to learn the confidence threshold. Here, the confidence threshold has been defined by computing entropy for each sample. The following steps are performed to compute the confidence threshold:

- Firstly, PCs are used to generate conditional class probabilities for each sample in training set using cross-validation process.
- Next, the conditional probabilities for correctly classified samples are grouped class-wise and then calculate the uncertainty information for each sample, using the Eq. (3.1).
- Finally, the confidence threshold is selected as a max value in each class group. The selected threshold is more relaxed because max value is taken based on all correctly classified samples in the group. The other choices can be mean, median, or quartile value based on specific domain knowledge and requirements.

The steps for leaning of confidence threshold are provided in *Algorithm* 3.1 from line number 24 to 30.

Step 3: Training the classifiers

In this final step, the complete training set $\mathcal{D}$ has been used for training the series of PCs. This process first identifies the initial safeguard point on the timeline to avoid the unnecessary training of classifiers. For example, if the initial safeguard point is the last

Algorithm 3.2: Model Prediction Phase

```
Input: $X'$, time series
       $\tau$, Boolean matrix contains safeguard point
       $\delta$, Matrix contains confidance threshold
       $\mathcal{H}$, a set of PCs
Output: predClass, predProbs
1 for $t \leftarrow 1$ to $T$ do
2     predClass $\leftarrow$ predictClass($\mathcal{H}[t]$, $X'$)
3     if $\tau[predClass][t] == True$ then
4         predProbs $\leftarrow$ predictProbabilities($\mathcal{H}[t], X'$)
5         entropy $\leftarrow$ Entropy(predProbs,K)
6         if $entropy \leq \delta[predClass][t]$ then
7             return (predClass, predProbs)
8     return (NA,NA)
```

time point in the timeline, only one classifier needs to be trained. Next, by including this initial safeguard point, PCs are trained at each time point throughout the timeline and saved for use in the prediction phase. The steps for training the classifiers are given in *Algorithm* 3.1 from line number 31 to 34.

3.3.1.2 Prediction phase:

This phase uses the trained model to predict the class of a new time series X'. In this process, X'_t at time t is assigned to corresponding classifier h_t which predict the class $\hat{y}_t$ at time t. The model checks the first criterion, safeguard point for $\hat{y}_t$. If this criterion is satisfied, then the model predicts the class probabilities and calculates the uncertainty in probabilistic output using Eq. (3.1). Now check the second criterion confidence threshold. If evaluated uncertainty is lie under the confidence threshold, then the model halt and return the predicted class $\hat{y}_t$ and time point t. Otherwise, the model waits for more data to be added and repeats the process. The pseudo-code for prediction phase is provided in *Algorithm* 3.2.

3.3.2 Experimental evaluation

We have implemented the proposed model in python on a personal computer with Intel Core i7 @ 3.6 GHz CPU and 16 GB main memory.

3.3.2.1 Dataset

The proposed model has been evaluated on publicly available time series datasets under the UCR time series classification archive [24]. This repository contains synthetic and real-world datasets. Moreover, it provides a separated training set and testing set split for each dataset. We have used the dataset partitions according to the repository without any preprocessing. Fifteen datasets have been considered for evaluation, in which each dataset includes at least five samples per class in the training set.

3.3.2.2 Parameter setting

In this section, we define the parameter settings that are used in experimental work. The model needs to be trained at each time point or defined by the user based on domain knowledge. In this experiment, we have considered $n = 20$ equidistant time points for training a set of classifiers which are defined as $t_i = \lfloor \frac{T}{20} * i \rfloor$ where $i \in [1, 2, \ldots 20]$ and T is length of complete time series. Next user can define the acceptable percentage of full-length accuracy for each class according to their requirement. In this experimental work, we have considered the desired accuracy as 100% of full-length accuracy for each class. Finally, training of the early classification model has been performed by using 5-fold cross-validation to overcome the over-fitting problem.

3.3.2.3 Performance measures

Two performance criterion accuracy and earliness are used to evaluate the proposed model.

Accuracy: It is a performance evaluation measure that is the proportion of correct predictions and total predictions, formally defined as:

$$Accuracy = \frac{1}{N}\sum_{i=1}^{N}(y_i == \hat{y}_i) \tag{3.2}$$

where N is the total number of samples; y_i and $\hat{y}_i$ are the true and predicted class labels respectively for i^{th} sample. Here Eq. (3.2) considers 1 when y_i and $\hat{y}_i$ are equal, 0 otherwise.

Earliness: It measures the average prediction time at which test samples are classified. It is formally defined as:

$$Earliness = \frac{1}{N}\sum_{i=1}^{N}\frac{t_i^*}{T} \times 100 \tag{3.3}$$

where t_i^* is the time point at which prediction is made and T is the length of complete time series.

3.3.2.4 Results analysis

In this section, we have evaluated our proposed model based on two performance measures, accuracy and earliness which are defined earlier in Eq. (3.2), Eq. (3.3) respectively.

Effect of classifiers:

In the proposed early classification model, any PCs can be used. Therefore we have analyzed the effect of different classifiers and its variants which includes: Gaussian Process (GP) [90] with two kernels (*DotProduct, rbf*), Support Vector Machine (SVM)[91] with two kernels (*linear, rbf*), and Naïve Bayes [92].

Results obtained from the experiment for different classifiers are presented in Table 3.2 and Table 3.3 for accuracy and earliness, respectively. In both tables, the best classifier for each dataset is highlighted in boldface. Comparison based on accuracy is defined as higher is the accuracy better is the model. In contrast, for earliness, lower is

Table 3.2: Accuracy of five classifiers on fifteen dataests: GP(DotProduct), GP(rbf), SVM(linear), SVM(rbf), Naive Bayes

Dataset	**GP (DotProd)**	**GP (rbf)**	**SVM (linear)**	**SVM (rbf)**	**Naive Bayes**
CBF	**0.92**	0.50	**0.92**	0.75	0.63
ChlorineConcentration	0.56	0.53	**0.57**	0.54	0.40
Coffee	**0.93**	0.89	0.89	0.86	**0.93**
ECG200	**0.87**	0.78	0.85	0.75	0.76
FaceAll	**0.76**	0.57	0.69	0.61	0.25
FISH	0.81	0.18	**0.83**	0.33	0.41
Gun_Point	0.87	0.87	**0.89**	0.71	0.58
ItalyPowerDemand	**0.92**	0.90	0.90	**0.92**	0.90
MedicalImages	0.67	0.61	**0.70**	0.56	0.46
MoteStrain	**0.85**	0.84	0.83	0.81	0.65
SonyAIBORobotSurface	0.89	0.89	0.88	0.90	**0.92**
StarLightCurves	**0.95**	0.88	**0.95**	0.86	0.80
Two_Patterns	0.88	0.27	**0.89**	0.29	0.26
wafer	0.95	**0.97**	0.96	0.89	**0.97**
yoga	**0.85**	0.71	0.77	0.61	0.50
win	**8**	**1**	**7**	**1**	**3**

Table 3.3: Earliness of five classifiers on fifteen dataests: GP(DotProduct), GP(rbf), SVM(linear), SVM(rbf), Naive Bayes

Dataset	**GP (DotProd)**	**GP (rbf)**	**SVM (linear)**	**SVM (rbf)**	**Naive Bayes**
CBF	28.29	**6.39**	26.53	24.34	20.47
ChlorineConcentration	11.23	**5.00**	9.67	5.56	31.69
Coffee	69.82	68.93	70.18	**66.43**	91.25
ECG200	44.95	14.10	50.55	20.05	**11.95**
FaceAll	36.69	**10.07**	50.55	43.41	12.74
FISH	43.63	**6.46**	54.91	12.69	16.60
Gun_Point	29.33	25.00	34.83	**14.50**	14.77
ItalyPowerDemand	67.45	**59.89**	66.26	77.25	82.71
MedicalImages	11.32	8.32	15.67	**5.70**	18.56
MoteStrain	**15.00**	37.80	15.34	24.32	28.36
SonyAIBORobotSurface	44.66	53.46	44.65	38.82	**22.75**
StarLightCurves	37.87	30.53	30.00	30.00	**10.32**
Two_Patterns	100	7.26	97.42	9.67	**6.98**
wafer	8.47	12.40	11.50	**5.08**	97.36
yoga	100	47.54	39.44	12.75	**6.56**
win	**1**	**5**	**0**	**4**	**5**

the earliness value, better is the model. In Table 3.2, GP(DotProd) and SVM(linear) are the best performers with 8 and 7 wins respectively over 15 datasets, while GP(rbf) and SVM(rbf) are the worst performers with only one win. The performance of classifiers also varies from dataset to dataset; for example, on *ItalyPowerDemand* dataset, all the classifiers attained nearly accuracy between 90% and 92%. On the other hand, the accuracy for *CBF* varies from 50% to 92%, as shown in Table 3.2.

Similarly for earliness parameter, classifiers GP(rbf), NB and SVM(rbf) recorded 5, 5, and 4 wins, respectively, as shown in Table 3.3. GP(DotProd) and SVM(linear) recorded 1 and 0 wins respectively. The results are according to the expectation because accuracy and earliness are conflicting objectives. When accuracy increases, earliness also increases and vice-versa. As a result, GP(DotProd) and SVM(linear) are good choices because our objective is to predict early in time while maintaining good accuracy. However, for a specific application, the choice may vary.

Table 3.4: Accuracy values on fifteen datasets for ECTS, EDSC , RelClass, ECDIRE and Proposed method

Dataset	**EDSC**	**EDSC**	**RelClass**	**ECDIRE**	**Proposed**
CBF	0.85	0.84	0.64	0.89	**0.92**
ChlorineConcentration	0.62	0.52	**0.82**	0.56	0.56
Coffee	0.75	0.75	0.89	**0.96**	0.93
ECG200	0.89	0.85	0.89	**0.91**	0.87
FaceAll	0.76	0.66	0.69	**0.87**	0.76
Fish	0.75	0.68	0.79	**0.81**	**0.81**
GunPoint	0.87	**0.94**	0.91	0.87	0.87
ItalyPowerDemand	**0.94**	0.82	0.85	0.93	0.92
MedicalImages	0.68	0.60	0.67	**0.74**	0.67
MoteStrain	**0.88**	0.78	0.58	0.80	0.85
SonyAIBORobotSurface	0.69	0.80	0.79	0.83	**0.89**
StarLightCurves	0.15	-	**0.95**	**0.95**	**0.95**
TwoPatterns	0.86	0.80	**0.93**	0.87	0.88
wafer	**0.99**	**0.99**	**0.99**	0.97	0.95
Yoga	0.81	0.71	0.83	**0.85**	**0.85**
Win	**3**	**2**	**4**	**7**	**5**

Table 3.5: Earliness values on fifteen datasets for ECTS, EDSC, RelClass, ECDIRE and Proposed.

Dataset	**EDSC**	**EDSC**	**RelClass**	**ECDIRE**	**Proposed**
CBF	71.50	31.85	**23.08**	28.55	28.29
ChlorineConcentration	66.07	33.33	97.59	14.42	**11.23**
Coffee	83.94	54.23	**38.44**	82.14	69.82
ECG200	60.11	**23.24**	68.81	90.1	44.95
FaceAll	63.85	38.94	96.27	56.49	**36.69**
Fish	60.94	47.70	85.42	55.17	**43.63**
GunPoint	46.92	45.58	71.33	32.37	**29.33**
ItalyPowerDemand	79.33	67.08	**35.92**	70.16	67.45
MedicalImages	53.87	31.95	88.96	21.20	**11.32**
MoteStrain	79.06	38.08	90.94	**12.10**	15.00
SonyAIBORobotSurface	68.49	47.03	57.70	62.26	**44.66**
StarLightCurves	82.25	-	90.02	53.10	**37.87**
TwoPatterns	86.52	**64.04**	91.82	98.76	100
wafer	44.38	27.99	30.75	10.87	**8.47**
Yoga	69.41	**38.57**	87.28	100	100
Win	**0**	**3**	**3**	**1**	**8**

Comparisons with other methods:

In the previous section, five classifier's performance have been analyzed based on both the parameters accuracy and earliness. We have found that GP(DotProd) performed slightly better as compared to SVM(linear). Therefore, it is selected as the probabilistic classifier in our proposed model. We have compared our proposed model with baseline methods (ECTS[9], EDSC[66], RelClass[77] and ECDIRE[60]), based on two measures accuracy and earliness as shown in Table 3.4 and Table 3.5. These tables highlighted the best model in boldface for each dataset. Table 3.4, based on winning rules, demonstrated that ECDIRE got the first position and EDSC got the last position by scoring 7 and 2 wins, respectively. Our proposed model got the second rank in this line. The critical difference (CD) diagram also depicts that our method is second best in terms of accuracy, as shown in Figure 3.1.

When the proposed model is compared based on the earliness parameter, it outperformed the other baseline methods, as shown in Table 3.5. The proposed model got the first rank with eight wins, while ECDIRE got the fourth rank with one win. It is clearly

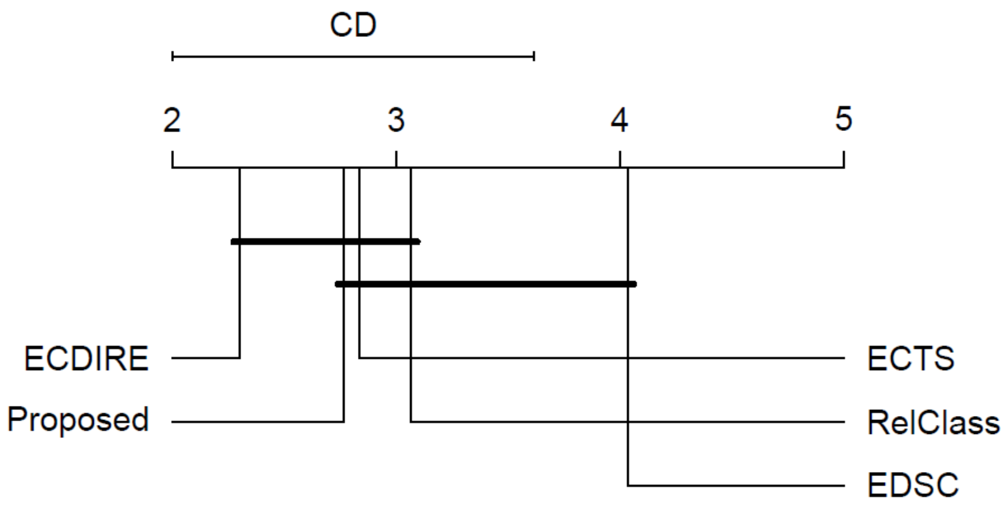


Figure 3.1: Models Comparison: CD diagram for accuracy using Nemenyi post-hoc procedure ($\alpha = 0.05$). The goodness of models is considered as best to worse (*left* to *right*). The bold line shows that methods do not yield statistically significance differences.

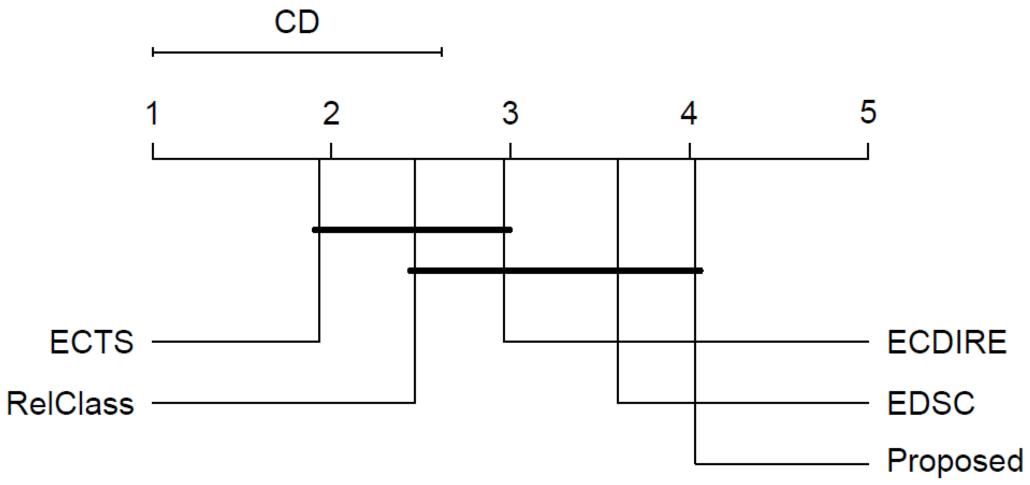


Figure 3.2: Models Comparison: CD diagram for earliness using Nemenyi post-hoc procedure ($\alpha = 0.05$). The goodness of models is considered as best to worse (*right* to *left*). The bold line shows that methods do not yield statistically significance differences.

depicted by the CD diagram in Figure 3.2. As explained in the introduction, accuracy and earliness are two conflicting objectives that need to be balanced. Therefore, while taking both evaluation measures into account, the proposed model performed significantly better than other baseline methods and got the best position in earliness and second-best in terms of accuracy.

3.4 Early classification by learning optimal decision rule

The previous method defines the decision criteria based on user-defined parameters to get the desired accuracy. Moreover, the decision criteria are not optimized by considering trade-off optimization. As the early classification's main objective is to optimize the trade-off between accuracy and earliness, we proposed Early Stopping Rules (ESRs) that consider both the objectives in decision making in this second approach.

The motivation behind our proposed method is the recent work [11] that introduced an optimization-based early classification model by considering both the objectives accuracy and earliness. Basically, this model combined a series of PCs and stopping rules and learn the rules through a genetic algorithm (GA).

3.4.1 Model description

This section presents the model description of the proposed early classification method. The proposed method consists of two phases, i.e., training and prediction. The block diagram of the proposed early classification model is presented in Figure 3.3. In the training phase, the model learns the two components ESRs and PCs. For learning the ESR, the model first generates the posterior class probabilities for each sample in the training set. This is achieved by K-fold cross-validation. The model trains the ad-hoc classifier using K-1 folds and generates the class probabilities for a remaining fold. This process is repeated for K times at each time point t for generating the class probabilities for the complete training set. Next, the model learns the parameters of ESR using the optimization method and saves them to future use in the prediction phase. For learning the PC, the model truncates the training set at each time point t and trains the classifiers using the corresponding truncated training set. In the prediction phase, the incoming time series is presented to the corresponding classifier at time t that returns the class probabilities and class label. Further, the model checks whether ESR is satisfied or not, and based on the result; the next step is taken. A detailed description of the phases is

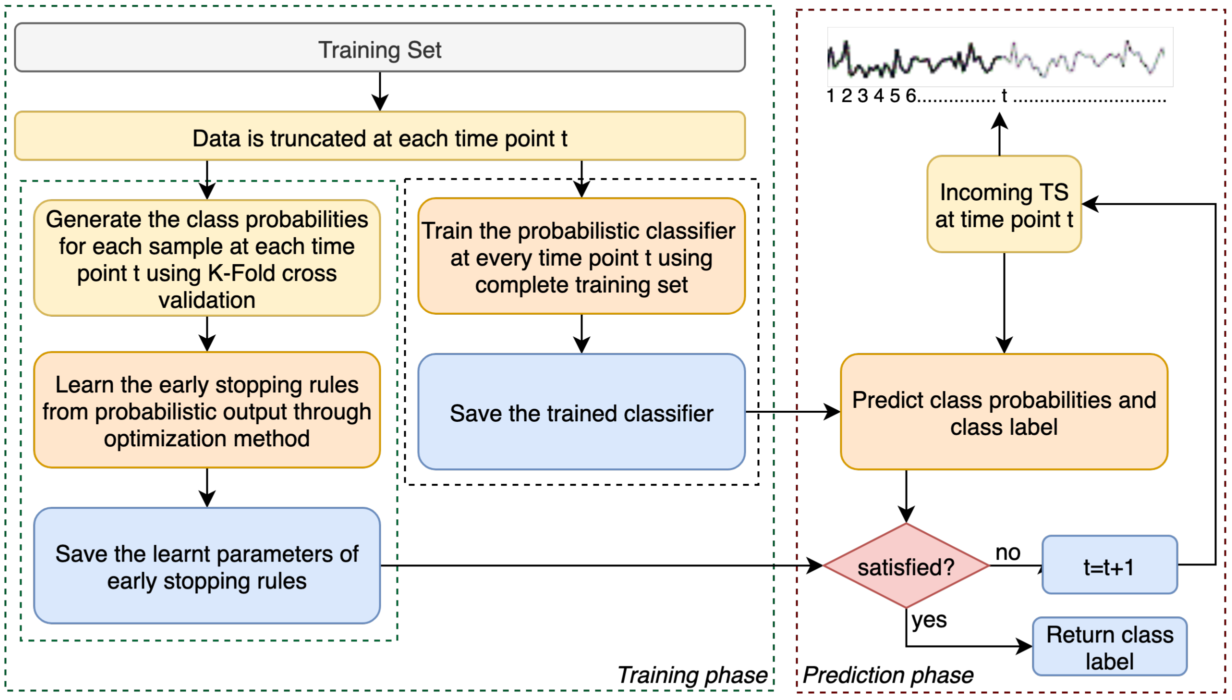

Training Set
Data is truncated at each time point t
Generate the class probabilities for each sample at each time point t using K-Fold cross validation
Learn the early stopping rules from probabilistic output through optimization method
Save the learnt parameters of early stopping rules
Train the probabilistic classifier at every time point t using complete training set
Save the trained classifier
Training phase
1 2 3 4 5 6............... t ....
Incoming TS at time point t
Predict class probabilities and class label
satisfied?
no
t=t+1
yes
Return class label
Prediction phase

Figure 3.3: Block diagram of the proposed early classification model

provided in subsequent subsections.

3.4.1.1 Training phase

In this phase, the step by step training process of model is described using labelled training set $\mathcal{D}$. The training phase is divided in four steps. *Step 1* describes the training process of a series of PCs. *Step 2* and *Step 3* define the ESRs and cost function respectively, used in this model. Finally, *Step 4* explains the learning process of these ESRs.

Step 1: Classifier training

This step explains the training process of a series of PCs $\mathcal{H} = \{h_t\}_{1 \leq t \leq T}$ by considering all the time steps or a subset of time steps, defined by the user. Each classifier h_t is trained using a truncated training set $\mathcal{D}^t$ at time point t. The illustrative example of training a series of classifiers is shown in Figure 3.4.

Step 2: Early stopping rules

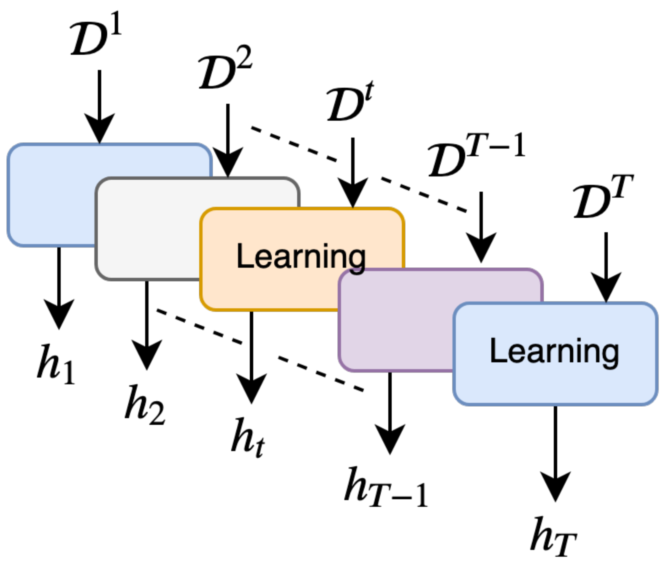


Figure 3.4: Learning of classifier using truncated training set at every time point t

In this step, Three ESRs (ESR1, ESR2 and ESR3) are defined which are the key steps in the decision process of early classification. The first proposed ESR is defined by the following linear equation

$$ESR1_{\boldsymbol{\rho}}(\pi^t, \boldsymbol{\rho}, t) = \begin{cases} 0 & \text{if } \rho_1\pi_1^t + \rho_2\pi_2^t + \rho_3\left(\pi_1^t - \pi_2^t\right) + \rho_4\frac{t}{T} \leq 0 \\ 1 & \text{otherwise} \end{cases} \tag{3.4}$$

where the parameter $\boldsymbol{\rho} = (\rho_1, \rho_2, \rho_3, \rho_4)$ takes the real values between -1 and 1; π^t is the vector of posterior class probabilities, provided by corresponding classifier h_t, over a set of classes K; π_1^t and π_2^t are the two highest posterior probabilities. The ESR defined in Eq. (3.4) is based on the two highest posterior probabilities and the difference of these probabilities at time t. The two highest probabilities (π_1^t, π_2^t) are useful for discriminating the classes over time t [60]. The classes are discriminated well if the differences between π_1^t and π_2^t is high. Hence, the prediction becomes more reliable. Besides, prediction time is also a crucial factor in decision-making because increasing the data points enhances prediction reliability but simultaneously increases the delaying decision cost. Therefore the prediction time is taken into consideration. Thus, we included these four terms in ESR. Moreover, the parameter $\boldsymbol{\rho}$ of ESR is learned using a cost function C_f so that relevance to each term in ESR will be given based on a corresponding ρ_i value.

In the second rule, all the posterior probabilities and difference of the two highest posterior probabilities are included and it is defined as:

$$ESR2_{\boldsymbol{\rho}}(\pi^t, \boldsymbol{\rho}, t) = \begin{cases} 0 & \text{if } \rho_1\pi_1^t + \cdots + \rho_k\pi_k^t + \rho_{k+1}\big(\pi_1^t - \pi_2^t\big) + \rho_{k+2}(\frac{t}{T}) \leq 0 \\ 1 & \text{otherwise} \end{cases} \tag{3.5}$$

The design of the third ESR is influenced by one of the *sequential decision making* techniques known as Wald's Sequential Probability Ratio Test [93] that uses the likelihood ratio of two probabilities in the binary classification problem. This ESR includes four components; highest posterior probability π_1^t, second highest class probability π_2^t, the ratio of two highest probabilities $(\frac{\pi_1^t}{\pi_2^t})$ and normalized time delay $\frac{t}{T}$.

$$ESR3_{\boldsymbol{\rho}}(\pi^t, \boldsymbol{\rho}, t) = \begin{cases} 0 & \text{if } \rho_1\pi_1^t + \rho_2\pi_2^t + \rho_3\big(\frac{\pi_1^t}{\pi_2^t}\big) + \rho_4\frac{t}{T} \leq 0 \\ 1 & \text{otherwise} \end{cases} \tag{3.6}$$

Step 3: Cost function

The aim of the proposed model is to find an optimized ESRs that takes account of earliness and accuracy. In this section, we define a cost function C_f which is required to learn the parameter $\boldsymbol{\rho}$ of ESR through optimization. The proposed cost function is a variant of cost functions, defined in [11] where authors presented cost function with l_0, l_1 regularizations. Here we have presented cost function with l_2 regularization because it is non-sparse in nature and provides the relevance to each term in ESR [94, 95].

$$C_f(\mathcal{D}, ESR) = \frac{1}{|\mathcal{D}|} \sum_{X \in \mathcal{D}} (\alpha C_{miss}(X, ESR_{\boldsymbol{\rho}}) + (1-\alpha) C_{delay}(X, ESR_{\boldsymbol{\rho}})) + \lambda \left\| \boldsymbol{\rho} \right\|_2 . \tag{3.7}$$

In the above cost function, $\alpha \in [0, 1]$ parameter is considered to provide the weight to each objective: accuracy and earliness, and λ is used as a regularization parameter. C_{miss} and C_{delay} are the cost functions for miss-classification and delaying decision respectively. These functions can be defined in many ways. In this, the cost of accuracy is evaluated using $(0-1)$ loss function, where the miss-classification cost of a sample is considered 0 if the predicted class label $(\hat{y})$ matches with the true class label (y) and 1 otherwise. The cost of delaying the decision is defined as $(\frac{t^*}{T})$ where t^* is the earliest time point at which ESR is satisfied.

Step 4: ESR parameter ($\boldsymbol{\rho}$) learning

The goal of this step is to learn the ESR parameter $\boldsymbol{\rho}$ by minimizing the cost function C_f defined in Eq. (3.7) through the optimization method. The nature of the C_f is non-convex or unknown; therefore population-based optimization method PSO has been explicitly used to learn the parameter $\boldsymbol{\rho}$. In this process, the cost is computed for each time series and summed up as the complete cost of the training set, which needs to be minimized. For each X in training set, the posterior probabilities are obtained by classifier h_t at time point t and passed it to ESR with parameter $\boldsymbol{\rho}$. If ESR returns 0 (unsatisfied), then repeat the process by adding the next data point in the X. If the output of ESR is 1 (satisfied), then the current time point t^* and current predicted class at this time point $(\hat{y} = \arg\max_{i \in 1,\ldots,k}\{\pi_i^{t*}\})$ are used to evaluate C_{miss} and C_{delay}.

To provide more generality in training of ESRs, K-fold cross-validation has been

Algorithm 3.3: Generation of class probabilities for training samples

```
Input: $\mathcal{D} = \{(X^i, y^i)_{1 \le i \le m}\}$, training dataset
       $TP = \{t_1, t_2, \ldots, t_n\}$, set of time point
       $K$, number of set of class labels
Output: $P \in \mathbb{R}^{m \times |TP| \times |K|}$, class probabilities
1  for $t$ in $TP$ do
       /* 5-fold cross-validation */
2      foreach fold do
3          $X_{test} \leftarrow$ training data in current fold
4          $y_{test} \leftarrow$ training data labels in current fold
5          $X_{train} \leftarrow$ all training data except current fold
6          $y_{train} \leftarrow$ all training data labels except current fold
7          $h_t \leftarrow$ modelTraining($X_{train}[1:t]$, $y_{train}$)
8          probs $\leftarrow$ predictProbabilities($h_t, X_{test}[1:t]$)
9          for $i$ in fold do
10             P[i,t] $\leftarrow$ probs[i]
```

Algorithm 3.4: Cost Evaluation

```
Input: $\mathcal{D} = \{(X^i, y^i)_{1 \le i \le m}\}$, training set
       $TP = \{t_1, t_2, \ldots, t_n\}$, set of time point
       $P$, class probabilities generated using Algorithm 3.3
       $\rho$, ESR parameter
       $\alpha$, balancing factor between accuracy and earliness
       $\lambda$, regularization parameter
Output: $C_f$, total cost
1  for $i$ in range(M) do
2      $t^* \leftarrow$ T
3      $C_{miss} \leftarrow 0$
4      $C_{delay} \leftarrow 0$
5      for $t$ in $TP$ do
6          $\pi^t \leftarrow P[i,t]$
7          flag $\leftarrow$ ESR($\pi^t$, $\boldsymbol{\rho}$, $t$)
8          if flag == 1 then
9              $t^* \leftarrow$ t
10             break
11     $\hat{y} \leftarrow \arg\max_{j \in 1,\ldots,K}\{\pi_j^{t*}\}$
12     $C_{miss} \leftarrow C_{miss}$ + C($\hat{y} \neq y^i$)
13     $C_{delay} \leftarrow C_{delay}$ + C($\frac{t^*}{T}$)
14 $C_f \leftarrow \frac{1}{M}(\alpha * C_{miss} + (1-\alpha) * C_{delay}) + \lambda * \|\boldsymbol{\rho}\|_2$
```

used as explained initially in Section 3.4.1. In K iteration process, h_t has been trained using all (K-1) subsets of training data and generate class probabilities for another unseen subset of training data at corresponding time step t. In this way, at each time step t, class probabilities for each time series in training data is generated as explained in *Algorithm* 3.3. Further, these probabilities are used to evaluate the C_{miss} and C_{delay} costs for each sample in the training set. Finally, the total cost needs to be minimized to learn the ESR parameter (ρ). The steps for cost evaluation is provided in *Algorithm* 3.4.

3.4.1.2 Prediction phase

In this phase, the trained model is used to predict the class of incoming time series X at a reliable time point t^*, as shown in Figure 3.5. Firstly, the incoming time series at time point t is passed to the classifier h_t that has been trained using the complete training set $\mathcal{D}$ as explained in *Step 1.* Next, the output of h_t is passed to the ESR. If ESR returns 1, the model halt and predicts the class label; otherwise, wait for more data points to be added in the incoming time series.

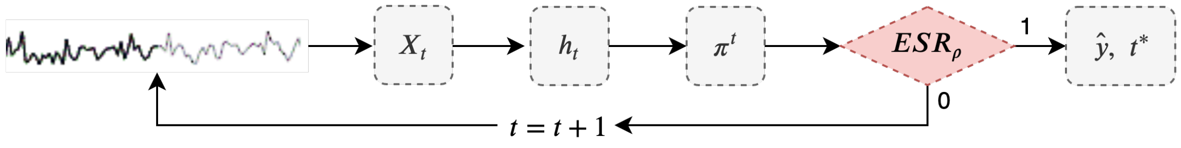


Figure 3.5: Prediction Process

3.4.2 Classification and optimization method

This section provides a brief introduction of the PC and population-based optimization method specifically used in the proposed early classification model.

3.4.2.1 Gaussian process classifier

A Gaussian Process (GP) is a collection of random variables (RV) such that every finite collection of RV has a joint Gaussian distribution [96]. GP is parameterized by its mean $\mu(x)$ and covariance function $cov(x, x')$:

$$f(x) \sim \mathcal{GP}(\mu(x), cov(x, x')) \tag{3.8}$$

In machine learning, GPs are defined over functions and used as priors for Bayesian interference. The prior specifies some properties of function and does not depend on training data. GPs can be used for regression or classification [90]. For classification, first define a continuous latent variable f(x) (eg. f(x)= mx + c) with GP priors. In the next step, the output of the latent function mapped onto [0,1] using some link functions such as probit or logistic. Finally, the resultant function $\mathcal{J}$ is used for prediction. For a given training data $\mathcal{D} = (\mathcal{X}, \mathcal{Y})$, the prediction of test instance (x*, y*) is calculated [90] as:

$$\begin{aligned} \mathcal{J}^* &= p(y^* = +1 | \mathcal{D}, \mathbf{x}^*) \\ &= \int \sigma(f^*) p(f^* | \mathcal{D}, \mathbf{x}^*) df^* \ , \end{aligned} \tag{3.9}$$

where

$$p(f^* | \mathcal{D}, \mathbf{x}^*) = \int p(f^* | \mathcal{X}, \mathbf{x}^*, \mathbf{f}) p(\mathbf{f} | \mathcal{D}) d\mathbf{f}, \tag{3.10}$$

and f^* is the latent variable.

3.4.2.2 Particle Swarm Optimization (PSO)

PSO is a population-based stochastic global optimization technique[97]. In this method, the population set of particles is considered as a swarm. Each particle in the swarm collaboratively exchanges the information to attain the collective goal. Every particle updates its trajectory towards the local best position and global best position, attained by any particle in the neighborhood. In this way, PSO takes advantage of a range of potential solutions, named as population, and detects the optimum solution via cooperation and competition among the population.

Suppose that the search space is d-dimensional and the number of particles in the swarm is s, then every particle in the swarm $P = (p_1, p_2, \ldots, p_s)$ has corresponding d-dimensional vector for position and velocity. The position and velocity of i^{th} particle can be represented by $p_i = (p_{i1}, p_{i2}, \ldots, p_{id})$, $v_i = (v_{i1}, v_{i2}, \ldots, v_{id})$ respectively. At each iteration, PSO updates the particle's velocity and position using Eq. (3.11) and Eq. (3.12) respectively based on particle's best solution $pb_i = (pb_{i1},\, pb_{i2},\, \ldots, pb_{id})$ and global population's best solution $pg_i = (pg_{i1}, pg_{i2}, \ldots, pg_{id})$ [97]:

$$v_{i,j}^{n+1} = wv_{i,j}^{n} + c_1 r_{1j}\left(pb_{i,j}^{n} - p_{i,j}^{n}\right) + c_2 r_{2j}\left(pg_{j}^{n} - p_{i,j}^{n}\right) \tag{3.11}$$

$$p_{i,j}^{n+1} = p_{i,j}^{n} + v_{i,j}^{n+1} \tag{3.12}$$

where $i = 1, 2, \ldots, s$; $j = 1, 2, \ldots, d$; w is *inertia* weight; c_1 and c_2 are the positive real number, known as *acceleration constant*; r_1, r_2 are the random values between 0 and 1; and $n = 1, 2, \ldots,$ determines the iteration number.

3.4.3 Experimental evaluation

This section provides the details of evaluation metrics, datasets description, various parameter settings for analyzing the results. The simulation of the proposed model is conducted on a personal computer with the Intel i7 processor with 3.6 GHz clock frequency and 16 GB main memory.

3.4.3.1 Datasets description

The proposed model is evaluated using publicly available datasets on the UCR Time series classification archive [24]. This archive provides separated training and testing set split for each dataset. We have used the dataset partitions according to the repository without any preprocessing. In this experiment, thirty datasets selected randomly, having at least five samples per class in the training set. These datasets include the cases of binary and multi-class both.

3.4.3.2 Parameter settings

In this early classification model, GP has been used with kernel type (linear inner product) and convergence threshold (1e−8) [98]. A simple way to use a GP classifier is to feed the raw time series as an input vector directly. In the proposed model, distance-based features are used as input to GP, which has shown good performance in [50, 60]. The distance-based feature vector of the raw time series contains the pairwise distance from all the time series in the training set. This transformation enriches the model to use different distance measures inside the model and supports different length time series problems while training the model. For example, a particular time point t, train and test sets are defined as $\mathcal{D}^t_{train} \in \mathbb{R}^{M,t}$, $\mathcal{D}^t_{test} \in \mathbb{R}^{n,t}$ respectively then $\mathcal{D}^t_{train}$ is transformed into $M \times M$ distance matrix by using some distance measure and $\mathcal{D}^t_{test}$ is converted into $N \times M$. Thus, the distance measure is another parameter to the GP classifier that is the euclidean distance in our experiments. Next, the optimization method PSO is used by considering population size (100), max iteration (150), and inertia weight (0.9)[99].

Further, the parameter α, used in the cost function, needs to be decided to incorporate the effect of accuracy and earliness. Here we analyzed the behavior of α on four values (0.6, 0.7, 0.8, 0.9). The value of α has been considered above 0.5 to give more weight to accuracy. The effect of α is analyzed in Section 3.4.3.3. Furthermore, the

Table 3.6: Parameter setting for other early classification models

Early classification models	**Parameter setting**
ECTS_OAE [11]	SR2-CF2, $\alpha = 0.8$
ECDIRE [60]	acc_parc=100%
RelClass [77]	Gaussian Naive Bayes Box, $\tau = 0.5$
EDSC [66]	EDSC-CHE, k-0.5
ECTS [9]	Strict, minimum support=0

next parameter λ in cost function C_f have considered the values (0.003, 0.001, 0.03, 0.01, 0.3, 0.1, 1,3). Finally, a set of time points are considered to train the model. As we have used different types of datasets in our experimental work, having time-series varying lengths from 60 to 570. Therefore, twenty equidistant time points have been considered, that is $(i * \frac{T}{20})$ where $1 \leq i \leq T$. In the real environment, the user can choose any set of time points based on application data domain knowledge.

3.4.3.3 Results analysis

This section provides the analysis of the experimental results on thirty datasets concerning different parameters setting of the proposed model. Also, results are compared with other early classification approaches.

Comparison to other early classification methods:
To analyze the performance of the proposed early classification model, we have compared the proposed strategies (ESR1, ESR2, and ESR3) with five state-of-the-art methods which includes ECTS [9], EDSC[66], RelClass [77], ECDIRE [60] and the best method from [11], denoted by ECTS_OAE. The codes of these methods are publicly available and the summary of parameter setting is provided in Table 3.6.

Table 3.7 and 3.8 presents the experimental results for accuracy and earliness respectively. In the tables, the best performing model for each dataset is highlighted in boldface. The results of proposed ESRs are compared by considering $\alpha = 0.8$. In Table 3.7, it is observed that ECDIRE and RelClass got the first and second place in terms

Table 3.7: Accuracy values for ECTS, EDSC, RelClass, ECDIRE, ECTS_OAE, and Proposed (ESR1, ESR2, ESR3) methods.

Dataset	**ECTS**	**EDSC**	**Rel Class**	**ECDIRE**	**ECTS _OAE**	**ESR1**	**ESR2**	**ESR3**
Beef	0.5	0.23	0.57	0.50	0.73	**0.80**	0.77	**0.80**
CBF	0.85	0.84	0.64	**0.89**	0.87	**0.89**	0.87	**0.89**
ChlorineConcentration	0.62	0.52	**0.82**	0.56	0.57	0.58	0.58	0.57
Coffee	0.75	0.75	0.89	**0.96**	0.93	0.93	0.93	0.89
Cricket_X	0.56	0.52	**0.61**	0.57	0.52	0.54	0.52	0.55
Cricket_Y	0.63	0.57	**0.68**	0.63	0.63	0.61	0.61	0.55
Cricket_Z	0.59	0.50	**0.66**	0.60	0.58	0.60	0.58	0.57
ECG200	0.89	0.85	0.89	**0.91**	0.86	0.86	0.86	0.86
ECGFiveDays	0.62	**0.74**	0.52	0.60	0.59	0.64	0.59	0.59
FaceAll	0.76	0.66	0.69	0.87	**0.88**	0.87	**0.88**	**0.88**
fish	0.75	0.68	0.79	0.81	**0.86**	**0.86**	**0.86**	**0.86**
Gun Point	0.87	**0.94**	0.91	0.87	0.91	0.91	0.91	0.91
InlineSkate	**0.33**	0.18	0.27	0.26	0.26	0.27	0.25	0.28
ItalyPowerDemand	**0.94**	0.82	0.85	0.93	0.87	0.88	0.90	0.88
Lightning-2	0.70	**0.80**	0.62	0.54	0.67	0.67	0.68	0.67
MedicalImages	0.68	0.60	0.67	0.74	0.74	0.74	0.74	**0.75**
MoteStrain	**0.88**	0.78	0.58	0.80	0.79	0.78	0.80	0.76
OSULeaf	0.49	**0.56**	0.48	0.52	0.49	0.50	0.51	0.52
SonyAIBORobotSurface	0.69	0.80	0.79	**0.83**	0.80	0.80	0.80	0.80
SonyAIBORobotSurfaceII	0.85	0.81	**0.88**	0.74	0.77	0.77	0.77	0.75
SwedhLeaf	0.78	0.47	0.83	**0.87**	0.86	0.85	0.86	0.79
synthetic_control	0.88	0.89	**0.98**	0.96	0.95	0.94	0.95	0.94
Trace	0.74	0.80	**0.86**	0.77	0.76	0.76	0.76	0.76
TwoLeadECG	0.73	**0.88**	0.72	0.81	0.76	0.76	0.76	0.78
Two Patterns	0.86	0.80	**0.93**	0.87	0.83	0.87	0.87	0.86
uWaveGestureLibrary_X	0.73	0.54	0.75	**0.77**	0.75	0.75	0.73	0.74
uWaveGestureLibrary_Y	0.63	0.37	0.68	**0.70**	0.67	0.68	0.65	0.68
uWaveGestureLibrary_Z	0.65	0.52	**0.71**	**0.71**	**0.71**	**0.71**	**0.71**	**0.71**
wafer	**0.99**	**0.99**	**0.99**	0.97	0.98	0.98	0.98	0.98
yoga	0.81	0.71	0.83	**0.85**	0.78	0.78	0.78	0.77
Win	**4**	**6**	**10**	**9**	**3**	**4**	**3**	**6**

of accuracy with 10 and 9 winnings out of 30 datasets. The proposed ESR3 and EDSC got third place with 6 winning each. However, the proposed ESR3 and ESR1 got the first and second place with 12 and 7 wins respectively while considering earliness as performance criteria as shown in Table 3.8. In this line, ECDIRE and ECTS are the worst performers in terms of earliness by scoring 0 wins out of 30 datasets.

The comparison of these methods further analyzed using a CD diagram as shown in Figures 3.6 and 3.7. These figures show the average ranking of methods and pairwise significance differences among methods. To draw the CD diagram, this statistical *Ne-*

Table 3.8: Earliness values for ECTS, EDSC, RelClass, ECDIRE, ECTS_OAE, and Proposed (ESR1, ESR2, ESR3) methods.

Dataset	**ECTS**	**EDSC**	**Rel Class**	**ECDIRE**	**ECTS _OAE**	**ESR1**	**ESR2**	**ESR3**
Beef	76.50	93.61	**25.70**	67.78	55.33	51.83	51.00	70.50
CBF	71.50	31.85	**23.08**	28.55	25.03	26.54	25.01	26.34
ChlorineConcentration	66.07	33.33	97.59	14.42	5.54	7.74	8.28	**5.47**
Coffee	83.94	54.23	38.44	82.14	35.36	35.35	35.18	**33.75**
Cricket_X	71.80	52.57	78.68	47.98	30.50	**26.87**	30.19	27.92
Cricket_Y	66.49	45.10	82.36	36.00	35.62	39.51	35.53	**30.41**
Cricket_Z	67.86	56.12	80.36	45.99	30.49	**29.24**	29.97	35.6
ECG200	60.11	23.24	68.81	90.10	10.95	**10.92**	**10.92**	11.25
ECGFiveDays	63.82	53.6	15.84	21.07	7.38	16.03	**7.34**	7.37
FaceAll	63.85	38.94	96.27	56.49	30.79	**30.04**	30.86	30.84
fish	60.94	47.70	85.42	55.17	37.20	35.43	35.26	**33.91**
Gun Point	46.92	45.58	71.33	32.37	26.30	26.47	26.27	**25.6**
InlineSkate	85.08	46.69	87.31	33.83	20.95	**16.64**	23.26	20.50
ItalyPowerDemand	79.33	67.08	35.92	70.16	**32.20**	**32.20**	37.21	34.04
Lightning-2	89.01	55.14	61.16	9.07	**5.00**	**5.00**	9.67	**5.00**
MedicalImages	53.87	31.95	88.96	21.20	**9.37**	12.53	9.82	12.96
MoteStrain	79.06	38.08	90.94	12.10	8.79	7.94	8.67	**6.88**
OSULeaf	76.59	54.38	97.10	47.52	14.55	16.82	**11.20**	20.79
SonyAIBORobotSurface	68.49	47.03	57.7	62.26	5.66	5.73	5.64	**5.63**
SonyAIBORobotSurfaceII	54.54	35.51	70.86	17.66	9.84	10.31	10.20	**9.38**
SwedhLeaf	76.27	62.34	91.96	45.97	27.39	28.12	28.46	**20.37**
synthetic_control	87.88	50.81	71.54	61.92	22.25	22.22	22.12	**21.58**
Trace	50.72	38.63	77.82	41.75	21.95	21.9	**21.85**	21.9
TwoLeadECG	64.43	46.85	83.63	69.38	**17.5**	18.06	18.04	21.74
Two Patterns	86.52	**64.04**	91.82	98.76	85.88	93.37	87.94	94.38
uWaveGestureLibrary_X	85.98	64.30	90.09	74.03	47.84	43.65	**41.47**	43.51
uWaveGestureLibrary_Y	86.29	70.14	81.96	97.09	47.33	50.85	**45.79**	52.92
uWaveGestureLibrary_Z	85.03	61.18	91.80	75.56	**42.13**	44.39	42.74	44.42
wafer	44.38	27.99	30.75	10.87	**6.38**	6.96	6.97	6.80
yoga	69.41	38.57	87.28	100	10.44	10.44	10.45	**10.23**
Win	**0**	**1**	**2**	**0**	**6**	**7**	**6**	**12**

menyi post-hoc test is carried out using *scmamp* package in R by setting significance level 0.05 [100]. Figure 3.6 shows that ECDIRE and EDSC are the best and the worst respectively in an average ranking of accuracy. Also, these two methods are different significantly and statistically. In this line, ESR1 and ESR2 are got second and fourth place in average ranking. Further, while considering second objective earliness, ESR3 and ECTS are the best and worst performers respectively, as shown in Figure 3.7. It can be seen that the proposed ESR1, ESR2 and ESR3 are significantly different from other methods except for ECTS_OAE. However, ESR3 and ESR2 are better in ranking

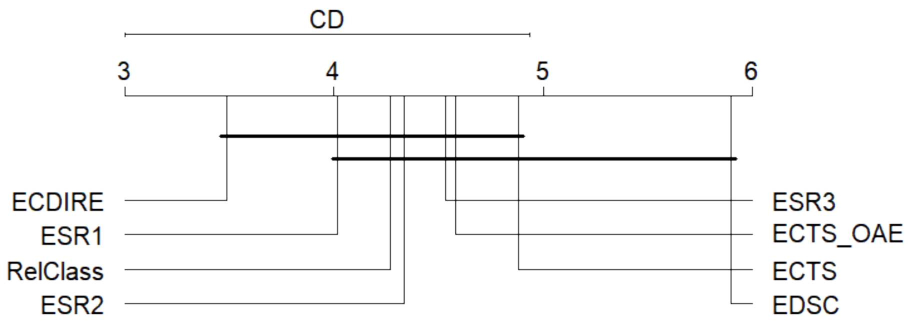


Figure 3.6: CD diagram for accuracy. The goodness of the model is considered as best to worse (left to right), and the bold line shows that methods do not yield statistical significance differences.

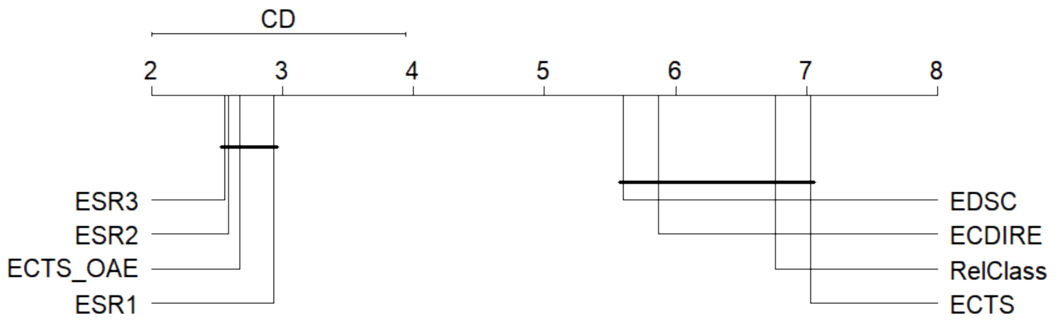


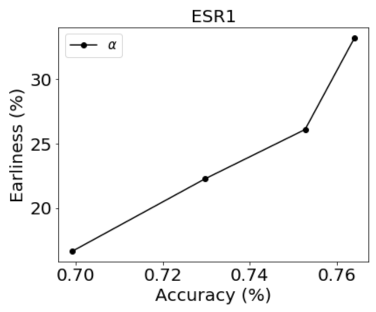
Figure 3.7: CD diagram for earliness. The goodness of the model is considered as best to worse (left to right), and the bold line shows that methods do not yield statistical significance differences.

as compared to ECTS_OAE. So, based on the above observation, It is found that all the state-of-the-art methods except ECTS_OAE are more centric toward accuracy and did not optimize the trade-off between accuracy and earliness well. Thus, the comparison further analyzed between the proposed model and ECTS_OAE, as it is near to the proposed model. It is observed that ESR2 and ESR3 outperform ECTS_OAE in terms of both accuracy and earliness, as shown in Figures 3.6 and 3.7 respectively.

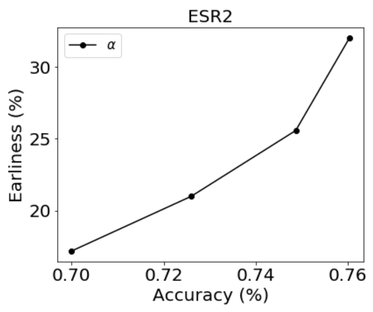
Furthermore, The proposed model has been compared with other methods from the multi-objective perspective based on the Pareto optimality criterion. This criterion states that one method dominated another method if it performs better in one of the objectives without lacking in others. Based on this criterion, Table 3.9 clearly

Table 3.9: Domination counts for the proposed ESR1, ESR2 and ESR3 compared to other methods. The first entry tells about how many times the proposed model dominates other methods, and third entry tells how many times others dominate the proposed model. In between, second entry refers to the draw condition.

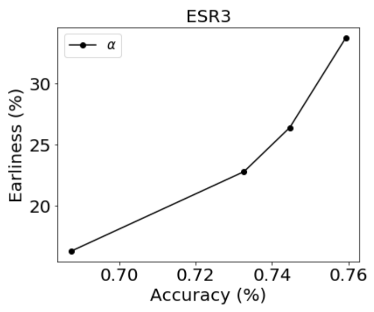
Method	**ECTS**	**EDSC**	**RelClass**	**ECDIRE**	**ECTS_OAE**
ESR1	[18, 12, 0]	[21, 9, 0]	[16, 13, 1]	[15, 14, 1]	[10, 11, 9]
ESR2	[16, 14, 0]	[21, 9, 0]	[13, 17, 0]	[11, 19, 0]	[14, 7, 9]
ESR3	[16, 13, 1]	[19, 11, 0]	[15, 14, 1]	[12, 18, 0]	[8, 17, 5]



(a)



(b)



(c)

Figure 3.8: Accuracy and earliness plot for ESR1, ESR2, and ESR3

demonstrated that the proposed (ESR1, ESR2, and ESR3) dominated the other methods. ESR1 showed better domination counts over ECTS, RelClass, and ECDIRE, while ESR2 showed better domination counts over the ECTS_OAE method. It has been observed that EDSC performs poorly and has zero domination count while ECTS_OAE dominated the proposed ESR1, ESR2, and ESR3 by counts 9, 9, and 5, respectively. Thus, based on the above observation, it can be seen that the proposed model obtained decent performance over the other state-of-the-art methods.

Effect of parameter α:

The parameter α is used to maintain the trade-off between accuracy and earliness. Figure 3.8(a)-3.8(c) plots the average value of earliness and accuracy over all the datasets for different value $\alpha \in (0.6, 0.7, 0.8, 0.9)$. It is observed that accuracy and earliness are increasing by changing the value of α from 0.6 to 0.9. It implies that accu-

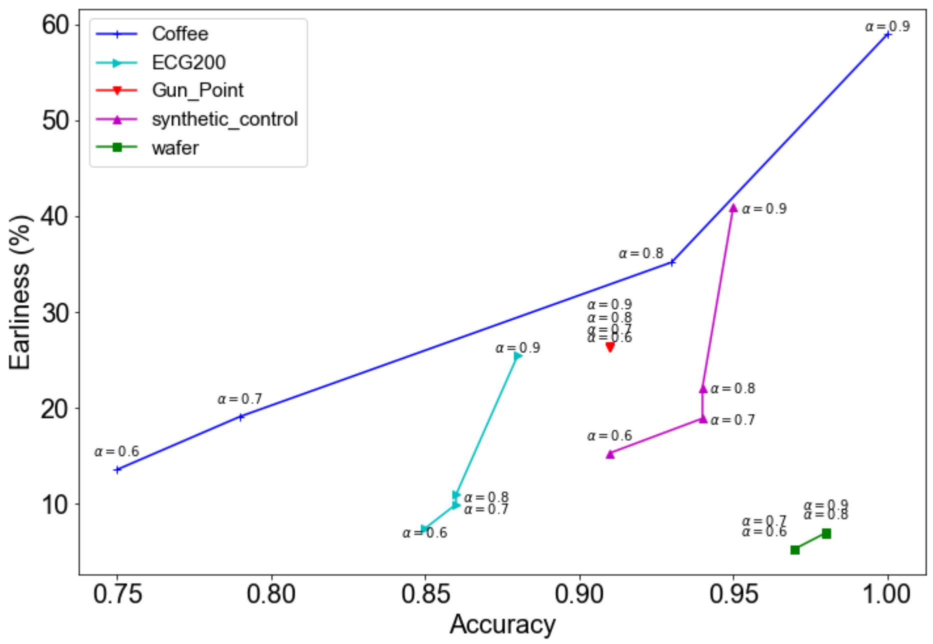
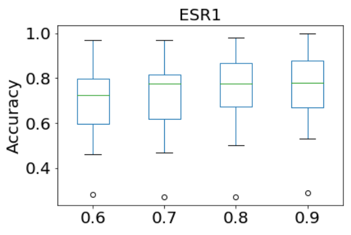
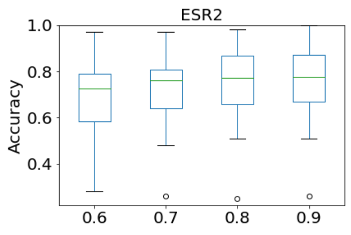


Figure 3.9: Accuracy and earliness plot for five sample datasets includes Coffee, ECG200, Gun_Point, Synthetic Control and Wafer

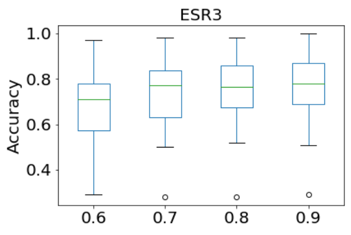
racy improves and earliness worsens when the value of α increases. It is also observed that accuracy and earliness gradually increase while α changing from 0.6 to 0.8, and then earliness increases with the high rate compared to accuracy for $\alpha = 0.9$. It implies that the user needs to pay more delaying decision cost in terms of earliness while selecting $\alpha \geq 0.8$. Thus, the selection of α depends on the requirement of the user. If a high requirement of accuracy, the value of α should be taken high, and if a high requirement of earliness, the value of α should be taken low. However, the behavior of α also depends on the nature of the dataset. To analyze the behavior of α on the individual dataset, we have plotted accuracy and earliness for five sample datasets, as shown in Figure 3.9. For the coffee dataset, Accuracy and Earliness both are gradually increasing while changing the value of α from 0.6 to 0.9. For ECG200 and synthetic control datasets, there is no significant improvement in accuracy when α changes from 0.7 to 0.8. Moreover, earliness becomes high when α changes from 0.8 to 0.9. For the Gunpoint dataset, accuracy and earliness both almost remain constant for all values of



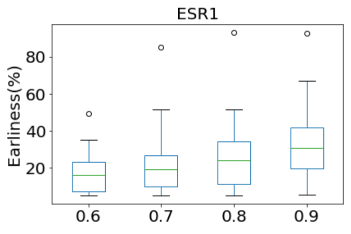
(a)



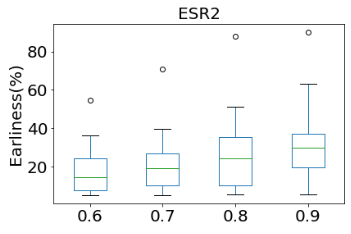
(b)



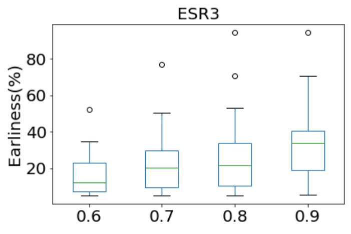
(c)



(d)



(e)



(f)

Figure 3.10: Accuracy and earliness plot over different values of α for ESR1, ESR2 and ESR3

$\alpha \in (0.6, 0.7, 0.8, 0.9)$. Thus, it is concluded that the selection of α strongly depends on the nature of application data and the user's requirement.

Further, the behavior of α analyzed for accuracy and earliness separately. Figure 3.10 displays the accuracy and earliness trends over 30 datasets for $\alpha \in (0.6, 0.7, 0.8, 0.9)$. In Figure 3.10(a)-3.10(c), dots indicate the extreme lowest accuracy obtained on one of the datasets. It is observed that the proposed model achieves the accuracy above 0.50 for all the datasets and the median accuracy value is above 0.79. Even accuracy is more stable for $\alpha \in (0.8, 0.9)$. In Figure 3.10(d)-3.10(f), dots indicate the extremely high earliness value obtained on one of the datasets. Further, it is observed that earliness is increasing for α while changing from 0.6 to 0.9. The proposed model achieves median earliness 17%, 21%, 26%, and 32% approximately for $\alpha \in (0.6, 0.7, 0.8, 0.9)$ respectively. Thus, it can be said that the proposed model also able to classify the time series by utilizing the very little data points in time series approximately 26% for $\alpha = 0.8$.

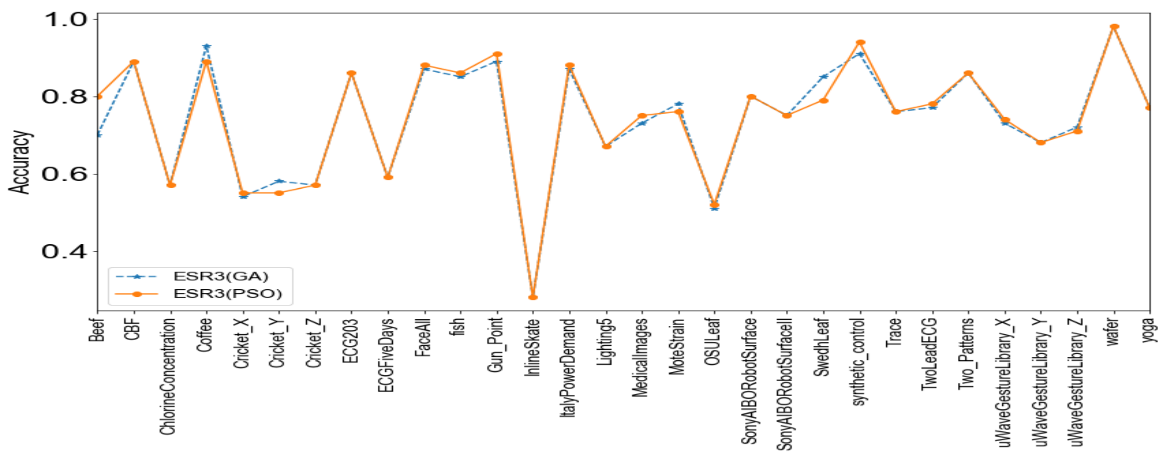
Effect of an optimization method:

In this section, we analyzed the effect of PSO over GA, as used in a previous study[11]. GA is considered with default parameter setting available in R [101]. Figure 3.11 demonstrates the comparative analysis for accuracy, earliness and execution time over 30 datasets by considering ESR3 and $\alpha = 0.8$. It is observed that no method is a straightforward winner when considering accuracy and earliness as a performance measure as shown in Figure 3.11(a)-3.11(b). If the method is superior in accuracy then it is inferior in earliness and vice-versa for most of the datasets, for example, beef, cricket_y, medicalimages, etc. However, PSO is computationally highly effective as compared to GA. Figure 3.11(c) plotted average execution time over the regularization parameter λ. It is observed that PSO is approximately 2 times faster as compared to GA over all the datasets.

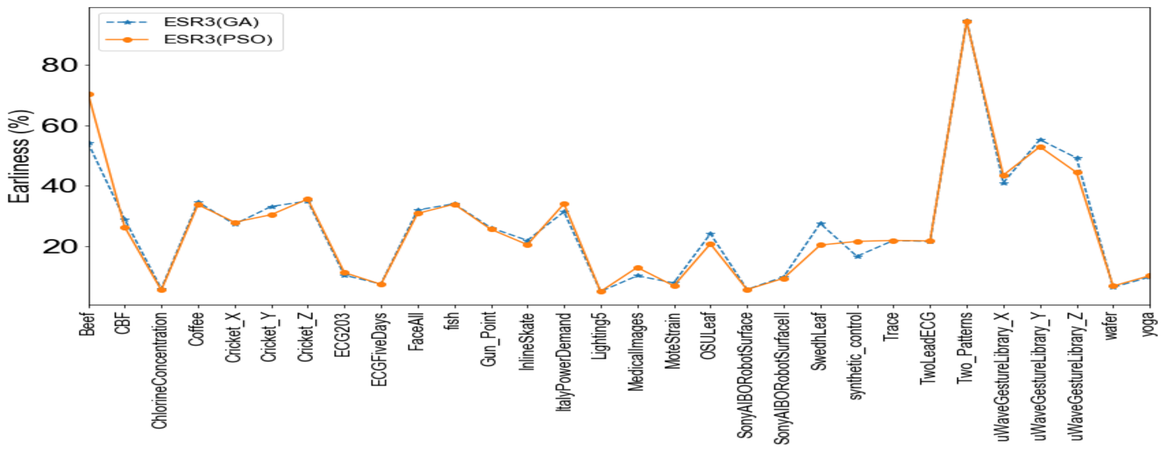
3.4.3.4 Application to early malware detection

Malware is one of the major cyber security threats in the digital world and is on the rise every day. It is malicious programs that were deliberately designed to undermine computer security or harm the computer system like a virus, worm, adware, spyware, Trojan etc. According to AV-TEST [102] report, 15.66 million new malware specimens are reported in November 2019 and in total, 985.12 million specimens for malware are reported by 2019 worldwide. Malware analysis or detection can be done in two ways: static malware analysis and dynamic malware analysis [103].

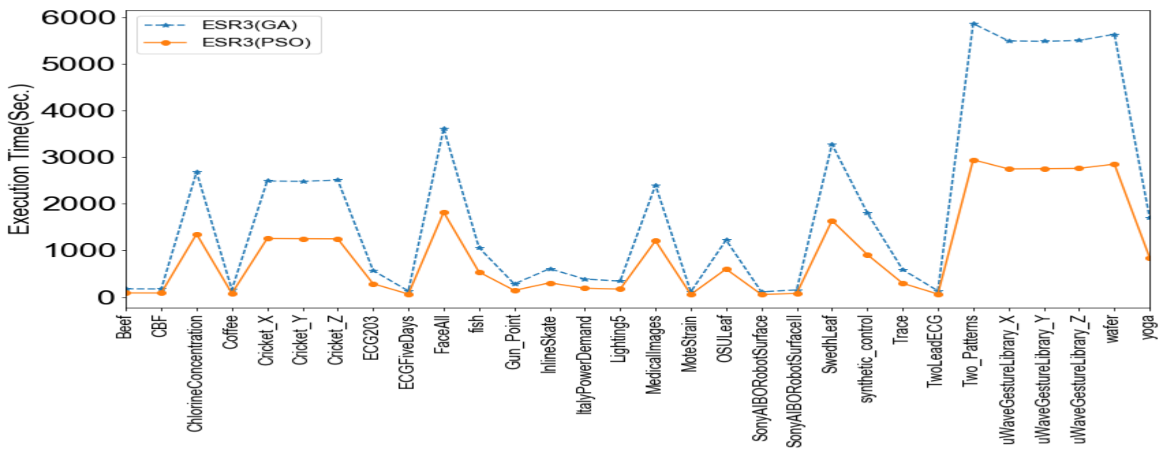
The static malware detection process analyses the code without actually running the code. In addition, static malware detection aims to infer the semantics of a piece of code to determine whether it can perform a malicious activity or not. It can be done quickly by comparing a set of handcraft features from a piece of code to previously identified malware features or signatures. This process makes the static malware analysis, vulnerable to code obfuscation techniques employed by metamorphic and polymorphic malware [104]. Static malware analysis is becoming less effective day by day due to powerful transformation techniques such as manifest cheating, call graph obfuscation,



(a)



(b)



(c)

Figure 3.11: Effect of accuracy, earliness, and execution time for PSO and GA

polymorphism, metamorphism etc. [105] and also not suitable for detecting zero-day malware [106].

On the other hand, behavior analysis (dynamic analysis) during the execution of a file is likely to be very difficult to obfuscate. The dynamic malware detection process monitors the behavior of malicious code at runtime and examines suspicious activities during its interaction with the system. These activities either can be observed in real-time while malicious code is running on a real system or in an isolated closely monitored virtual environment. Moreover, dynamic malware analysis based on behavioral data such as API call sequence is more effective towards the vulnerability of code transformation and also has the capability to detect the zero-day (completely new) malware. However, behavioral data collected during program execution takes a relatively long time as compared to static analysis [107]. Thus, early malware detection in a dynamic environment is highly useful. Thus, our proposed method for early classification on time series is well suited for this problem. Also, as per the best of our knowledge, this is the first work to detect malware early in time-based on the API call sequence.

The model proposed in section 3.4.1 is utilized for the early detection of malware by considering the publicly available malware API call sequences dataset [25]. In this dataset, API call sequences are captured by using open-source malware analysis systems (Cuckoo Sandbox), while running in an isolated virtual environment. These API call sequences are extracted from the parent process to analyze the dynamic behavior. This dataset has considered 307 unique API calls, and each sequence contains only the first 100 non-consecutive repeated API calls. Moreover, it contains malware and goodware API call sequence of 42797 and 1079 samples respectively.

In this experiment, the balanced dataset is obtained by random undersampling of the majority class containing 1079 samples for each class of malware and goodware. Further, training and testing set partition is performed at a ratio of 70% and 30% of the balanced dataset, using stratified splitting. Finally, training and test set contains

Table 3.10: The performance of the proposed early classification model (ESR3) on malware dataset by considering different values of α

α	**Accuracy**	**Precision**	**Recall**	**F1-Score**	**Earliness**
0.60	0.8349	0.8182	0.8611	0.8391	5.24
0.70	0.8349	0.8201	0.8580	0.8386	5.42
0.80	0.8380	0.8230	0.8611	0.8416	7.23
0.90	0.8380	0.8230	0.8611	0.8416	7.23
0.95	0.8812	0.8665	0.9012	0.8835	40.73

the pair of goodware and malware samples of (755, 756) and (324, 325) respectively. The proposed model is tested exhaustively by considering the parameter α (0.60, 0.70, 0.80, 0.90, and 0.95) and the performance are evaluated based on Accuracy, Precision, Recall, F1-Score and Earliness. The high recall value means a small number of false negatives that can be perceived as a high rate of malware detection. Moreover, the high precision value indicates a small number of false positives that is less critical as compared to a false negative, but desirable for malware detection.

Table 3.10 presents the values of performance measures for different values of α. It can be seen that all performance measures are improving by changing the value of the α parameter from 0.60 to 0.95. Moreover, the proposed model is able to detect the malware by utilizing the initial few API call sequences. It is also observed that the proposed model is able to achieve high recall value that further implies the ability of the proposed model to detect the malware effectively at an earlier stage. As it can be seen, at $\alpha = 0.6$, the model achieves 0.86 recall value while utilizing approximately 5.24% of the API call sequence. Also, for $\alpha = 0.95$, the model provides higher accuracy and recall values 0.88 and 0.90 respectively but with the cost of earliness, approximately 40.73% of API call sequences length. Thus, the proposed model shows decent performance for the early detection of malware. Therefore, the proposed method could be a more effective solution for malware detection in a hybrid model.

3.5 Summary

This chapter addressed the problem of early classification on UTS. Specifically, we have proposed two early classification models with different decision policies. The first model has been developed based on a set of PCs and confidence thresholds. In this model, decision policy has two critical aspects, i.e., safeguard points and confidence thresholds. The former criterion reduces the unnecessary overhead of training classifiers and also ensures the user-defined prediction accuracy. The latter criterion checks for prediction reliability at different time points and predicts the class label if the decision criterion satisfies, otherwise discard the prediction and wait for more data points. The confidence threshold has been defined by measuring the uncertainty in the predicted out of correctly classified samples. The proposed model has been evaluated with five PCs on publicly available datasets, and GP with dot product kernel provided the best results. Further, the model has been compared to other state-of-the-art baseline methods. The results demonstrate that the proposed model outperformed the other methods in terms of earliness with comparable accuracy. This method is more inclined towards accuracy and does not take trade-off optimization between accuracy and earliness into consideration.

Therefore, in the second method, the optimization-based approach has been used to learn the optimal decision criteria. The model has been defined based on PCs with ESRs. These ESRs have been learned by minimizing the cost between miss classification costs and delaying decision costs simultaneously. Moreover, GP probabilistic classifier and particle swarm optimization have been used for training the model. The proposed model outperformed the state-of-the-art methods on publicly available thirty datasets and provided a decent balance between earliness and accuracy. Further, we considered the problem of early malware detection to validate its applicability. The proposed model demonstrated excellent performance on the publically available malware dataset having API call sequences. The model was able to classify the malware with 88% accuracy by using approximately 40% of the API call sequence.